Teamcenter 9.1

Workflow Designer Guide
Proprietary and restricted rights notice

This software and related documentation are proprietary to Siemens Product Lifecycle Management Software Inc.

© 2012 Siemens Product Lifecycle Management Software Inc. All Rights Reserved.

Siemens and the Siemens logo are registered trademarks of Siemens AG. Teamcenter is a trademark or registered trademark of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries. All other trademarks, registered trademarks, or service marks belong to their respective holders.
Contents

View the root task .................................................. 2-3
Viewing a subprocess ............................................ 2-3
View task attributes ............................................... 2-3
View task handlers ................................................. 2-5

Adding tasks to workflow process templates .................. 3-1
Workflow task actions and states ............................... 3-1
Task template definitions ....................................... 3-4
Using Do tasks ..................................................... 3-5
Using Acknowledge tasks ....................................... 3-7
Using Review tasks ............................................... 3-9
Using Route tasks .................................................. 3-11
Using generic tasks ............................................... 3-13
Using Impact Analysis tasks .................................. 3-14
Using Prepare ECO tasks ....................................... 3-15
Using Checklist tasks ........................................... 3-17
Using Condition tasks .......................................... 3-19
Using Validate tasks ............................................. 3-24
Using Or tasks ..................................................... 3-38
Using Add Status tasks ......................................... 3-39
Drag and drop a task ............................................. 3-40
Cut and paste a task ............................................. 3-41
Delete a task ....................................................... 3-42

Linking tasks in a workflow process template ............... 4-1
Linking tasks in a workflow process template ............... 4-1
Refreshing Workflow Designer ................................ 4-1
Link tasks manually ............................................. 4-2
Deleting tasks and links ........................................ 4-2
Delete links ........................................................ 4-2
Creating failure paths .......................................... 4-3

Modifying task behavior ........................................ 5-1
Modifying task behavior ........................................ 5-1
Edit task attributes .............................................. 5-2
What are task handlers? ........................................ 5-4
View task handlers .............................................. 5-4
Create task handlers based on existing handlers .......... 5-5
Create new task handlers ...................................... 5-6
Edit task handlers ............................................... 5-7
Delete task handlers ............................................ 5-7

Using workflow templates at multiple Teamcenter sites .... 6-1
Using workflow templates at multiple Teamcenter sites .... 6-1
Replicating workflow templates ............................... 6-1
Importing and exporting workflow templates using Workflow Designer .... 6-3

Workflow handlers ................................................ A-1
Workflow handlers ................................................ A-1
Syntax for handler arguments ................................ A-1
Debugging handler data ......................................... A-13
Contents

Action handlers .......................................................... A-13
Legacy syntax for selected action handlers ...................... A-237
Rule handlers .................................................................. A-262
Handler examples ......................................................... A-341

Glossary ................................................................. B-1

Index ................................................................. Index-1

Figures

Sample EPM workflow process structure ....................... 1-2

Tables

LOV usage ................................................................. A-135
DATA-LOV=lov-name .................................................. A-137
LOV SYS_EPM_get_ecr_relation_data ............................ A-137
# Before you begin

<table>
<thead>
<tr>
<th>Prerequisites</th>
<th>You need Teamcenter administrator privileges to use the Workflow Designer application in <strong>Edit</strong> mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Workflow Designer</td>
<td>Workflow Designer does not need to be enabled before you use it, but during installation, this feature must be selected. If you have trouble accessing Workflow Designer, see your system administrator; it may be a licensing issue.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>You can log on to Teamcenter only once. If you try to log on to more than one workstation at a time, you see an error message.</td>
</tr>
<tr>
<td>Configure Workflow Designer</td>
<td>You can accept Workflow Designer's default configuration settings, or modify its behavior using workflow preferences. For more information about workflow preferences, see the <em>Preferences and Environment Variables Reference</em>.</td>
</tr>
<tr>
<td>Start Workflow Designer</td>
<td>Click <strong>Workflow Designer</strong> in the navigation pane.</td>
</tr>
</tbody>
</table>
What is a workflow?

A workflow is the automation of business procedures in which documents, information, or tasks are passed from one participant to another in a way that is governed by rules or procedures. Teamcenter workflows allow you to manage your product data processes. You can create any type of workflow to accommodate your business procedures.

Example

A pharmaceutical company decides to implement workflows to shorten drug development time, speeding medicines to people in need and strengthening business performance.

After researching workflow solutions and investigating their own company processes, the company determines the need for imaging software to manage the drug test case report forms, data query software to reduce correction time when errors were found in clinical data, and data management software to enforce data integrity. Life cycle data management software such as Teamcenter provides all these solutions in a single product.

A production workflow is created and run in Teamcenter. The workflow is initiated against each product revision (each version of each drug testing). The workflow sends the required forms to the appropriate users, verifies product requirements, routes approvals and notifications to stakeholders, sends cost spreadsheets to the financial department at specific intervals, and rigorously manages the company’s change management processes.

The benefits of automating your business processes include:

• Improved efficiency. The automation of your business processes can result in the elimination of unnecessary steps.

• Better process control. Company business processes are more easily managed with standardized work methods and the availability of audit trails.

• Improved customer service. Consistent business processes increases predictability in levels of response to customers.

• Flexibility. Computer-modeled processes can be quickly and easily redesigned to meet changing business needs.

• Continual process improvement. The resulting focus on business processes leads to their streamlining and simplification.
What is Workflow Designer?

Workflow stems from the concept that all work goes through one or more workflow processes to accomplish an objective. Workflow is the automation of these business processes. Using workflow, documents, information, and tasks are passed between participants during the completion of a particular workflow process.

As a system administrator, use Workflow Designer to design workflow process templates that incorporate your company’s business practices and procedures into workflow process templates. End users use the templates to initiate workflow processes in My Teamcenter and Workflow Viewer.

To design and maintain workflow processes in Workflow Designer, you can perform the following actions:

• Create templates.
• View templates.
• Add tasks to templates.
• Link tasks.
• Modify task behavior.
• Import and export workflow templates.
Workflow process template

A workflow process describes the individual tasks and the task sequence required to model the workflow process. Workflow process templates define a blueprint of a workflow process or task to be performed at your site.

**Browse mode** is the default mode when you first access the Workflow Designer. Click **Browse** to view workflow process data and the details of the workflow process. You cannot make any modifications in this mode.

The graphic-oriented Workflow Designer display allows you to easily browse through the workflow process templates.

- Task flow
- Task hierarchy
- Task attributes
- Task handlers
Workflow task template

A task template is a blueprint of a workflow task. A task is a fundamental building block used to construct a workflow process. Each task defines a set of actions, rules, and resources used to accomplish that task.

<table>
<thead>
<tr>
<th>Task</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Task</td>
<td>Has two options if at least one failure path is configured: <strong>Complete</strong> confirms the completion of a task and triggers the branching to a success path. <strong>Unable to Complete</strong> indicates the task is unable to complete, for various reasons. Uses the <strong>EPM-hold</strong> handler, which stops the task from automatically completing when started.</td>
</tr>
<tr>
<td>Acknowledge Task</td>
<td>Uses the <strong>Acknowledged</strong> and <strong>Not Acknowledged</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td>Review Task</td>
<td>Uses the <strong>select-signoff-team</strong> and <strong>perform-signoffs</strong> subtasks, each of which has its own dialog box. <strong>Wait for Undecided Reviewers</strong> is an option that allows the workflow designer user to set the <strong>Review</strong> task to wait for all reviewers to submit their decisions before completing and following the appropriate path.</td>
</tr>
<tr>
<td>Route Task</td>
<td>Uses the <strong>Review</strong>, <strong>Acknowledge</strong>, and <strong>Notify</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td>Task</td>
<td>Use it as a starting point for creating your own custom tasks, such as tasks to carry your custom forms or other site-specific tasks for users to complete. This task template is synonymous with the <strong>EPMTask</strong> template.</td>
</tr>
<tr>
<td>Impact Analysis Task</td>
<td>Provides an impact analysis for a user to complete for the associated EC revision. The task provides <strong>Reference</strong>, <strong>Impact Analysis Form</strong>, <strong>Viewer</strong>, and <strong>Task Info</strong> tabs. The <strong>Impact Analysis Task</strong> template is for use in EC processes only. It cannot be used on a workflow process.</td>
</tr>
</tbody>
</table>
## Workflow task template

<table>
<thead>
<tr>
<th>Task</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prepare ECO Task</strong></td>
<td>Provides EC requests or EC orders for a user to complete. The task provides ECO Sample and Task Info tabs.</td>
</tr>
<tr>
<td></td>
<td>The Prepare ECO Task template is for use in EC processes only. It cannot be used on a workflow process.</td>
</tr>
<tr>
<td><strong>Checklist Task</strong></td>
<td>Provides a checklist for a user to complete. The checklist form is a form type with a number of logical fields. You can create a custom form type with a site-specific field list using Java code to represent the form as a checklist. The task provides Check List and Task Info tabs.</td>
</tr>
<tr>
<td></td>
<td>The Checklist Task template is for use in EC processes only: it cannot be used on a workflow process.</td>
</tr>
<tr>
<td><strong>Condition Task</strong></td>
<td>Branches a workflow according to defined query criteria. Requires that the succeeding task contains a check-condition handler that accepts a Boolean value of either True or False.</td>
</tr>
<tr>
<td><strong>Validate Task</strong></td>
<td>Branches a workflow along two or more paths. Active paths flowing out of the task are determined by whether specified workflow errors occur.</td>
</tr>
<tr>
<td></td>
<td>Use this task to design workflows around anticipated errors.</td>
</tr>
<tr>
<td><strong>Add Status Task</strong></td>
<td>Creates and adds a release status to the target objects of the workflow process. It is a visual milestone in a workflow process. No dialog box is associated with this type of task.</td>
</tr>
<tr>
<td><strong>Or Task</strong></td>
<td>Continues the workflow process when any one of its multiple task predecessors is completed or promoted. There is no limit to the number of predecessors an or task may have.</td>
</tr>
</tbody>
</table>
Modifying active workflow processes

There are two methods for modifying active workflows in Teamcenter:

- Using Workflow Viewer, you can modify a single active workflow by selecting an object associated with the workflow (typically one of the workflow targets or attachments), using the Send To command to view the active workflow in Workflow Viewer, and then editing the workflow process in Design mode.

  For more information about this procedure, see the Workflow Viewer Guide.

- Using Workflow Designer, you can modify all active workflow processes based on a particular workflow template by selecting the workflow template to be edited and changing to Edit mode to make your edits. (Changing to Edit mode prompts you to take the process template offline; do so) After making your edits, selecting the Set Stage to Available check box displays a dialog box asking if you want to apply your changes to all active workflow processes, and if so, whether you want this update to take place in the background. Run updates in the background if the changes affect a large number of active workflow processes and therefore take considerable time. If you do not run the updates in the background, you can not continue to use the Teamcenter interface until the updates are complete.

  For more information about how updates are processed, see Determining which editing options to use.

By default, this behavior is not enabled. You must configure the ability to modify all active workflow processes by setting the EPM_enable_apply_template_changes site preference to either OPTIONAL or AUTOMATIC.

For more information about setting this preference, see the Preferences and Environment Variables Reference.
Workflow errors

When a **Start** action is triggered on a task, all the handlers placed on that action are executed in the order listed. If all the handlers complete, the state transitions to **Started**, then the handlers on the **Complete** action are executed. When the handlers on the **Complete** action successfully complete, the state transitions to **Completed**.

If all the handlers do not complete successfully, a workflow error is generated. If necessary, an error message appears. For example, if there is an error during workflow process initiation, an error message may state that the action of initiating the workflow process was successful but that a downstream error was generated by one of the root task’s subtasks.

For more information about individual workflow errors, see *Find error codes*.

**Note**

If an error occurs at workflow process creation, the workflow process is not created and the new workflow process does not exist in the database.

If an error occurs on the root task, the workflow process is automatically deleted. A workflow process with no started tasks has no visibility, and without the root task, the workflow process itself cannot be performed.
Mechanics of the Workflow Designer user interface

It is useful to understand Workflow Designer behaviors while editing workflow templates.

Using the Delete key within workflow templates

While working in Edit mode in Workflow Designer, the system interprets the use of the Delete key on your keyboard as an instruction to delete a workflow object.

Caution
Do not use the Delete key to delete characters in text boxes within a workflow template.

To change existing text in a Description or Instructions box:

• Use the Backspace key to remove unwanted text; type new characters into the box

To change text in the Argument and Value(s) boxes in the Handlers dialog box:

• Double-click in the box containing the text you want to modify or delete. Use the Backspace key to remove unwanted text; type new characters into the box.

Note
Handler values are case sensitive and must be accurate to the letter.

Docking and undocking the Handler dialog box

Undocking the Handler dialog box allows you resize it and move it anywhere in the Teamcenter window.

1. Click the Handler button to open the Handler dialog box.
2. Double-click anywhere in the dialog box to undock it.
### Mechanics of the Workflow Designer user interface

#### Example

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Docked</th>
<th>Undocked</th>
</tr>
</thead>
</table>

---

When you leave the **Handler** dialog box docked, you can move between one task's handlers and another task's handlers by selecting a different task in the task hierarchy tree. For example:

1. Click the **Handler** button to open the **Handler** dialog box.
   (Do not undock the dialog box.)

2. Select the **Change Admin II (CM)** task in the task hierarchy tree.
   The dialog box is populated with all the handlers on the **Change Admin II (CM)** task.
   Modify handler arguments and values as needed.

3. Select the **Check Change Type** task in the task hierarchy tree.
The dialog box is populated with all the handlers on the **Check Change Type** task.

Modify handler arguments and values as needed.
Syntax definitions

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

```
harvester_jt.pl [bookmark-file-name bookmark-file-name ...]
    [directory-name directory-name ...]
```

The conventions are:

**Bold**

- Bold text represents words and symbols you must type exactly as shown.
- In the preceding example, you type `harvester_jt.pl` exactly as shown.

**Italic**

- Italic text represents values that you supply.
- In the preceding example, you supply values for `bookmark-file-name` and `directory-name`.

**text-text**

- A hyphen separates two words that describe a single value.
- In the preceding example, `bookmark-file-name` is a single value.

| A vertical bar represents a choice between mutually exclusive elements.

[ ] Brackets represent optional elements.

... An ellipsis indicates that you can repeat the preceding element.

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

```
harvester_jt.pl
harvester_jt.pl assembly123.bkm
harvester_jt.pl assembly123.bkm assembly124.bkm assembly125.bkm
harvester_jt.pl AssemblyBookmarks
```
Workflow Designer interface

Workflow Designer uses the standard Teamcenter rich client interface.

1  **Process Template** box  Lists either all *process* templates or all *task* templates, depending on whether you click the *Process* or *Task* button for the *Template Type*. 
**Workflow Designer interface**

2 Task hierarchy tree Displays hierarchical relationship of all tasks in the selected workflow process template or of all subtasks contained within the selected task template. For example, selecting a container task displays all its subtasks.

Task order within this tree does not indicate task execution order.

3 Process flow pane Displays a graphical representation of all tasks in the selected workflow process template or of all subtasks within a selected task template.

4 Template manager pane Contains elements related to managing the selected workflow process template or task template. The elements displayed depend on the status and configuration of the selected template.

In this example, the template stage is set to **Under Construction**; the template is visible only to users with administrative privileges. When you select this workflow process template, the Set Stage to Available check box displays. This check box does not display when the template stage is set to **Available**.

**Workflow Designer menus**

**File menu**

File menu commands allow you to create workflow process templates and exit Workflow Designer and the rich client user interface.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New Root Template</strong></td>
<td>Allows you to create a new workflow process and task templates.</td>
</tr>
<tr>
<td></td>
<td>For more information, see <strong>New Root Template</strong></td>
</tr>
</tbody>
</table>

**New Root Template**

The following table lists the elements available in the **New Root Template** dialog box.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New Root Template Name</strong></td>
<td>Type a name for the new template. The default name is <strong>New Process #</strong>, where # is the next number available to make the template name unique.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Based On Root Template</strong></td>
<td>Choose a template from the list. The default choice is <strong>Empty Template</strong>, which provides a blank template on which to build.</td>
</tr>
<tr>
<td><strong>Core templates</strong></td>
<td>are delivered with rich client. You can base a new template on a core template or on any other existing workflow process template listed in the list.</td>
</tr>
<tr>
<td><strong>Template Type</strong></td>
<td>Choose the type of template to create:</td>
</tr>
<tr>
<td><strong>Process template</strong></td>
<td>Encompasses an entire workflow process, beginning with the Start action, ending with the Finish action, and containing all required tasks to complete the workflow process.</td>
</tr>
<tr>
<td><strong>Task template</strong></td>
<td>Contains only a single task.</td>
</tr>
<tr>
<td><strong>Task hierarchy tree</strong></td>
<td>Lists the tasks included in the selected template. Tasks are listed in the order they were created. The task hierarchy order will not necessarily be replicated in the process flow pane because of the great flexibility for graphically arranging task flow that the latter provides.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Lists the name of the selected template.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Lists descriptive notes added by users.</td>
</tr>
<tr>
<td><strong>Task Attributes</strong> button</td>
<td>Click to view the task attributes for the selected template.</td>
</tr>
<tr>
<td><strong>Task Handlers</strong> button</td>
<td>Click to view the task handlers for the selected template.</td>
</tr>
</tbody>
</table>
### Workflow Designer interface

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Signoff button</td>
<td>Click to view the task signoff team member profiles for the selected template. When creating a template, you can view, but you cannot modify, the task signoff team member profiles.</td>
</tr>
<tr>
<td>Process flow pane</td>
<td>Shows the task flow of the selected template.</td>
</tr>
<tr>
<td>OK button</td>
<td>Click to finish creating the new template and close the dialog box.</td>
</tr>
<tr>
<td>Apply button</td>
<td>Click to finish creating the new template. The dialog box remains open, allowing you to create additional templates.</td>
</tr>
<tr>
<td>Cancel button</td>
<td>Click to cancel the operation.</td>
</tr>
</tbody>
</table>

### Edit menu

Edit menu commands allow you to build and edit workflow process templates.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>Lists the task templates available in Teamcenter.</td>
</tr>
<tr>
<td>Task 🏃‍♂️</td>
<td>Workflow Designer default template setting. The <strong>Task</strong> template is synonymous with the <strong>EPMTask</strong> template.</td>
</tr>
<tr>
<td>Do Task 🌟</td>
<td>Has two options if at least one failure path is configured: <strong>Complete</strong> confirms the completion of a task and triggers the branching to a success path. <strong>Unable to Complete</strong> indicates the task is unable to complete, for various reasons. Uses the <strong>EPM-hold</strong> handler, which stops the task from automatically completing when started.</td>
</tr>
<tr>
<td>Review Task 📝</td>
<td>Uses the <strong>select-signoff-team</strong> and <strong>perform-signoffs</strong> subtasks, each of which has its own dialog box. <strong>Wait for Undecided Reviewers</strong> is an option to set the <strong>Review</strong> task to wait for all reviewers to submit their decisions before completing and following the appropriate path.</td>
</tr>
<tr>
<td>Add Status Task 🧵</td>
<td>Creates and adds a release status to the target objects of the workflow process. It is a visual milestone in a workflow process. There is no dialog box associated with this type of task.</td>
</tr>
</tbody>
</table>
### Command | Description
--- | ---
Or Task | Inserts an Or task into the workflow process. This task continues the workflow process when any one of its multiple task predecessors is completed or promoted. There is no limit to the number of predecessors an Or task may have.

Acknowledge Task | Inserts an Acknowledge task into the workflow process. This task uses the Acknowledged and Not Acknowledged subtasks, each of which has its own dialog box.

Condition Task | Inserts a Condition task into the workflow process. This task requires that the succeeding task contains a check-condition handler that accepts a Boolean value of either True or False.

Route Task | Inserts a Route task into the workflow process. This task uses the Review, Acknowledge, and Notify subtasks, each of which has its own dialog box.

Validate Task | Inserts a Validate task into the workflow process. This task gives you the ability to respond to errors by providing an alternate path which the workflow process traverses when an error occurs.

Impact Analysis Task | The Impact Analysis Task template is for use in EC processes only. It cannot be used on a workflow process.

Inserts an impact analysis task into the workflow process. This task provides an impact analysis for a user to complete for the associated EC (Engineering Change) revision.

Prepare ECO | The Prepare ECO task template is for use in EC processes only. It cannot be used on a workflow process.

Inserts a Prepare ECO task into the workflow process. This task provides EC Requests or EC Orders for a user to complete.

Checklist Task | The Checklist Task template is for use in EC processes only. It cannot be used on a workflow process.

Inserts a Checklist task into the workflow process. This task provides a checklist for a user to complete. The checklist form is a form type with a number of logical fields. You can create a custom form type with a site-specific field list using Java code to represent the form as a checklist. The task provides Check List and Task Info tabs.

Template Filter | Limits the list of workflow process templates to the user’s group and object type. You cannot apply this to multiple objects, only works on one object at a time. The template filter allows you to associate templates with a designated group and target type. This association is not deep, so if there are subgroups or subtypes, the association must be repeated for these as well.

Mode | Lists the two working modes: Edit and Browse.
Workflow Designer interface

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse</td>
<td>Allows you to view the workflow process data and inspect the details of the workflow process. You cannot make any modifications in this mode.</td>
</tr>
<tr>
<td></td>
<td>Browse mode is the default mode.</td>
</tr>
<tr>
<td>Edit</td>
<td>Allows you to create and edit workflow process templates. To use the Workflow Designer in Edit mode, you need to be a member of the system administration group.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>Access may be restricted even if you have administrator privileges.</td>
</tr>
</tbody>
</table>

View menu

View menu commands allow you to view workflow process template properties.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh All</td>
<td>Discards all changes since the last save.</td>
</tr>
<tr>
<td>Task Properties</td>
<td>Opens the Task Properties dialog box allowing you to view the Task Attributes and Task Handlers dialog box. The Task Signoff dialog box is also available if the selected task is a select-signoff-team task.</td>
</tr>
</tbody>
</table>

Tools menu

Tools menu command allows you to import, export, and purge workflow templates.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export</td>
<td>Exports a workflow template to a file. For more information, see Export workflow templates.</td>
</tr>
<tr>
<td>Import</td>
<td>Imports a workflow template from a file. For more information, see Import workflow templates.</td>
</tr>
<tr>
<td>Purge Templates</td>
<td>Deletes old workflow templates.</td>
</tr>
</tbody>
</table>

Go menu

Go menu commands allow you to maneuver through a workflow process template.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up a Level</td>
<td>Opens the parent task of the currently selected task from the task hierarchy tree.</td>
</tr>
</tbody>
</table>
Workflow Designer interface

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Down a Level</td>
<td>Opens a container task (Review task, Acknowledge task, Route task) currently selected in the task hierarchy tree. If the selected task is not a container task, no task is opened.</td>
</tr>
<tr>
<td>Top Level</td>
<td>Opens the root task of the workflow process.</td>
</tr>
</tbody>
</table>

Workflow Designer buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Properties</td>
<td>Displays the name and description of the selected task.</td>
</tr>
<tr>
<td>Browse Mode</td>
<td>Displays the workflow process templates. Browse mode is available to all users, and provides read-only access of workflow process templates.</td>
</tr>
<tr>
<td>Edit Mode</td>
<td>Edits the workflow process templates.</td>
</tr>
<tr>
<td></td>
<td>To use the Workflow Designer in Edit mode, you must be a member of the system administration group.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>Access may be restricted even if you have administrator privileges.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>The infodba user is an administrative superuser account and should not be used for production work.</td>
</tr>
<tr>
<td>Task Attributes</td>
<td>Displays and opens for edit the named ACL, task type, and quorum requirements for the selected task.</td>
</tr>
<tr>
<td>Task Handlers</td>
<td>Displays and opens for edit task handlers for the selected task.</td>
</tr>
<tr>
<td>Task Signoffs</td>
<td>Displays and opens for edit the group, role, quorum, and number of reviewer requirements for the selected task.</td>
</tr>
<tr>
<td>Task</td>
<td>Inserts an empty task with no handlers into the workflow template for you to customize.</td>
</tr>
<tr>
<td>Do Task</td>
<td>Inserts a Do task into the workflow template. This task has two options, if at least one failure path is configured: Complete confirms the completion of a task and triggers the branching to a success path. Unable to Complete indicates the task is unable to complete, for various reasons.</td>
</tr>
<tr>
<td></td>
<td>This task uses the EPM-hold handler, which stops the task from automatically completing once started.</td>
</tr>
<tr>
<td>Button</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Review Task</td>
<td>Inserts a <strong>Review</strong> task into the workflow template. This task uses the <code>select-signoff-team</code> and <code>perform-signoffs</code> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td><strong>Wait for Undecided Reviewers</strong> is an option that allows the workflow designer user to set the <strong>Review</strong> task to wait for all reviewers to submit their decisions before completing and following the appropriate path.</td>
<td></td>
</tr>
<tr>
<td>Add Status Task</td>
<td>Inserts an <strong>Add Status</strong> task into the workflow template. This task creates and adds a release status to the target objects of the workflow process. It is a visual milestone in a workflow process. There is no dialog box associated with this type of task.</td>
</tr>
<tr>
<td>Or Task</td>
<td>Inserts an <strong>Or</strong> task into the workflow process. This task continues the workflow process when any one of its multiple task predecessors is completed or promoted. There is no limit to the number of predecessors an <strong>Or</strong> task may have.</td>
</tr>
<tr>
<td>Acknowledge Task</td>
<td>Inserts an <strong>Acknowledge</strong> task into the workflow template. This task uses the <strong>Acknowledged</strong> and <strong>Not Acknowledged</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td>Condition Task</td>
<td>Inserts a <strong>Condition</strong> task into the workflow template. This task requires that the succeeding task contains a <code>check-condition</code> handler that accepts a Boolean value of either <strong>True</strong> or <strong>False</strong>.</td>
</tr>
<tr>
<td>Route Task</td>
<td>Inserts a <strong>Route</strong> task into the workflow template. This task uses the <strong>Review</strong>, <strong>Acknowledge</strong>, and <strong>Notify</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td>Validate Task</td>
<td>Inserts a <strong>Validate</strong> task into the workflow template. This task gives you the ability to respond to errors by providing an alternate path which the workflow process traverses when an error occurs.</td>
</tr>
<tr>
<td>Impact Analysis Task</td>
<td>Inserts an impact analysis task into the workflow template. This task provides an impact analysis for a user to complete for the associated Engineering Change (EC) revision.</td>
</tr>
<tr>
<td>The <strong>Impact Analysis Task</strong> template is for use in EC processes only. It cannot be used on a workflow process.</td>
<td></td>
</tr>
<tr>
<td>For more information about working with ECs, see <a href="#">Using Impact Analysis tasks</a>.</td>
<td></td>
</tr>
<tr>
<td>Prepare ECO</td>
<td>Inserts a <strong>Prepare ECO</strong> task into the workflow template. This task provides <strong>EC Requests</strong> or <strong>EC Orders</strong> for a user to complete.</td>
</tr>
<tr>
<td>The <strong>Prepare ECO</strong> task template is for use in EC processes only. It cannot be used on a workflow process.</td>
<td></td>
</tr>
</tbody>
</table>
Workflow Designer interface

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Checklist Task</strong> [ ]</td>
<td>Inserts a <strong>Checklist</strong> task into the workflow template. This task provides a checklist for a user to complete. The checklist form is a form type with a number of logical fields. You can create a custom form type with a site-specific field list using Java code to represent the form as a checklist. The task provides <strong>Check List</strong> and <strong>Task Info</strong> tabs. The <strong>Checklist Task</strong> template is for use in EC processes only. It cannot be used on a workflow process.</td>
</tr>
<tr>
<td><strong>Up a Task Level</strong> ▲</td>
<td>Displays the task one level higher than the current task.</td>
</tr>
<tr>
<td><strong>Down a Task Level</strong> ▼</td>
<td>Displays the task one level lower than the current task.</td>
</tr>
</tbody>
</table>

**Task attributes**

The following table lists the elements available in the **Attributes** pane.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Named ACL</strong></td>
<td>Click to display the <strong>Named ACL</strong> dialog box. For more information about configuring access control lists (ACLs), see the <strong>Security Administration Guide</strong>.</td>
</tr>
<tr>
<td><strong>Task Type</strong></td>
<td>Lists the type of task template assigned to the selected task.</td>
</tr>
<tr>
<td><strong>Icons</strong></td>
<td>Displays the symbol that has been assigned to the selected task. You can also add custom symbols to this list.</td>
</tr>
<tr>
<td><strong>Condition Query</strong></td>
<td>Displays when a <strong>Condition</strong> task is selected. The entry lists the query selected to determine the true and false paths of the Condition path. If a query is not yet defined, it is listed as empty. Click the entry to display the <strong>Condition Query</strong> dialog box, which you can use to change, modify, or delete the defined query.</td>
</tr>
<tr>
<td><strong>Duration</strong></td>
<td>Displays when the selected task contains a defined duration. The entry lists the length of time allowed for the completion of the project. If the task is not completed within the specified amount of time, the task's status changes to late, and the task becomes overdue. Click <strong>Set</strong> to display the <strong>Set Duration</strong> dialog box, which you can use to set a length of time in which the task must be performed. If the task is not completed within the specified amount of time the task's status changes to late, and the task becomes overdue.</td>
</tr>
</tbody>
</table>
Workflow Designer Interface

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipients</td>
<td>Displays the names of users selected to receive program mail when the selected task becomes overdue. Click <strong>Set</strong> to display the <strong>Select Recipients</strong> dialog box, which you can use to select users who will receive program mail if the selected task becomes overdue.</td>
</tr>
<tr>
<td>Show Task in Process Stage List</td>
<td>Displays the task in the <strong>Process Stage List</strong> property for the target object. Tasks in the <strong>Process Stage List</strong> are used to determine the ACL for the target objects.</td>
</tr>
</tbody>
</table>

Task handlers

The following table lists the elements available in the **Handlers** pane of the **Properties** dialog box.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Handler tree</td>
<td>A hierarchical tree consisting of folders representing each of the task actions. Each folder contains the handlers associated with that task action. Action handlers exist as direct descendants of the parent task action folders. Rule handlers exist as children of rules. Rules are direct descendants of task action folders.</td>
</tr>
<tr>
<td>Move Handler Up</td>
<td>Moves the selected handler up within a folder.</td>
</tr>
<tr>
<td>Move Handler Down</td>
<td>Moves the selected handler down within a folder.</td>
</tr>
<tr>
<td>Expand All Folders</td>
<td>Expands all folders.</td>
</tr>
<tr>
<td>Collapse All Folders</td>
<td>Collapses all folders.</td>
</tr>
<tr>
<td>Handler Type</td>
<td>Indicates an action handler or rule handler.</td>
</tr>
<tr>
<td>Quorum</td>
<td>In <strong>Browse</strong> mode, when a predefined rule handler is selected, displays an integer representing the number required for the quorum. In <strong>Edit</strong> mode, you can type or modify the quorum number, but only when a rule handler is selected as the <strong>Handler Type</strong>.</td>
</tr>
<tr>
<td>Task Action</td>
<td>The selected task action from the list receives a handler when it is created.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Action/Rule Handler</td>
<td>Allows you to select an existing handler or define a new one. The system reads the existing handlers from a properties file. Edit this box only when an action handler or rule handler is selected at definition time, and Workflow Designer is in Edit mode.</td>
</tr>
<tr>
<td>Argument</td>
<td>When a predefined handler is selected, this box displays the handler's predefined arguments. In Edit mode, you can add new arguments by clicking the Add button and typing new arguments and values. You can also remove arguments and reorder them using the Remove, ↑, and ↓ buttons.</td>
</tr>
<tr>
<td>Value(s)</td>
<td>When a predefined handler is selected, this box displays the values of the handler's predefined arguments. In Edit mode, you can add new values to arguments by clicking the Add button and typing new arguments and values.</td>
</tr>
<tr>
<td>Create</td>
<td>This button is available only when Workflow Designer is in Edit mode. Click Create to create a new handler using the data currently displayed in the handler display area.</td>
</tr>
<tr>
<td>Delete</td>
<td>This button is available only when Workflow Designer is in Edit mode. Click Delete to remove the selected handler from the current list of handlers for the task.</td>
</tr>
<tr>
<td>Modify</td>
<td>This button is available only when Workflow Designer is in Edit mode. Click Modify to update the selected handler to reflect the data currently displayed in the handler display area.</td>
</tr>
<tr>
<td>Help</td>
<td>Selecting a handler from the Handler box and clicking Help displays the documentation for the selected handler.</td>
</tr>
</tbody>
</table>

**Task signoffs**

The following table lists the elements available in the Signoff Profile pane.
## Workflow Designer Interface

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signoff Profiles</td>
<td>Reflects when the task state is modified as a result of other activities, such as assignment or completion of signoffs. Task state is displayed at run time only. It is never editable from within this pane.</td>
</tr>
<tr>
<td>Group</td>
<td>Lists the user responsible for the task.</td>
</tr>
<tr>
<td>Role</td>
<td>Click the <strong>Named ACL</strong> to display the <strong>Named ACL</strong> dialog box. For more information, see the Security Administration Guide. After typing the required information into the dialog box, the named ACL appears as a label.</td>
</tr>
<tr>
<td>Number of Reviewers</td>
<td>Click the menu to select an to be associated with the selected task.</td>
</tr>
<tr>
<td>Allow sub-group members</td>
<td>Grants members of subgroups permission to sign off instead of members of the designated group.</td>
</tr>
<tr>
<td>Signoffs Quorum</td>
<td><strong>Numeric</strong>: Select numeric and type a whole number or <strong>ALL</strong>. <strong>Percentage</strong>: Enter a percentage. <strong>Wait for Undecided Reviewers</strong>: Select this option ensure all users have a chance to review and comment. Without this option, it is possible for the workflow process to be approved or rejected before all users have had a chance to review and comment.</td>
</tr>
<tr>
<td>Create</td>
<td>This button is available only when Workflow Designer is in <strong>Edit</strong> mode. Click <strong>Create</strong> to create a new signoff profile using the data currently displayed in the signoff profile display area.</td>
</tr>
<tr>
<td>Delete</td>
<td>This button is available only when Workflow Designer is in <strong>Edit</strong> mode. Click <strong>Delete</strong> to remove the selected profile from the current list of signoff profiles for the task.</td>
</tr>
<tr>
<td>Modify</td>
<td>This button is available only when Workflow Designer is in <strong>Edit</strong> mode. Click <strong>Modify</strong> to update the selected to reflect the data currently displayed in the signoff profile display area.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Close  | Clicking **Close** dismisses the dialog box.  
As you make selections, the system enters into the database all selections made within the dialog box. |

**Teamcenter rich client perspectives and views**

Within the Teamcenter rich client user interface, functionality is provided in *perspectives* and *views*. Some applications use perspectives and views to rearrange how the functionality is presented. Other applications use a single perspective and view to present information.

- **Perspectives**
  
  Are containers for a set of views and editors that exist within the perspective.
  
  - A perspective exists in a window along with any number of other perspectives, but only one perspective can be displayed at a time.
  
  - In applications that use multiple views, you can add and rearrange views to display multiple sets of information simultaneously within a perspective.
  
  - You can save a rearranged perspective with the current name, or create a new perspective by saving the new arrangement of views with a new name.

- **Views and view networks**

In some Teamcenter applications, rich client views and view networks let you navigate a hierarchy of information, display information about selected objects, open an editor, or display properties.

  - Views that work with related information typically react to selection changes in other views.
  
  - Changes to data made in a view can be saved immediately.

  - Any view can be opened in any perspective, and any combination of views can be saved in a current perspective or in a new perspective.

  - A view network consists of a primary view and one or more secondary views that are associated. View networks can be arranged in a single view folder or in multiple view folders.

  - Objects selected in a view may provide context for a shortcut menu. The shortcut menu is usually displayed by right-clicking.

For more information about using the shortcut menu, see the *My Teamcenter Guide*. 
Workflow Designer interface

Note
If your site has online help installed, you can access application and view help from the rich client Help menu or by pressing F1. Some views, such as Communication Monitor, Print Object, and Performance Monitor, are auxiliary views that may be used for debugging and that may not be displayed automatically by any particular perspective.

For more information about auxiliary views, see the Client Customization Programmer’s Guide.

For more information about perspectives and views and changing the layout of your rich client window, see the Rich Client Interface Guide.
Chapter

1  Creating workflow process templates

Structuring a workflow process in Workflow Designer .................. 1-1
Building a workflow process template ........................................ 1-2
Create templates in Workflow Designer .................................... 1-3
Delete templates ................................................................. 1-4
Configuring background processing of processes and tasks .......... 1-5
  Configuring background processing of processes and tasks ........ 1-5
  Configure Teamcenter Dispatcher for background processing ...... 1-5
  Configure Security Services for background processing ........... 1-7
  Configure an SOA URL for background processing .................. 1-7
  Configure notifications for background processing .................. 1-8
Editing templates ............................................................... 1-9
  Determining which editing options to use .............................. 1-9
  Editing offline versus online ............................................. 1-10
  Edit workflow templates .................................................. 1-11
  Configure ability to apply template edits to active processes .... 1-12
  Applying template edits to active workflow processes ............. 1-13
  Apply template edits to all active workflow processes .......... 1-14

Creating baseline workflow process templates .......................... 1-14

Create a quick-release workflow process template ...................... 1-15

Creating subprocesses ....................................................... 1-15
  What are subprocesses .................................................. 1-15
  Creating subprocesses from a workflow template .................... 1-16
    Creating subprocesses for multiple targets ....................... 1-17
    Creating subprocesses for assemblies ............................. 1-23
    Creating subprocesses for related objects ....................... 1-24
    Creating ad hoc subprocesses ..................................... 1-24

Core templates .............................................................. 1-25
Chapter

1 Creating workflow process templates

Structuring a workflow process in Workflow Designer

A workflow process describes the individual tasks and the task sequence required to model the workflow process. In Enterprise Process Modeling (EPM), tasks have both temporal (time) and hierarchical (structure) relationships, which allows individual tasks to complete sequentially (serially) or asynchronously (in parallel).

A workflow process template is a blueprint of a workflow process. You can define a specific workflow process by placing workflow tasks (Do task, perform-signoffs task, Route task, and so on) in the required order of performance. You can define additional workflow process requirements (such as placing a status on targets, creating subprocesses, and so on) in the template using workflow handlers. Workflow Designer allows you to create both serial and parallel workflow process templates, and provides you with core templates on which you can build new workflow process templates.

In EPM, each instance of a workflow process uses a workflow process template. This allows each workflow process template to be used as a blueprint for creating multiple workflow processes.

Each EPM workflow process contains a group of nested tasks. The top-level task of every workflow process is referred to as the root task. The following figure shows a sample EPM workflow process structure.
Sample EPM workflow process structure

The root task is the top-level parent task that contains all the other tasks as subtasks. It is the first task executed when a workflow process is initiated and the last task to complete before the workflow process itself is completed.

In the following graphic, the root task is the first task shown in the task hierarchy tree, the **CMII Change Notice** task.

To place handlers on the root task, select the **Start** node and click the **Handlers** button.

**Building a workflow process template**

Workflow process templates define a blueprint of a workflow to be performed at your site.
For example, a workflow process template outlining the workflow process required for a final design review, named Final Design Review, contains the following tasks:

- A **Review** task in which the assigned user is responsible for choosing signoff team members who meet specified group or role requirements. **Wait for Undecided Reviewers** is an option that allows the workflow designer user to set the **Review** task to wait for all reviewers to submit their decisions before completing and following the appropriate path.

- A **Do** task containing instructions to publish the review findings.

- Another **Do** task containing instructions to implement review edits.

- An **Add Status** task which changes the status of the target objects to **Released** upon completion of the workflow process.

After you complete designing a new workflow process template, you must select the **Set Stage to Available** check box to allow the template to display in the Task Hierarchy list.

**Note**

When you close Workflow Designer, the system displays a dialog box listing workflow process templates that are not marked as available. From this dialog box, you can select one or more workflow process templates to be made available to users.

The Task Hierarchy list is accessible from within both Workflow Designer and My Teamcenter. Users initiate a workflow process on a Teamcenter object from within My Teamcenter by choosing File→Workflow process and working through the New Process dialog box.

### Create templates in Workflow Designer

1. Choose **File→New Root Template**.

   The **New Root Template** dialog box appears.

2. In the **New Root Template Name** box, type a template name. The box can contain a maximum of 32 characters.

3. Select **Process** or **Task** for the template type.

4. From the **Based On Root Template** list, select an existing template on which to base the new template.

   The list displays either workflow process templates or task templates.

   When you choose an existing template from the **Based On Root Template** list, workflow process and task information displays for the selected template in the task hierarchy tree and in the viewer. Selecting a task from the displays any subtasks in the viewer; the task name and description are displayed in their respective boxes. This information regarding the existing template is only for viewing within the **New Root Template** dialog box; it cannot be modified.
Chapter 1  Creating workflow process templates

You can also click the Task Attributes, Task Handlers and Task Signoff buttons to view the existing template's task attribute, task handler, and task signoff information.

5. After you view all the necessary template information, click one of the following:
   • **OK** to create the template and close the dialog box.
   • **Apply** to create the template and retain the dialog box so you can create another template.
   • **Cancel** to cancel the operation.

   In Workflow Designer, the Task Hierarchy list displays the template name. The **under construction** symbol to the left of the template name indicates that the template is still in the process of being designed.

   **Note**

   Templates with the **under construction** designation are visible only to system administrators within Workflow Designer. They are not visible to end users who are using the File→New Process option in My Teamcenter to associate a workflow process with objects.

6. Configure your template:
   • **Workflow process template**
     For more information, see Workflow task actions and states and Linking tasks in a workflow process template.
   • **Task template**
     For more information, see Modifying task behavior.

7. Close the **New Root Template** dialog box.

8. Select **Set Stage to Available** in the lower-left panel.

   In Workflow Designer, the Process Template list no longer displays the **under construction** symbol next to the template name.

   In My Teamcenter, the Process Template list, within the New Process dialog box, displays the template name. All users at your site can now access the template.

### Delete templates

1. Select the template you want to delete from the **Process Template** list.

   **Warning**

   Do not delete the Process template. Teamcenter needs this template to create new templates. You cannot create new templates unless you import or create another one with this name.

2. At the top of the task hierarchy tree, select the template.
3. In the toolbar, click the **Delete** button.

4. In the **Delete** dialog box, click **Yes**.

   The selected template is removed from the system.

---

**Configuring background processing of processes and tasks**

Background processing of template edits applied to active workflow processes allows the edits to be performed asynchronously (behind the scenes) without pausing your interaction with Workflow Designer.

Consider the processing time required to apply edits to all active workflow processes based on a particular workflow template. If Workflow Designer is processing edits to 10–20 active workflow processes, as may occur when testing the edits, the Workflow Designer interface does not noticeably slow down. But if the workflow template is in a production environment and has generated hundreds of active templates, processing time can be extensive. Performing the edits in the backgrounds prevents Workflow Designer from pausing until the edits complete.

Background processing of workflow objects requires a four-tier architecture environment. Users running in a two-tier environment can successfully submit requests for asynchronous processing if there is a four-tier Teamcenter environment available to accept the request.

Configuring background processing of workflow objects requires the following configuration tasks:

- Configure Teamcenter Dispatcher for background processing
- Configure Security Services for background processing
- Configure an SOA URL for background processing
- Configure notifications for background processing

**Configure Teamcenter Dispatcher for background processing**

Background processing also requires that Teamcenter Dispatcher be enabled and configured for background processing.

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

   ```xml
   <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>
   ```
Chapter 1  Creating workflow process templates

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

   **Note**
   
   Skip this step if you use TEM to install and configure this service.

3. Set the **maxlimit** attribute to the maximum number of requests your server manager pool can process simultaneously.

   For example:

   ```xml
   <AsyncService provider="SIEMENS" service="asyncservice" maxlimit="2" isactive="true"/>
   
   By default, the Dispatcher module runs only one request of a particular type at a time, which limits your throughput for test cases of submitting numerous requests.

4. Set the cleanup intervals in the (in minutes) in the **dispatcher_root>/Dispatcher/DispatcherClient/conf/Service.properties** file.

   For example, the following settings direct the Dispatcher to check the database and staging directory every 2 hours and to clean up successful and unsuccessful requests when they become 3 days old:

   ```properties
   Service.RequestCleanup.Successful.Interval=120
   Service.RequestCleanup.Successful_THRESHOLD=4320
   Service.RequestCleanup.Unsuccessful.Interval=120
   Service.RequestCleanup.Unsuccessful_THRESHOLD=4320
   
   By default, the Dispatcher checks the database and staging area every 5 minutes and cleans up successful and unsuccessful requests when they become three days old.


   For example, the following setting sets the polling interval to 5 seconds:

   ```properties
   Service.PollingInterval=5
   
   By default, the Dispatcher client pools for new requests every 60 seconds.

6. Edit the **CHANGE_ME** properties in the **asyncservice.bat** (Windows) or **asyncservice.sh** (UNIX) file from the **Dispatcher\Module\Translators\asyncservice\** directory.

   **Note**
   
   Skip this step if you use TEM to install and configure this service.

7. (Optional) Reset the **async_invoker** retry count.
By default, if async_invoker cannot connect to the destination four-tier system, it retries 60 times, one time every 60 seconds. If it has not connected after 60 attempts, it fails.

To reset the retry count or interval, use the preferences_manager utility to import and set the following preferences:

- preferences_manager -u=infodba -p=infodba -g=dba -mode=import
  -preference=ASYNC_connection_retries -scope=SITE -values=1
  -action=OVERRIDE

- preferences_manager -u=infodba -p=infodba -g=dba -mode=import
  -preference=ASYNC_connection_retry_interval -scope=SITE
  -values=10 -action=OVERRIDE

Configure Security Services for background processing

Background processing (asynchronous functionality) supports Security Services for authentication of the asynchronous session in BACKGROUND and BLOCKING mode.

When calling requests on a different site, both the calling and destination site must be using the same Security Services directory. In addition, Teamcenter and Security Services must be configured to define a long time-out period for asynchronous requests.

1. In the Security Services LDAP directory, define a pseudo-application ID for each Teamcenter application ID, with the original application ID and the suffix Async.

   For example, if the Teamcenter application ID is Tc1, define a pseudo-application as Tc1Async.

   Configure this pseudo-application with the desired long time-out period (in seconds) for asynchronous requests.

   For example, if all asynchronous requests are to be executed in the same day, set the time-out value to 60*60*24=86400.

2. Determine the mediator password for the Security Services installation. This value must be installed as an encryption key in the Teamcenter database. Run the install_encryptionkeys utility as follows, and enter the mediator password when prompted:

   install_encryptionkeys -u=infodba -p=password -g=dba
   -f=install_mediator_key

   When the caller calls an asynchronous request in BACKGROUND mode, the native C++ SOA framework obtains a 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 calls the request, async invoker uses the mediator key to decrypt the token and uses it to log on to the new Teamcenter session as the original user.

Configure an SOA URL for background processing

Background processing (asynchronous functionality) requires a service oriented architecture (SOA) URL.
1. Open the Organization application in Teamcenter.

2. Select the top-level Sites node from the Organization List tree. The Sites pane appears.

3. Type the SOA URL in the SOA URL box. This URL is used for SOA calls to this site.

Configure notifications for background processing

Background processing (asynchronous functionality) uses Subscription Manager to notify users of completed and failed requests. Configure notification behavior by importing and configuring the ASYNC_subscribe_to_background_tasks preference and defining event types.

1. Use the preferences_manager utility to import and set the preference:

   ```
   preferences_manager -u=infodba -p=infodba -g=dba -mode=import
   -preference=ASYNC_subscribe_to_background_tasks -scope=SITE
   -values=NONE|BOTH|FAIL|SUCCEED -action=OVERRIDE
   ```

   The -values value must be one of the following:

   - NONE: No notification e-mail is sent.
   - Both: Notification e-mail is sent upon success and upon failure.
   - FAIL: Notification e-mail is sent upon failure.
   - SUCCEED: Notification e-mail is sent upon success.

2. Install new event type mappings by creating a file with the following content:

   ```
   DispatcherRequest,DispatcherRequest,__Async_Request_Succeeded,true,true
   DispatcherRequest,DispatcherRequest,__Async_Request_Failed,true,true
   ```

3. Execute the install_event_types utility.

   ```
   install_event_types -u=infodba -p=infodba -f=file -overwrite
   ```

4. Ensure Subscription Manager preferences are configured to correctly send notifications.

   a. Choose Edit→Options to open the Options dialog box.

   b. Click the Index tab at the bottom left of the dialog box and type TC_subscription in the Search on preference name box. Confirm that the value is set to ON.

   c. Type Mail_server_name in the Search on preference name box. Set the value to your mail server.

   d. Type TC_notification_msg_ext in the Search on preference name box. Confirm that the value is set to txt to ensure your e-mail system does not block the attachment in the notification e-mail.
Editing templates

Determining which editing options to use

Perform edits on existing workflow process templates by selecting the template to be edited and clicking the Edit button.

Consider the following questions before editing a workflow template.

<table>
<thead>
<tr>
<th>Editing task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit offline or online?</td>
<td>Offline editing prevents users from accessing the workflow template while you edit. Use this option when you do not want the old version of the workflow template available for use until your edits are complete.</td>
</tr>
<tr>
<td></td>
<td>Online editing allows users to initiate workflows based on the old version of the workflow template while you edit a copy of the same workflow template. When you switch the edited version to the Available stage, the older copy is overwritten; only the edited copy remains available from the interface.</td>
</tr>
<tr>
<td></td>
<td>For more information about the behavior of offline and online editing, see Editing offline versus online.</td>
</tr>
<tr>
<td>Apply edits to running workflow processes?</td>
<td>After editing a workflow template, you can choose to apply the edits to all active workflow processes based on that template.</td>
</tr>
<tr>
<td></td>
<td>When you select the Set Stage to Available check box to change the template’s stage to Available, the Apply Template Changes dialog box asks whether to apply the edits to all active workflow processes based on the template.</td>
</tr>
<tr>
<td></td>
<td>Select the Apply template changes to all active workflow processes check box to update each active workflow process based on the workflow template as follows:</td>
</tr>
<tr>
<td></td>
<td>• If the edits in the workflow template occur later in the workflow than the active workflow process has reached, the edits are applied to the workflow.</td>
</tr>
<tr>
<td></td>
<td>• If the edits in the workflow template occur earlier, and the active workflow has already passed the place where the edits were made, the edits do not take effect, unless the task/path is re-executed using backward branching/loops or when a task is demoted.</td>
</tr>
<tr>
<td></td>
<td>• If the edits in the workflow template impact an active task, the edits are applied after the task completes and only take effect if the task is re-executed.</td>
</tr>
</tbody>
</table>
Chapter 1  
*Creating workflow process templates*

<table>
<thead>
<tr>
<th>Editing task</th>
<th>Description</th>
</tr>
</thead>
</table>
| Which workflow components can be edited? | You can edit any aspect of the workflow process template, including:  
  - Changing the template name  
  - Adding and removing tasks  
  - Adding, deleting, redrawing, and resetting flow paths  
  - Adding, deleting, and resetting handlers, attributes, task attributes, and attachments  
  For more information about the editing workflow template procedure, see *Mechanics of the Workflow Designer user interface.* |

**Editing offline versus online**

Deciding whether to edit a workflow template online or offline is determined by whether you want to grant users access to the existing version of the workflow template while you edit it.

- **Online editing** allows users to initiate workflows based on the old version of the workflow template while you edit a copy of the same workflow template.

  Select **No** in the **Offline?** dialog box to edit online. The system makes a copy of the workflow template and sets it to the **Under Construction** stage; this is the version you edit. Both versions of the workflow template display in the **Process Template** list in the **New Process** dialog box. The **Under Construction** symbol displays next to the version being edited.

  Users can continue to use the unedited version of the workflow template. When you switch the edited version to the **Available** stage, the older copy is overwritten; only the edited copy remains available from the interface.

- **Offline editing** prevents users from accessing the workflow template while you edit it.

  Select **Yes** in the **Offline?** dialog box to edit offline. With this option, there is only one instance of the template. The system sets the workflow template to the **Under Construction** stage. The template is not available to users initiating workflow processes against objects; it does not display in the **Process Template** list in the **New Process** dialog box.
Only users with privileges to edit workflow templates can see the workflow template in the **Process Template** list, which is marked with the **Under Construction** symbol. When you switch the workflow template to the **Available** stage, the edited workflow template becomes available to users.

**Edit workflow templates**

1. Select the desired workflow template from the **Process Template** box.

2. Select **Edit** mode.
   
   A dialog box asks whether you want to take the selected process template offline. Select **Yes** to take the workflow template offline, preventing users from initiating workflow processes based on this template while you edit. The workflow template is not available to users from the **Process Template** list while you keep the template offline.

3. (Optional) Rename the template by selecting the existing template name in the **Name** box under the **Set Stage to Available** check box and typing a new name over the selection. Alternatively, backspace from the end of the name to delete the characters. After you type a new name, click one of the tasks in the task hierarchy tree to set the new name. You cannot change the name using the **Process Template** box.

   **Warning**

   You cannot select the existing name and use the Delete key to delete the entire name at once. The system interprets use of the Delete key as a command to delete an object from the database.

4. (Optional) Add, place, and remove tasks.
   
   For more information about tasks, see **Task template definitions**.

5. (Optional) Add, remove, and modify task attributes by clicking the **Task Attributes** button.
   
   For more information, see **Edit task attributes**.

6. (Optional) Edit task handlers by clicking the **Task Handlers** button.
   
   For more information, see **Edit task handlers**.

7. (Optional) Edit perform signoff teams by clicking the **Task Signoff** button.
   
   For more information, see **Task signoffs**.

8. After you finish editing the workflow template, select the **Set Stage to Available** check box.
   
   The **Stage Change** dialog box appears, stating that changing the template stage to available makes the template visible to all users and asking if you want to continue. Click **Yes** to save your changes to the database, make the template visible to all users, and return to **Browse** mode.
Click **No** to remain in **Edit** mode.

**Configure ability to apply template edits to active processes**

Before you can apply workflow template edits to active workflow processes, you must configure the **EPM_enable_apply_template_changes** site preference. By default, this preference is set to **NONE**, which suppresses this functionality.

1. Choose **Edit→Options** to open the **Options** dialog box.

2. Click the **Index** tab at the bottom left of the dialog box and type **EPM_enable_apply_template_changes** in the **Search on preference name** box.

3. Select the **EPM_enable_apply_template_changes** preference from the **Preferences List** and set the values to one of the following:

   **OPTIONAL**
   
   Allows you to choose on a case-by-case basis whether to apply workflow template edits to active workflow processes based on the workflow template.

   After editing a workflow template and selecting the **Set Stage to Available** check box to change its stage to **Available**, the **Apply Template Changes** dialog box allows you to apply your edits to all active workflow processes based on the edited template.

   Select the **Apply template changes to all active workflow processes** check box to apply your edits as described in **Applying template edits to active workflow processes**.

   **AUTOMATIC**
   
   Automatically applies edits to a workflow template to all active workflow processes based on the edited template.

   After editing a workflow template and selecting the **Set Stage to Available** check box to change its stage to **Available**, the edits are automatically applied to all active workflow processes based on the edited template.

   By default, this setting applies the edits in the background. However, this functionality requires a four-tier architecture environment. (Users running in a two-tier environment can successfully submit requests for asynchronous processing if there is a four-tier Teamcenter environment available to accept the request.) Additionally, Dispatcher must be enabled and configured for asynchronous processing.

   **Note**
   
   If background processing is not configured and supported at your site, active workflow processes are updated in real time. When updating in real time, the Teamcenter interface pauses until the updates complete.

   For more information about this preference, see the *Preferences and Environment Variables Reference*.

Updating the workflow processes in the background is the recommended method, and, by default, the **Update processes in background** check box is selected.
Note

If you apply the updates in real time, the Teamcenter interface is unavailable until the updates complete. This method is suitable for testing. It is not recommended when updating more than 30–50 workflow processes.

The update duration depends on the type of edits made to the workflow processes. For example, it takes longer to remove tasks than add tasks. Edits within tasks (handlers, attributes, etc.) require minimal processing time.

Applying template edits to active workflow processes

You can use Workflow Designer to apply workflow template edits to all active workflow processes based on the previous (unedited) version of the workflow template.

Applying workflow template edits to all active workflow process is a powerful way to edit many active processes simultaneously. Because this is a far-reaching procedure, it is important to understand exactly how the edits are applied:

• If the edits in the workflow template occur later in the workflow than the active workflow process has reached, the edits are applied to the workflow.

• If the edits in the workflow template occur earlier, and the active workflow has already passed the place where the edits were made, the edits do not take effect, unless the task/path is re-executed using backward branching/loops or when a task is demoted.

• If the edits in the workflow template impact an active task, the edits are applied after the task completes and only take effect if the task is re-executed.

• If the edits deletes the currently active task, the next task is started.

Note

This can result in users logging on and finding that tasks they were working on were removed from their worklist.

Additionally, active workflow processes can be updated in a similar manner when importing updated versions of a workflow template, either through the Workflow Designer application or using the plmxml_import utility.

For more information about importing workflow templates using Workflow Designer, see Import workflow templates.

For more information about importing workflow templates using the plmxml_import utility, see the Utilities Reference.

Before you can fully use this behavior, several procedures are required to enable and configure two types of functionality:

• Applying template edits to active workflow processes

• Allowing the active workflow processes to be updated in the background

For more information about the required configuration procedures, see Configure ability to apply template edits to active processes and Configuring background processing of processes and tasks.
Apply template edits to all active workflow processes

You can apply edits to active workflow processes after you have completed editing a workflow template and are ready to make the workflow template available to users.

1. Select the **Set stage to available** check box to change the workflow template’s stage to **Available**.

   The **Apply Template Changes** dialog box appears asking whether to apply your edits to all active workflow processes based on the template.

   **Note**
   
   You can also change a workflow template’s stage from **Under Construction** to **Available** when closing Workflow Designer. The **Set To Available Stage Template** dialog box displays whenever under construction workflow templates exist when you close Workflow Designer.
   
   Using this dialog box to change a template’s stage does *not* allow you to apply template edits to active workflow processes.

2. Select the **Apply template changes to all active workflow processes** check box.

   Your edits are applied to each active workflow process based on that workflow template. Edits are applied as listed in **Applying template edits to active workflow processes**.

3. (Optional) Select the **Update processes in background** check box.

   Your edits are applied in the background. The updates run asynchronously and you are notified by Teamcenter mail when the updates complete.

   Typically, you only want to update workflow processes in real time when your changes impact 10–20 active workflow processes, as in testing scenarios.

   **Caution**
   
   Asynchronous processing must be configured.

   For more information about the required configuration procedures, see **Configuring background processing of processes and tasks**.

   You can also edit an active workflow process in Workflow Viewer, in which you edit the particular active workflow process, not the workflow template on which it is based. This method allows you to edit only one active workflow process at a time.

   For more information about this method, see **Workflow Viewer Guide**.

Creating baseline workflow process templates

The baseline feature allows you to create a baseline, or a snapshot of a work-in-process item revision and its component objects without incrementing the revision of the item. This enables you to capture a product design at a particular stage without having to stop work or generate an undesired revision of the item.

Before you can implement baseline functionality, you must create one or more custom workflow process templates to support **baseline release procedures**. These workflow process templates must define a zero-step release procedure, which
allows the baseline to become a released object that cannot be modified. This type of workflow process template is referred to as a quick release template.

After the quick release template is created, you need to set its name in the Baseline_release_procedures preference. Once this preference is set, the name of the quick release workflow process template displays in the Release Procedure list and can be selected by a user.

Create a quick-release workflow process template


   The New Root Template dialog box appears.

2. In the New Root Template dialog box, select the Process option for Template Type, type a name in the New Root Template Name box, and select Empty Template from the Based on Root Template list.

3. Click OK.

4. On the toolbar, click the Add Status Task Template button.

5. Double-click between the Start and Finish tasks in the process flow pane to insert the new Add Status task.

6. Create a path between the Start node and the Add Status task by placing the cursor in the body of the Start node and dragging it to the body of the Add Status task.

7. Create a path between the Add Status task and the Finish node by placing the cursor in the body of the Add Status task and dragging it to the body of the Finish node.

8. Select the Set Stage to Available check box to make the template available.

By adding the Add Status task, your new quick-release workflow process template contains the required create-status and add-status handlers.

The template displays in the Process Template list and in the Based On Root Template list in the New Root Template dialog box.

Creating subprocesses

What are subprocesses?

Subprocesses are workflow processes associated with a parent workflow process. If there is an association between the parent process and subprocess, but not a dependency, the parent process may complete before the subprocess completes.

If the parent process is dependent on the subprocess, the parent process cannot complete until the subprocess completes. For example, if the EPM-create-sub-process action handler is used to create subprocesses for
multiple targets from a parent process, the parent processes are dependent on the subprocesses.

If a subprocess is created from an in-process task, the task cannot complete until the subprocess completes. End users can create subprocesses in this manner. Subprocesses are created in two locations:

<table>
<thead>
<tr>
<th>Parent workflow templates</th>
<th>Administrators can configure workflow templates to create subprocesses. For example, a parent workflow template can be configured to automatically launch subprocesses for each target of the parent workflow. For more information about creating subprocesses from a parent workflow template, see Creating subprocesses from a workflow template.</th>
</tr>
</thead>
<tbody>
<tr>
<td>My Worklist</td>
<td>End users can create ad hoc workflow subprocesses while performing tasks from their worklist or from Workflow Viewer. For more information about creating ad hoc subprocesses, see Creating ad hoc subprocesses.</td>
</tr>
</tbody>
</table>

Creating subprocesses from a workflow template

Sometimes you want a workflow process to generate additional workflows as it proceeds. For example, you may want a workflow to generate additional workflows (subprocesses) for each target of the parent process. Use the EPM-create-sub-process action handler to create subprocesses. You can add the handler multiple times to a single task action, allowing you to use different workflow process templates per target object type. Use the handler to:

- Set dependencies between the parent process and its subprocesses.
- Define targets and attachments for the subprocesses.
- Transfer attachments from the parent process to a subprocess.
- Create subprocesses for multiple targets.
- Create subprocesses for assemblies.
- Create subprocesses for related objects.

The handler accepts numerous arguments, allowing you to create a wide variety of instances for generating subprocesses. For example:

- The following argument settings create a subprocess based on the Clinical Trials Phase I template, which inherits all the targets and reference attachments from the parent process. Because the workflow process name is not defined, a workflow process name for the child process is automatically generated in the format parentprocess:count.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>Clinical Trials Phase I</td>
</tr>
<tr>
<td>-from_attach</td>
<td>ALL</td>
</tr>
</tbody>
</table>
### Creating workflow process templates

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-to_attach</td>
<td>ALL</td>
</tr>
</tbody>
</table>

- The following argument settings launch a subprocess based on the **Clinical Trials Phase I** workflow process template. All item revisions from the parent process are excluded as targets for the new workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>Clinical Trials Phase I</td>
</tr>
<tr>
<td>-from_attach</td>
<td>ALL</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-exclude_types</td>
<td>ItemRevision</td>
</tr>
</tbody>
</table>

- The following argument settings launch multiple subprocesses based on the **Clinical Trials Phase I** workflow process template. Each item revision that was a target or reference attachment of the parent process launches a new subprocess with that item revision as the target.

For example, if the parent process contained three item revisions as targets, three different subprocesses are launched.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>Clinical Trials Phase I</td>
</tr>
<tr>
<td>-from_attach</td>
<td>ALL</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-include_types</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-multiple_processes</td>
<td></td>
</tr>
</tbody>
</table>

For more information about using this handler to create subprocesses, see [EPM-create-sub-process](#).

### Creating subprocesses for multiple targets

There are various configurations of the **EPM-create-sub-process** action handler you can use to create subprocesses for multiple targets from a parent process.

The most straightforward method is to use the **-multiple_processes** argument to create individual subprocesses for each target in the parent process. The newly created subprocesses can either be a clone of the parent process or a different workflow process.

You can refine this method by using the **-include_types** argument along with the **-multiple_processes** argument to create individual subprocesses for each target of a specific type in the parent process. Or you can use the **-exclude_types** argument along with the **-multiple_processes** argument to create individual subprocesses for each target except the specified types in the parent process and so on.

All these methods are based on the concept of the parent process always creating one or more subprocesses.
Depending on your business process needs, a more elegant method is to create a workflow process branched with a **Condition** task that is configured to query for multiple targets. The technique of querying for multiple targets means a subprocess is only created when there are multiple targets. When there is a single target, the other branch of the parent process is followed. This is an efficient design if subprocesses are only needed when multiple targets are involved.

Consider the following workflow template, in which a generic task template is named **Multiple Targets** and configured to create subprocesses for each target.

In this example, Pharmaceuticals, Inc., uses such a workflow for its drug trial reviews. The typical trial contains multiple products, but occasionally a trial contains only one product.

If this workflow process is initiated on an item revision containing three targets, the **Condition** task query returns **True** and follows the **True** path containing the **Multiple Targets** task, which creates three subprocesses. One subprocess for each target in the parent process. Each subprocess is a clone of the parent process.

Because each of the subprocesses always only contains a single target, as each subprocess is initiated the **Condition** task query returns **False** and follows the **False** path containing the **Launch Trial** and **Review Results** tasks.

In trials that review only a single product, the parent process follows the **False** path. No unnecessary subprocess is created.

The following procedure illustrates how to configure the workflow in this example:

**Note**

Before you begin, confirm that the **EPM_multiple_processes_targets** site preference is set to **ON** by choosing **Edit→Options** to launch the **Options** dialog box and locating the preference using the **Index** tab.

If the preference is not created at your site, create the preference and set it to **ON**.

For more information about creating preferences, see the **Rich Client Interface Guide**.

1. in Workflow Designer, choose **File→New Root Template** to create a new workflow process template.
2. Type a name for the new workflow process in the **New Root Template Name** box and click **OK**.
   The workflow process template appears in the process flow pane.

3. On the toolbar, ensure you are in **Edit 🔍** mode.
   This allows you to edit the workflow process template.

4. Insert a **Condition** task into the workflow process by clicking the **Condition Task** button on the toolbar, and then double-clicking in the process flow pane to the right of the **Start** node.
   The new **Condition** task is inserted at the cursor point.

5. Rename the **Condition** task by selecting the task in the task hierarchy tree, and then typing **Has Multiple Targets?** in the **Name** box in the template manager pane, and pressing the Enter key.

6. Draw a flow path from the **Start** task to the **Has Multiple Targets?** task by placing the cursor in the body of the **Start** task and dragging it to the body of the **Has Multiple Targets?** task.

7. Insert a **Do** task above and to the right of the **Condition** task.

8. Rename the **Do** task to **Launch Trial**.

9. Insert a **Review** task to the right of the **Launch Trial** task.

10. Rename the **Review** task to **Review Results**.

11. Draw a flow path from the **Launch Trial** task to the **Review Results** task by placing the cursor in the body of the **Launch Trial** task and dragging it to the body of the **Review Results** task.

12. Draw a flow path from the **Has Multiple Targets?** task to the **Launch Trial** task.
   By default, the path is a **True** path.

13. To change the flow path to a **False** path, right-click the line you have just drawn and choose **Select Path To False Path**.
   The flow path changes to a **False** path.

14. Insert a generic task below and to the right of the **Has Multiple Targets?** task.

15. Rename the task to **Multiple Targets**.

16. Draw a flow path from the **Has Multiple Targets?** task to the **Multiple Targets** task.
   By default, the path is a **True** path.

17. Create a query for the **Has Multiple Targets?** task to determine whether the workflow process contains multiple targets by completing the following steps:
   a. In Teamcenter, switch to the Query Builder application.
b. In Query Builder, create a new query called **WF - Has Multiple Targets** by completing the query boxes as shown and clicking **Create**.

![Query Builder screenshot](image)

For more information about using this application to create queries, see the **Query Builder Guide**.

c. Return to Workflow Designer.

18. Associate the **WF - Has Multiple Targets** query with the **Has Multiple Targets?** task.

a. Select the **Has Multiple Targets?** task and click **Task Attributes** in the template manager pane.

b. In the **Task Attributes** dialog box, click the **Condition Query** box. (The box currently indicates it is empty because no queries are associated with the **Condition** task.)

The **Condition Query** dialog box appears.

c. In the **Condition Query** dialog box, scroll down the **Build/Select Query** list to the **WF - Has Multiple Targets** query and double-click the query.

The query name appears in the **New Query** box at the top of the dialog box.

d. Select **Task** as the **Query Against** option.
Creating workflow process templates

e. Click **Assign** at the top of the dialog box to assign this query to the **Has Multiple Targets?** task.

f. Click **Close** to exit the dialog box.

   The **Task Attributes** dialog box reappears. **WF - Has Multiple Targets** displays in the **Condition Query** box.

g. Close the **Task Attributes** dialog box.

   The **Has Multiple Targets?** task is now configured to query whether the workflow process contains multiple targets. When the workflow process contains multiple targets the **True** path is followed; when the workflow process contains a single target, the **False** path is followed.

19. Configure the **Has Multiple Targets?** task to retrieve the number of targets from the **Multiple Targets** task by completing the following steps:

   a. In the process flow pane, select the **Has Multiple Targets?** task and click **Task Handlers** in the template manager pane.

   b. In the task action in the left-side of the dialog box, select the **Start** action.

   c. In the right-side of the dialog box, select **Action Handler** for the handler type.

   d. In the **Action Handler** list, select **EPM-set-task-result-to-property**.

   e. Type **-property** in the **Argument** box and **num_targets** in the **Value(s)** box.

   f. Click **Add** in the right side of the dialog box to add another argument/value line.

   Type **-source** in the **Argument** box and **task** in the **Value(s)** box.

   g. Click **Create** at the bottom of the dialog box to add the handler to the **Start** action of the **Has Multiple Targets?** task.

20. When you created the **WF - Has Multiple Targets** query on the **Has Multiple Targets?** task, the **set-condition** handler was automatically placed on the task’s **Start** action.

   Confirm the handler contains the following settings:

   a. The **$Query** in the **Argument** box and **WF - Has Multiple Targets** in the **Value(s)** box.

   b. The **-query_type** in the **Argument** box and **Task** in the **Value(s)** box.

       **Note**

       The order of the two handlers on the **Start** action is important. **EPM-set-task-result-to-property** must be before **set-condition**.

21. Remove the **EPM-attach-item-revision-targets** handler from the **Start** task by completing the following steps:
a. In the process flow pane, select the Start task and click Task Handlers in the template manager pane.

b. Select the EPM-attach-item-revision-targets handler and click the Delete button.

c. Close the Task Handlers dialog box.

22. Configure the Launch Trial task to attach the dataset and BOM view revision by completing the following steps:

a. In the process flow pane, select the Launch Trial task and click Task Handlers in the template manager pane.

b. In the task action tree in the left side of the dialog box, select the Start action.

c. In the right side of the dialog box, select Action Handler for the handler type.

d. In the Action Handler list, select EPM-attach-related-objects.

e. Type -relation in the Argument box and IMAN_specification in the Value(s) box.

f. Click Add in the right side of the dialog box to add another argument/value line.

g. Type -att_type in the Argument box and TARGET in the Value(s) box.

h. Click Create in the bottom of the dialog box to add the handler.

i. Repeat steps d–h, but use PSBOMViewRevision as the value for the -relation argument in step e.

23. Configure the Multiple Targets task to generate subprocesses by completing the following steps:

a. In the process flow pane, select the Multiple Targets task and click Task Handlers in the template manager pane.

b. In the task action tree in the left side of the dialog box, select the Complete action.

c. In the right side of the dialog box, select Action Handler for the handler type.

d. In the Action Handler list, select EPM-create-sub-process.

e. Type -template in the Argument box and the name of the workflow process template (defined in Step 2) in the Value(s) box.

f. Click Add in the right side of the dialog box to add another argument/value line.

g. Type -from_attach in the Argument box and Target in the Value(s) box.
Creating workflow process templates

h. Click Add in the right side of the dialog box to add another argument/value line.

i. Type -to_attach in the Argument box and Target in the Value(s) box.

j. Click Add in the right side of the dialog box to add another argument/value line.

k. Type -process_name in the Argument box and SubProcess in the Value(s) box.

l. Click Add in the right side of the dialog box to add another argument/value line.

m. Type -multiple_processes in the Argument box. Do not type a value in the Value(s) box.

n. Click Create in the bottom of the dialog box to add the handler to the Complete action of the Multiple Targets task.

o. Close the Handlers dialog box.

24. Reconcile the True and False paths by inserting an Or task and linking it to the Review Results and Multiple Targets tasks.

a. Click the Or task button > on the toolbar, and then double-click in the process flow pane to the right of the Review Results and Multiple Targets tasks. The new Or task is inserted at the cursor point.

b. Draw a flow path from the Review Results task to the Or task.

c. Draw a flow path from the Multiple Targets task to the Or task.

25. Draw a flow path from the Or task to the Finish node to complete the workflow.

Creating subprocesses for assemblies

In workflow processes that contain assemblies, there are various arguments you can use with the EPM-create-sub-process action handler to create subprocesses for components of the assemblies.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>-process_assembly</td>
<td>Searches for assemblies in the target, reference, or all (as specified by the -from_attach argument) and creates subprocesses for each component.</td>
</tr>
<tr>
<td>-depth</td>
<td>Specifies the depth to which the assembly is traversed.</td>
</tr>
<tr>
<td>-rev_rule</td>
<td>Specifies the revision rule applied to the assembly.</td>
</tr>
</tbody>
</table>
## Creating workflow process templates

### Argument | Behavior
--- | ---
-include_related_types | Creates subprocesses only for assembly components of the types specified in this argument.
-exclude_related_types | Does not creates subprocesses for assembly components of the types specified in this argument.

**Note**

The -include_related_types and -exclude_related_types arguments can be used in conjunction with each other. If used in conjunction, the -include_related_types argument takes precedence; first the objects are processed against -include_related_types and then processed against -exclude_related_types.

### Creating subprocesses for related objects

There are various arguments you can use with the EPM-create-sub-process action handler to create subprocesses for related objects of target and reference data.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation</td>
<td>Creates subprocesses for each object attached by the specified relation to the target or reference object. (Specify a particular target, or reference object, or all, using the -from_attach argument.)</td>
</tr>
<tr>
<td>include_related_types</td>
<td>Creates subprocesses only for related objects of the type(s) specified in this argument.</td>
</tr>
<tr>
<td>exclude_related_types</td>
<td>Does not creates subprocesses for related objects of the type(s) specified in this argument.</td>
</tr>
</tbody>
</table>

**Note**

The -include_related_types and -exclude_related_types arguments can be used in conjunction with each other. If used in conjunction, the -include_related_types argument takes precedence; first the objects are processed against -include_related_types, and then -exclude_related_types.

### Creating ad hoc subprocesses

End users can create ad hoc workflow subprocesses while performing tasks from their worklist or from Workflow Viewer.

For example, users might want to create a workflow subprocess after receiving a task in their worklist dependent upon the completion of one or more tasks not
Creating workflow process templates

tracked by the existing workflow. They create a workflow subprocess to track the additional tasks.

For more information about how users create ad hoc subprocesses, see the Workflow Viewer Guide.

Core templates

The following table lists the templates and their associated types included with the rich client.

<table>
<thead>
<tr>
<th>Template name</th>
<th>Task template definition type</th>
<th>Task type value specified in task template</th>
<th>Executing task's real type</th>
<th>Executing task's task type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>EPMTaskDefinition</td>
<td>EPMTask</td>
<td>EPMTask</td>
<td>EPMTask</td>
</tr>
<tr>
<td>Review Process Task</td>
<td>EPMTaskDefinition</td>
<td>EPMTask</td>
<td>EPMTask</td>
<td>EPMTask</td>
</tr>
<tr>
<td>Review Task</td>
<td>EPMTaskDefinition</td>
<td>EPMTask</td>
<td>EPMTask</td>
<td>EPMReviewTask</td>
</tr>
<tr>
<td>Do Task</td>
<td>EPMDoTaskDefinition</td>
<td>EPMDoTask</td>
<td>EPMTask</td>
<td>EPMDoTask</td>
</tr>
<tr>
<td>Or Task</td>
<td>EPMTaskDefinition</td>
<td>EPMTask</td>
<td>EPMTask</td>
<td>EPMTask</td>
</tr>
<tr>
<td>Add Status Task</td>
<td>EPMTaskDefinition</td>
<td>EPMTask</td>
<td>EPMTask</td>
<td>EPMTask</td>
</tr>
<tr>
<td>Change Management Item</td>
<td>EPMTaskDefinition</td>
<td>EPMTask</td>
<td>EPMTask</td>
<td>EPMTask</td>
</tr>
</tbody>
</table>
Chapter

2 Viewing workflow process templates

Filter template display based on user group and target object .................. 2-1
View a workflow process template ......................................................... 2-2
View a subtask ...................................................................................... 2-2
View a parent task .................................................................................. 2-2
View the root task ................................................................................... 2-3
Viewing a subprocess .............................................................................. 2-3
View task attributes ................................................................................ 2-3
  Set Duration ......................................................................................... 2-4
  Set Recipients list ............................................................................... 2-5
View task handlers .................................................................................. 2-5
Chapter

2 Viewing workflow process templates

Filter template display based on user group and target object

You can define which workflow process templates display in the Process Template list based on the group of the initiating user and the object that is selected as the target object.

If you associate templates with object types and they have subtypes, Teamcenter does not automatically associate the templates with the subtypes. You must associate the templates with the subtypes as well.

If a user subgroup has no associated templates for an object type, the subgroup inherits its templates from its first parent group up the hierarchy that has associated templates for that object type. If you explicitly associate templates with a subgroup, the subgroup does not inherit any from its parent group.

1. Choose Edit→Template Filter.
   The Process Template Filter Dialog dialog box appears.

2. From the Group Name list, select the group whose workflow process template list you want to filter.

3. From the Object Type list, select the target object.
   The Object Type list displays all the target object types defined in the database.

4. From the Defined Process Template list, select the workflow process template you want to display for the selected group and object and click the button.
   The selected workflow process template moves to the Assigned Process Template list.

5. Repeat the previous step until you have selected all the workflow process templates you want to display for the selected group and object type.

6. Click one of the following:
   - **OK** to save the Assigned Process Template list and exit the dialog box.
   - **Apply** to save the Assigned Process Template list. The dialog box remains open allowing you to create additional filters.
   - **Clear** to refresh the Assigned Process Template list based on the previous saved result.
View a workflow process template

The task hierarchy tree presents a root-level workflow process, along with its tasks and subtasks, in a hierarchical listing. Task precedence is based on the order in which tasks are created.

The process flow pane provides graphical views of the different levels of a workflow process. You can view all the tasks in an entire workflow process, or the subtasks in a task, or the subtasks of subtasks, and so on.

View a subtask

You can move down a level in a workflow process template from either the task hierarchy tree or the process flow pane while in either Edit or Browse mode.

- In the task hierarchy tree, select a task whose subtasks you want to view. Click Go→Down a Task Level.
  
The subtasks display in the process flow pane.
  
  For example, selecting a container task displays the task’s subtasks in the process flow pane. Selecting the root task displays the first task listed in the task hierarchy tree in the process flow pane.

- In the process flow pane, double-click the task node whose subtasks you want to view.
  
The process flow pane displays the subtasks of the selected task.

  Note

  If you select a task node with no subtasks, the process flow pane displays an empty template, with only the Start and Finish nodes showing.

- In the task hierarchy tree, select the task node whose subtasks you want to view. Click Down a Task Level.
  
The process flow pane displays the subtasks of the selected task node.

View a parent task

You can move up a level in a workflow process template from either the task hierarchy tree or the process flow pane, while in either Edit or Browse mode.

You can view the parent task in one of these ways:

- In the process flow pane, select the task node whose parent task you want to view. Click Up a Task Level.
  
The process flow pane displays the parent task of the selected task.

- Cancel to close the dialog box without applying the changes.
Note
If the root task’s subtasks are showing in the process flow pane, you are already at the top level and the system ignores the **Up a Task Level** action.

- In the task hierarchy tree, select the task node whose parent task you want to view. Click **Up a Task Level**.
  The process flow pane displays the parent task of the selected task.

**View the root task**

You can move to the top level from anywhere in a workflow process template from either the task hierarchy tree or the process flow pane, while in either edit or browse mode.

1. In the process flow pane, select any task node. Choose **Go→Top Level**.
   The process flow pane displays the top level of the workflow process.

   **Note**
   If the root task’s subtasks are showing in the process flow pane, you are already at the top level.

2. In the task hierarchy tree, select any task node. Click **Go→Top Level**.
   The process flow pane displays the top level of the workflow process.

**Viewing a subprocess**

Subprocesses are started from the parent workflow process under each task of the parent workflow process. You can cut and paste a workflow process to create a new subprocess.

When you expand a task in **My Worklist**, a subprocess folder displays with **Target** and **Reference** folders. All the subprocesses of the parent workflow process display under this folder. If the workflow process does not have any workflow subprocesses, the system does not display any folders.

**View task attributes**

When you view task attributes in browse mode, you have read access only.

1. Click **Browse Mode**.

2. Select the task whose attributes you want to view.
   - Click **Task Properties** in the toolbar.
     The **Task Properties** dialog box appears. The **Name** box displays the name of the selected workflow process or task template. The **Description** box lists the task description.
   - Click **Attributes Panel**.
Chapter 2  

Viewing workflow process templates

The Attributes Pane dialog box appears.

- The Named ACL box lists the Named ACL, assigned to this task. For more information about Named ACLs, see the Security Administration Guide.

- The Task Type box lists the type of task template assigned to the selected task.

- The Icons box displays the symbol that has been assigned to the selected task. You can also add custom symbols to this list.

- If a Condition task is selected, the Condition Query box displays the name of the assigned query. If a query has not yet been defined, only the Condition Query button displays.

If a Condition task is selected, the Condition Result box displays the result of the query; either true or false. If a query has not yet been defined, the result is listed as unset.

If a task immediately succeeding a Condition task is selected, the Condition Path box appears. Click the Display condition path values box to display the Condition Path dialog box, which lists the value of the path between the Condition task and the selected task: either true or false.

3. Select Show Task in Process Stage List to enable template staging functionality. The Set Stage to Available check box is displayed for new templates.

4. Click Close.

Set Duration

The Duration box displays the length of time allowed for the completion of the project. You can define the duration length in the template of the selected task. You can also define the duration length in the Attributes dialog box when the selected task is in a Pending state.

1. Click Set to the right of the Duration box.

The Set Duration dialog box appears.

2. Type an integer value for any or all of the following fields to indicate the length of time that can pass before the selected tasks need to reach completion:

- Years
- Weeks
- Days
- Hours
- Minutes

3. Click one of the following:

- OK to save the changes to the database and close the dialog box.
- Clear to clear all boxes.
- Cancel to close the dialog box without applying the changes.
Set Recipients list

The Recipients list displays the names of users selected to receive program mail when the selected task becomes overdue. You can set the Recipients list from this dialog box.

1. Click Set to the right of the Recipient box.
   The Select Recipients dialog box is displayed.

2. Type the user, group, or address list search criteria for users you want to select.

3. Click either User, Group, or Address List, based on the search criteria you entered. The search results display in the boxes below. To display all users in the selected grouping, type an asterisk and click the appropriate button. All users in the selected grouping are displayed in the box below.

4. Select the users you want to define as recipients from the search results. You can choose multiple users by pressing Ctrl and selecting the desired names.

5. Click User.
   The selected users display in the box in the right side of the dialog box. These are the selected recipients.

6. Click one of the following:
   - OK to save the changes to the database and close the dialog box.
   - Cancel to close the dialog box without applying the changes.

7. (Optional) Select the Show Task in Process Stage List to display the task in the Process Stage List property for the target object.

8. Click Close.

View task handlers

Viewing task handlers in browse mode allows read access only. For information about editing task handlers, see What are task handlers?.

1. Click Browse Mode.

2. Select the task whose handlers you want to view. To view handler information for the root task of the workflow process (the initial Start task), select the workflow process.

3. Click the Task Handlers panel.
   The Task Handlers dialog box appears. In the left pane, the Handler lists the handlers assigned to the selected task.

4. Click Expand All Folders or Collapse All Folders to view the contents of the Handler.
Based on the type of handler selected, either the **Rule Handler** or **Action Handler** appear, listing the name of the rule or action handler assigned to the selected task.

If the selected task involves selecting signoff teams or performing signoffs, the **Quorum** box lists the number or percentage required for a quorum.

The **Argument** list shows the arguments assigned to the selected task.

The **Task Action** list shows the actions assigned to the selected task.

5. Click **Close**.
Chapter

3 Adding tasks to workflow process templates

Workflow task actions and states ........................................... 3-1
Task template definitions .................................................... 3-4
Using Do tasks ................................................................. 3-5
  Insert and configure a Do task ........................................... 3-6
Using Acknowledge tasks ..................................................... 3-7
  Insert and configure an Acknowledge task ............................ 3-7
Using Review tasks ............................................................ 3-9
  Insert and configure a Review task ..................................... 3-10
Using Route tasks .............................................................. 3-11
  Insert and configure a Route task ....................................... 3-12
Using generic tasks ............................................................ 3-13
  Insert and configure a Custom task ..................................... 3-13
Using Impact Analysis tasks .................................................. 3-14
  Insert and configure an Impact Analysis task ....................... 3-14
Using Prepare ECO tasks ...................................................... 3-15
  Insert and configure a Prepare ECO task ............................. 3-16
Using Checklist tasks .......................................................... 3-17
  Insert and configure a Checklist task .................................. 3-18
Using Condition tasks .......................................................... 3-19
  Creating manual Condition tasks ...................................... 3-20
  Creating automatic Condition tasks .................................... 3-21
  Configuring Condition tasks ............................................. 3-22
  Add a Condition task to a process template ......................... 3-22
  Set Condition task paths .................................................. 3-23
Using Validate tasks ........................................................... 3-24
  Find error codes ................................................................ 3-25
  Add error codes ................................................................ 3-26
  Insert and configure a Validate task .................................... 3-28
Validate task example: Close gaps in your workflow .................. 3-28
Validate task example: Improve user response time .................... 3-32
Validate task example: Track errors from custom handlers ............... 3-34
Validate task behavior ......................................................... 3-38

Using Or tasks ................................................................. 3-38
Insert and configure an Or task ............................................. 3-39

Using Add Status tasks ....................................................... 3-39
Insert and configure an Add Status task ................................. 3-40

Drag and drop a task .......................................................... 3-40

Cut and paste a task .......................................................... 3-41

Delete a task ................................................................. 3-42
Chapter

3 Adding tasks to workflow process templates

Workflow task actions and states

A task is the building block used to construct a workflow process template. Each task defines a set of actions, rules, and resources used to accomplish that task, and every task is always in one of seven defined states. Each instance of a task uses a task template, enabling you to use each task template as a blueprint for creating multiple tasks.

When workflow process templates are used in run time, that is, when the templates are used to run an actual workflow process in Workflow Viewer or My Teamcenter, the workflow process moves through actions and states.

- **Actions**
  Transition a task from one state to another. The goal for each task is to eventually reach the **Completed** state.

- **States**
  Control and coordinate the execution of each individual task in a workflow process.

The workflow process is run by the state transition engine. This engine controls workflow process flow by:

- Executing handlers and related internal logic.
- Setting tasks to their required state, based on task execution results.
- Placing workflow tasks in the appropriate **My Worklist** folders.

The following graphic illustrates how the workflow states and actions interact. States are circled, actions are designated by arrowed lines, indicating the direction the action moves one state to another.
The following table lists the possible beginning states each action can transition from, and the possible ending states each action can transition to:

<table>
<thead>
<tr>
<th>Action</th>
<th>Beginning state</th>
<th>Ending state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>Unassigned</td>
<td>Pending</td>
<td>Assigns a task to a responsible party.</td>
</tr>
<tr>
<td>Start</td>
<td>Pending</td>
<td>Started</td>
<td>Starts a task.</td>
</tr>
<tr>
<td>Complete</td>
<td>Started</td>
<td>Completed</td>
<td>Completes a task.</td>
</tr>
</tbody>
</table>
## Adding tasks to workflow process templates

<table>
<thead>
<tr>
<th>Action</th>
<th>Beginning state</th>
<th>Ending state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perform</strong></td>
<td>Any state</td>
<td>Any state</td>
<td>Executes any handlers placed on the <strong>Perform</strong> action. For interactive tasks, displays the appropriate perform dialog box for that task. This action does not transition a task’s state. This action can be performed multiple times on any given task, and can be triggered by both the state transition engine and by handlers.</td>
</tr>
<tr>
<td><strong>Suspend</strong></td>
<td>Any state</td>
<td>Suspended</td>
<td>Puts a task on hold.</td>
</tr>
<tr>
<td><strong>Resume</strong></td>
<td>Unassigned</td>
<td>Any state</td>
<td>Resumes a suspended task by returning the task to its previous state.</td>
</tr>
<tr>
<td><strong>Skip</strong></td>
<td>Started, Completed, Skipped, Failed</td>
<td>Skipped</td>
<td>Bypasses the current task and starts the successor task(s).</td>
</tr>
<tr>
<td><strong>Undo</strong></td>
<td>Started, Completed, Skipped, Failed</td>
<td>Pending</td>
<td>Undoes a task by returning the task to the <strong>Pending</strong> state.</td>
</tr>
<tr>
<td><strong>Fail</strong></td>
<td>Started</td>
<td>Failed</td>
<td>Indicates a task configured with a failed path is unsuccessful in fulfilling its requirements.</td>
</tr>
<tr>
<td><strong>Abort</strong></td>
<td>Any state</td>
<td>Aborted</td>
<td>Cancels a task without attempting to complete it.</td>
</tr>
</tbody>
</table>

An example of how *actions* and *states* work is that when a **Start** action is triggered on a task, all the handlers placed on that action are executed in the order listed. If the handlers all complete successfully, then the task’s state transitions to **Started**. The **Complete** action is automatically triggered on the task and all the handlers placed on that action are executed in the order listed. If the handlers all complete successfully, the task’s state transitions to **Complete**. The system attempts to start the successor tasks.
## Task template definitions

This table lists the task templates available in Workflow Designer. Click the task template name for step-by-step instructions on adding the task template to a workflow process template.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Task template</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌟</td>
<td>Do Task</td>
<td>Has two options if at least one failure path is configured: <strong>Complete</strong> confirms the completion of a task and triggers the branching to a success path. <strong>Unable to Complete</strong> indicates the task is unable to complete, for various reasons. Uses the <strong>EPM-hold</strong> handler, which stops the task from automatically completing when started.</td>
</tr>
<tr>
<td><img src="image" alt="Acknowledge Task" /></td>
<td>Acknowledge Task</td>
<td>Uses the <strong>Acknowledged</strong> and <strong>Not Acknowledged</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td><img src="image" alt="Review Task" /></td>
<td>Review Task</td>
<td>Uses the <strong>select-signoff-team</strong> and <strong>perform-signoffs</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td><img src="image" alt="Route Task" /></td>
<td>Route Task</td>
<td>Uses the <strong>Review</strong>, <strong>Acknowledge</strong>, and <strong>Notify</strong> subtasks, each of which has its own dialog box.</td>
</tr>
<tr>
<td><img src="image" alt="Task" /></td>
<td>Task</td>
<td>Use it as a starting point for creating your own custom tasks, such as tasks to carry your custom forms or other site-specific tasks for users to complete. This task template is synonymous with the <strong>EPMTask</strong> template.</td>
</tr>
<tr>
<td><img src="image" alt="Impact Analysis Task" /></td>
<td>Impact Analysis Task</td>
<td>Provides an impact analysis for a user to complete for the associated EC revision. The task provides <strong>Reference</strong>, <strong>Impact Analysis Form</strong>, <strong>Viewer</strong>, and <strong>Task Info</strong> tabs.</td>
</tr>
</tbody>
</table>

The **Impact Analysis Task** template is for use in EC processes only. It cannot be used on a workflow process.
## Adding tasks to workflow process templates

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Task template</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Prepare ECO Task" /></td>
<td><strong>Prepare ECO Task</strong></td>
<td>Provides EC requests or EC orders for a user to complete. The task provides <strong>ECO Sample</strong> and <strong>Task Info</strong> tabs. The <strong>Prepare ECO Task</strong> template is for use in EC processes only. It cannot be used on a workflow process.</td>
</tr>
<tr>
<td><img src="image" alt="Checklist Task" /></td>
<td><strong>Checklist Task</strong></td>
<td>Provides a checklist for a user to complete. The checklist form is a form type with a number of logical fields. You can create a custom form type with a site-specific field list using Java code to represent the form as a checklist. The task provides <strong>Check List</strong> and <strong>Task Info</strong> tabs. The <strong>Checklist Task</strong> template is for use in EC processes only; it cannot be used on a workflow process.</td>
</tr>
<tr>
<td><img src="image" alt="Condition Task" /></td>
<td><strong>Condition Task</strong></td>
<td>Branches a workflow according to defined query criteria. Requires that the succeeding task contains a <strong>check-condition</strong> handler that accepts a Boolean value of either <strong>True</strong> or <strong>False</strong>.</td>
</tr>
<tr>
<td><img src="image" alt="Validate Task" /></td>
<td><strong>Validate Task</strong></td>
<td>Branches a workflow along two or more paths. Active paths flowing out of the task are determined by whether specified workflow errors occur. Use this task to design workflows around anticipated errors.</td>
</tr>
<tr>
<td><img src="image" alt="Add Status Task" /></td>
<td><strong>Add Status Task</strong></td>
<td>Creates and adds a release status to the target objects of the workflow process. It is a visual milestone in a workflow process. No dialog box is associated with this type of task.</td>
</tr>
<tr>
<td><img src="image" alt="Or Task" /></td>
<td><strong>Or Task</strong></td>
<td>Continues the workflow process when any one of its multiple task predecessors is completed or promoted. There is no limit to the number of predecessors an or task may have.</td>
</tr>
</tbody>
</table>

## Using Do tasks

Use the **Do** task to define actions for a user to complete. When this task is performed in a workflow process, it displays the required actions to the user in the **Instruction** box of the task.

If you require user authentication before this **Do** task is performed, add the **require-authentication** handler to the **Perform** action of the task. When you
implement user authentication for this task, a password box appears below the **Comments** box. Users must type their user password in this box before they can click **Apply** and complete the task.

After completing the instructions, the user must select the **Complete** check box. The task does not complete until the user selects the check box. (This task is automatically configured with the **EPM-hold** handler to stop the task from completing until the check box is selected.) When the user selects the check box, the task sets the handler's argument to **False** and changes the status to **Complete**.

If the task is configured with a failure path the user can select one of the following check boxes:

- **Complete** confirms the completion of the task and continues the workflow down the success path.

- **Unable to Complete** indicates the user is unable to complete the instructions and continues the workflow down the failure path.

**Insert and configure a Do task**

1. On the toolbar, click **Edit Mode**.

2. On the toolbar, click **Do Task**.

3. In the process flow pane, double-click where you want to place the new **Do** task.

   A new **Do** task appears with the default name of **New Do Task #**, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the **Name** box, type a new name for the task.

5. (Optional) In the **Instructions** box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.

   For more information about linking this task to predecessor and successor tasks, see **Link tasks manually**.

7. (Optional) Configure task attributes by clicking **Task Attributes** in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

   For more information about using the **Task Attributes** dialog box, see **Task attributes**.

8. Configure task handlers by clicking **Task Handlers** in the template manager pane.

   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mails, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.
For more information about using the Task Handlers dialog box, see Task handlers.

When this task is performed in a workflow process, it displays required actions to the user in the Instruction box of the task. After completing the specified action, the user must select the Complete check box.

If the task is configured with a failure path, the user can select one of the following check boxes:

- **Complete** confirms the completion of the task and continues the workflow down the success path.
- **Unable to Complete** indicates the user is unable to complete the instructions and continues the workflow down the failure path.

### Using Acknowledge tasks

Use the Acknowledge task to define the Signoff Team profiles with which a user complies to assign acknowledgement responsibilities to other users. This template also provides the perform-signoffs task for the Signoff Team members to complete.

**Caution**

Do not add or delete subtasks from the Acknowledge task. It may cause an error that prevents the task from executing.

When this task is performed in a workflow process, the Acknowledge task displays two decision commands to members of the selected signoff team: Acknowledged and Not Acknowledged. Signoff team members choose one of the above commands to perform the signoff.

If you require user authentication before this Acknowledge task is performed, add the require-authentication handler to the Perform action of the task. When you implement user authentication for this task, a password box appears below the Comments box. Users must type their user password in this box before they can click Apply and complete the task.

### Insert and configure an Acknowledge task

1. On the toolbar, click Edit Mode.

2. On the toolbar, click Acknowledge Task.

3. In the process flow pane, double-click where you want to place the new Acknowledge task.
   
   A new Acknowledge task appears, with a default name of New Acknowledge Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type the actions the user must perform.
6. Explicitly link the task to the predecessor tasks.
   For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.
   For more information about using the Task Attributes dialog box, see Task attributes.

8. Configure task handlers by clicking Task Handlers in the template manager pane.
   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.
   For more information about using the Task Handlers dialog box, see Task handlers.

9. Define a signoff profile.
   a. Double-click the Acknowledge task in the task hierarchy tree.
      The task expands, listing the select-signoff-team and perform-signoffs subtasks.
   b. Select the select-signoff-team subtask, and then click the Task Signoff Panel button in the lower left of the Workflow Designer window.
      The Signoff Profiles dialog box appears.
   c. Select a group from the Group list.
   d. Select a role from the Role list.
      
      **Note**
      Define the signoff profiles by group or role, not by individual users. For example, if you want three managers from the Marketing group, all of the managers from the Engineering group, and 51% of the engineers from the Engineering group to sign off on this particular Acknowledge task, create three group profiles: a Marketing/manager profile, an Engineering/manager profile, and an Engineering/engineer profile.
      You can use the wildcard (*) to leave both the group and role category undesignated.
   e. Select or type the number of reviewers or percentage required for this particular group/role signoff profile.
      In the previous example, the Marketing/manager profile requires three reviewers, the Engineering/manager profile requires all reviewers, and the Engineering/engineer profile requires 51% of reviewers.
f. Select the **Allow sub-group members** check box to grant members of subgroups permission to sign off instead of members of the designated group.

g. Click **Create** to add this profile to the **Signoff Profiles** list.

h. Click **Modify** to change an existing profile in the **Signoff Profiles** list.

i. Click **Delete** to delete an existing profile in the **Signoff Profiles** list.

10. Select and type the number or percentage of reviewers required to satisfy a quorum.

You can designate the number or percentage of reviewers required for the quorum to be between one and the total number of users required for the selected signoff. The default setting is **Numeric** and the value is **All**. Select **Wait for Undecided Reviewers** if you want all of the required users to have a chance to review and comment before the workflow process can be rejected or approved.

11. After you add all the customer profiles, close the **Signoff Profiles** dialog box by clicking **Close** in the upper right corner of the dialog box.

### Using Review tasks

Use the **Review** task to route workflow targets (documents, parts, designs, and so on) for review.

The task includes two subtasks:

- **The select-signoff-team** subtask requires the workflow process initiator to select the users who will perform the review (the signoff team). You can configure this subtask with predefined group/role profiles that the workflow process initiator must select or allow the workflow process initiator to select users of his choice in an ad hoc manner.

  This subtask uses selection functionality from the Organization application, allowing the selector to search by group/role/user and to select signoff members individually or by project teams or address lists.

- **The perform-signoffs** subtask is then distributed to the selected signoff team, prompting them to review the target objects and signoff.

**Caution**

Do not add or delete subtasks from the Review task. It may cause an error that prevents the task from executing.

When this task is performed in a workflow process, the **perform-signoffs** task displays three options to each signoff team member: **Approve**, **Reject**, and **No Decision**. Selecting either **Approve** or **Reject** performs the task. **No Decision** is the default selection, selecting this option does not perform the task.

If you require user authentication before this **Review** task can be performed, add the **require-authentication** handler to the **Perform** action of the task. When you implement user authentication for this task, a password box appears below the **Comments** box. Users must type their user password in this box before they can click **Apply** and complete the task.
Chapter 3  Adding tasks to workflow process templates

Insert and configure a Review task

Caution
Do not add or delete subtasks from the Review task. It may cause an error that prevents the task from executing.

1. On the toolbar, click Edit Mode.

2. On the toolbar, click Review Task.

3. In the process flow pane, double-click where you want to place the new Review task.
   A new Review task displays with a default name of New Review Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.
   For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.
   For more information about using the Task Attributes dialog box, see Task attributes.

8. Configure task handlers by clicking Task Handlers in the template manager pane.
   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.
   For more information about using the Task Handlers dialog box, see Task handlers.

9. Define a signoff profile.
   • Double-click the Review task in the task hierarchy tree.
     The task expands, listing the select-signoff-team and perform-signoffs subtasks.
   • Select the select-signoff-team subtask, and then click Task Signoff in the lower left of the Workflow Designer pane.
     The Signoff Profiles dialog box appears.
Select a Group and Role.

**Note**

Define the signoff profiles by group or role, not by individual users. For example, if you want three managers from the Marketing group, all managers from the Engineering group, and 51% of the engineers from the Engineering group to sign off on this particular Review task, create three group profiles: a Marketing/manager profile, an Engineering/manager profile, and an Engineering/engineer profile.

You can use the wildcard (*) to leave both the group and role category undesignated.

Select and type the number or percentage of reviewers required for this particular group/role signoff profile. In the previous example, the Marketing/manager profile requires three reviewers, the Engineering/manager profile requires all reviewers, and the Engineering/engineer profile requires 51% of reviewers.

Select the Allow sub-group members check box to grant members of subgroups permission to sign off instead of members of the designated group.

Click Create to add this profile to the Signoff Profiles list.

Click Modify to change an existing profile in the Signoff Profiles list.

Click Delete to delete an existing profile in the Signoff Profiles list.

Select and enter the number or percentage of reviewers required to satisfy a quorum.

You can designate the number or percentage of reviewers required for the quorum, to be between one and the total number of users required for the selected signoff. The default setting is Numeric with the value of All. Select Wait for Undecided Reviewers if you want all of the required users to have a chance to review and comment before the workflow process can be rejected or approved.

After you add all the customer profiles, close the Signoff Profiles dialog box by choosing Close in the upper right corner of the dialog box.

**Using Route tasks**

Use the Route task as a router sheet with which a user assigns review, acknowledge and notification responsibilities to specified users.

When this task is performed in a workflow process, the Route task displays three subtasks: Review, Acknowledge, and Notify. The workflow process initiator can then assign other users to perform these tasks. The selected users are the signoff team.

**Caution**

Do not add or delete subtasks from the Route task. It may cause an error that prevents the task from executing.
Chapter 3  Adding tasks to workflow process templates

After the Route task is performed, the selected signoff team is prompted to perform the Review or Acknowledge tasks or simply notified of the review through program mail. Notified users do not need to perform any task.

If you want to require user authentication before the Review or Acknowledge subtasks can be performed, add the require-authentication handler to the Perform action of the subtask (the perform-signoffs task of either the Review or Acknowledge subtasks). When you implement user authentication for either of these subtasks, a password box appears below the Comments box. Users must type their user password in this box before they can click Apply and complete the task.

Insert and configure a Route task

1. On the toolbar, click Edit Mode.

2. On the toolbar, click Route Task.

3. In the process flow pane, double-click where you want to place the new Route task node.

   A new Route task node displays with a default name of New Route Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type any instructions for the task.

   Warning

   The Route task template is designed to be used as an electronic routing sheet. The workflow process initiator assigns specific signoff members. Signoff profiles for the Review and Acknowledge subtasks should not be defined within this task template. The task template does not function properly if signoff profiles are defined at this stage.

6. Explicitly link the task to the predecessor tasks.

   For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

   For more information about using the Task Attributes dialog box, see Task attributes.

8. Configure task handlers by clicking Task Handlers in the template manager pane.

   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks.
Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.

For more information about using the Task Handlers dialog box, see Task handlers.

Using generic tasks

The Task task is the default task template. Use it as a starting point for creating your own custom tasks, such as tasks to carry your custom forms or other site-specific tasks for users to complete. The Task task is synonymous with the EPMTask template.

Insert and configure a Custom task

1. On the toolbar, click Edit Mode.

2. On the toolbar, click Task.

3. In the process flow pane, double-click where you want to place the new Custom task.

   A new task appears, with a default name of New Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.

   For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

   For more information about using the Task Attributes dialog box, see Task attributes.

8. Configure task handlers by clicking Task Handlers in the template manager pane.

   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.

   For more information about using the Task Handlers dialog box, see Task handlers.
Chapter 3  Adding tasks to workflow process templates

Using Impact Analysis tasks

The **Impact Analysis** task is used for engineering change (EC) processes only. It cannot be used on a workflow process. Use this task to define an impact analysis for users to complete for the associated EC revision.

When this task is performed in a workflow process, the **Perform Impact Analysis Task** task displays four tabs:

- **Reference** tab
  Users can perform **Where Used** and **Where Referenced** searches in the **Reference** view to determine whether additional affected and solution items should be added to the EC revision. This information can be used to complete the **Impact Analysis** form.

- **Impact Analysis Form** tab
  Users can complete the defined **Impact Analysis** form.

- **Viewer** tab
  Users can complete any other forms associated with the EC revision which are available for completion in this view.

- **Task Info** tab
  Users complete the task by clicking the **Task Info** tab and selecting the **Done** check box.

By default, an **EPM-create-form** action handler is part of the **Start** action of the task that creates an instance of the **Impact Analysis** form. If you want to create instances of other forms, you can add an **EPM-create-form** handler for each additional form.

Add the **EPM-display-form** action handler to the **Perform** action to display the **Impact Analysis** form. The **EPM-hold** handler on the **Complete** action prevents the task from automatically completing, allowing the form to be completed by the user.

You can create customized rule handlers that prevents the task from completing until required boxes are entered in the form.

**Warning**

This task template is an ECM template. It can only be added to an EC process. This template cannot be added to a workflow process.

Insert and configure an Impact Analysis task

1. On the toolbar, click **Edit Mode**.

2. On the toolbar, click **Impact Analysis** task.

3. In the process flow pane, double-click where you want to place the new **Impact Analysis** task.
Adding tasks to workflow process templates

A new **Impact Analysis** task displays with a default name of **New Impact Analysis Task #**, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the **Name** box, type a new name for the task.

5. (Optional) In the **Instructions** box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.
   For more information about linking this task to predecessor and successor tasks, see **Link tasks manually**.

7. (Optional) Configure task attributes by clicking **Task Attributes** in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.
   For more information about using the **Task Attributes** dialog box, see **Task attributes**.

8. Configure task handlers by clicking **Task Handlers** in the template manager pane.
   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.
   For more information about using the **Task Handlers** dialog box, see **Task handlers**.

9. In the **Handlers** dialog box, select **EPM-create-form** in the **Start** folder.

10. Add handler arguments or change the value of the existing arguments. You can set default values for the form boxes by adding **-default** arguments to the **Argument** list.

11. Select the **Perform** folder and **EPM-display-form** in the **Action Handler** list.

12. Add the **-type** argument and its value to the **Argument** list. You can also add the **-form** argument if its default value is incorrect for your form.
   For more information about these arguments, see **EPM-display-form**.

13. Click the **Create** button under the **Argument** list.

14. Close the **Handlers** dialog box.

**Using Prepare ECO tasks**

The **Prepare ECO** task is used for engineering change (EC) processes only. It cannot be used on a workflow process. Use the task to define EC requests or EC orders for users to complete.
Chapter 3  Adding tasks to workflow process templates

When this task is performed in a workflow process, the Prepare ECO task displays two tabs:

- **ECO Sample** tab
  Users can complete the defined ECO Sample form.

- **Task Info** tab
  Users complete the task by selecting the Task Info tab and selecting the Done check box.

By default, an EPM-create-form action handler is part of the Start action of the task that creates an instance of the ECO Sample form. If you want to create instances of other forms, you can add an EPM-create-form handler for each additional form.

Add the EPM-display-form action handler to the Perform action to display the ECO Sample form. The EPM-hold handler on the Complete action prevents the task from automatically completing, allowing the form to be completed by the user.

You can create customized rule handlers that prevents the task from completing until required boxes are entered in the form.

**Warning**

This task template is an ECM template. It can only be added to an EC process. This template cannot be added to a workflow process.

**Insert and configure a Prepare ECO task**

1. On the toolbar, click Edit Mode 📐.

2. On the toolbar, click Prepare ECO Task 📝.

3. In the process flow pane, double-click where you want to place the new Prepare ECO task.

   A new Prepare ECO task displays with a default name of New Prepare ECO Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.

   For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes 📊 in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

   For more information about using the Task Attributes dialog box, see Task attributes.
8. Configure task handlers by clicking Task Handlers in the template manager pane.

Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.

For more information about using the Task Handlers dialog box, see Task handlers.

9. In the Handlers dialog box, select EPM-create-form in the Start folder.

10. Add handler arguments or change the value of the existing arguments. You can set default values for the form boxes by adding -default arguments to the Argument list.

11. Select the Perform folder and EPM-display-form in the Action Handler list.

12. Add the -type argument and its value to the Argument list. You can also add the -form argument if its default value is incorrect for your form.

For more information about these arguments, see EPM-display-form.

13. Click the Create button under the Argument list.


Using Checklist tasks

The Checklist task is used for engineering change (EC) processes only. It cannot be used on a workflow process. Use the task to define a checklist for users to complete. For example, define a checklist of tasks to be completed before the selected EC process can continue. The checklist form is a form type with a number of logical fields. You can create a custom form type with a site-specific field list using Java code to represent the form as a checklist.

When this task is performed in a workflow process, the Perform Checklist task displays two tabs:

- Checklist tab
  Users can complete the defined Checklist form.

- Task Info tab
  Users complete the task by clicking the Task Info tab and selecting the Done check box.

By default, an EPM-create-form action handler is part of the Start action of the task that creates an instance of the Checklist form. If you want to create instances of other forms, you can add an EPM-create-form handler for each additional form.
Add the **EPM-display-form** action handler to the **Perform** action to display the **Checklist** form. The **EPM-hold** handler on the **Complete** action prevents the task from automatically completing, allowing the form to be completed by the user.

You can create customized rule handlers that prevents the task from completing until required boxes are entered in the form.

**Warning**

This task template is an ECM template. It can only be added to an EC process. This template cannot be added to a workflow process.

### Insert and configure a Checklist task

1. On the toolbar, click **Edit Mode**.
2. On the toolbar, click **Checklist Task**.
3. In the process flow pane, double-click where you want to place the new **Checklist** task.
   
   A new **Checklist** task displays with a default name of **New Checklist Task #**, where # is incremented until the task name becomes unique within this workflow process template.
4. (Optional, but recommended) In the **Name** box, type a new name for the task.
5. (Optional) In the **Instructions** box, type the actions the user must perform.
6. Explicitly link the task to the predecessor tasks.
   
   For more information about linking this task to predecessor and successor tasks, see **Link tasks manually**.
7. (Optional) Configure task attributes by clicking **Task Attributes** in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.
   
   For more information about using the **Task Attributes** dialog box, see **Task attributes**.
8. Configure task handlers by clicking **Task Handlers** in the template manager pane.
   
   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.
   
   For more information about using the **Task Handlers** dialog box, see **Task handlers**.
9. In the **Handlers** dialog box, select **EPM-create-form** in the **Start** folder.
Adding tasks to workflow process templates

10. Add handler arguments or change the value of the existing arguments. You can set default values for the form boxes by adding -default arguments to the Argument list.

11. Select the Perform folder and EPM-display-form in the Action Handler list.

12. Add the -type argument and its value to the Argument list. You can also add the -form argument if its default value is incorrect for your form.
   For more information about these arguments, see EPM-display-form.

13. Click the Create button under the Argument list.


Using Condition tasks

Use the Condition Task template to branch your workflow process according to defined criteria. Because this task template is used to branch workflow process flow, you must always create at least two paths branching off from the task. The paths can be either success paths, failure paths, or a combination of the two.

- Success paths can be either true paths, false paths, or paths with a customized result.
   For more information about creating true paths and false paths, see Set Condition task paths.

- Failure paths can only be generated from manual Condition tasks. They allow an alternate course when a specified task is rejected, a user determines the path cannot be completed, or an error occurs.
   For more information about failure paths, see Creating failure paths.

Tip

If you use a Condition task to branch your workflow process, you can use one or more Or tasks later in the workflow process to resolve the paths into a single path.

The system determines which of the branches flowing from a Condition task to perform based on the task result. The task result is stored in the Condition task. The successor tasks have a handler configured with a value that may match the task result. After the task result is set, the successor tasks are examined and any successor tasks containing a value matching the task result are started. Use any of the following methods to set the task results:

- Create a query against the target (automatic only).
  For more information about creating queries, see Add a Condition task to a process template.

- Create a query against the task (automatic only).
  For more information about creating queries, see Add a Condition task to a process template.
Chapter 3  Adding tasks to workflow process templates

• Configure the task result from the manual **Condition** task’s dialog box.
  
  For more information, see *Set Condition task paths*.

A **Condition** task can be configured to complete either automatically or manually. You need to determine which configuration is best suited for the workflow process template you are defining. Typically, if a handler can determine the criteria, it is best to configure the task as automatic.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic <strong>Condition</strong> task</td>
<td>Add an action handler that sets the task’s result to true, false, or a customized value. The simplest way to achieve this is to use the task template’s interface to define a condition query at design time; this automatically inserts the action handler. Alternatively, you can create a custom action handler that uses ITK to verify criteria. For more information, see <em>Creating automatic Condition tasks</em>.</td>
</tr>
<tr>
<td>Manual <strong>Condition</strong> task</td>
<td>During design, you do not define a query or add an action handler to the task template. Because no query is defined and no action handler is configured to set the task result, when the workflow process is run, the end user must manually indicate a value using an interactive dialog box. The value chosen by the end user is used to set the task result.</td>
</tr>
</tbody>
</table>

**Creating manual Condition tasks**

Condition tasks configured to proceed manually require a user action before the task can proceed to completion.

• When the workflow reaches this task’s **Start** action, the task appears in the selected user’s worklist.

• The user completes the instructions, defines the condition path as **True** or **False**, clicks **OK** to complete the task and allow the workflow to continue.

  You should type text in the **Task Instructions** box that poses a question or set of parameters that require a true or false answer.

• If the user selects **Unset**, the task does not complete.

Use a manual **Condition** task when it requires additional information from the user and cannot be automated.
Example

For example, the task may require a part temperature reading from a usage test. In this case, because the stress test results are not input into Teamcenter, the database cannot be queried on the resulting temperature range. Instead, you can create a manual Condition task whose instructions state: **Check part temperature. If more than 100°F, set to True.** The task displays in the assigned user's Inbox. The user can then carry out the instructions and set the condition path either to True (if the part temperature was more than 100°F) or to False (if the part temperature was less than 100°F).

Create a manual Condition task by inserting the Condition Task template into the workflow process. Do not define a condition query or any custom handler that defines a result for the task.

If you want to require user authentication before a manual Condition task can be performed, add the require-authentication handler to the Perform action of the task. When you implement user authentication for this task, a password box appears below the Comments box. Users must type their user password in this box before they can click Apply and complete the task.

Creating automatic Condition tasks

Condition tasks configured to proceed automatically act as visual milestones in the workflow process. There is no action for a user to perform, and therefore, no dialog box is associated with the automatic Condition task.

Use an automatic Condition task when a database query can be defined for the decision branch; whether a specific part review has been approved, for example. If all part reviews are tracked through workflow, this information is in the database. To determine if the review of a specific part came back approved or rejected, you can perform a database query.

Example

For example, use a Condition Task template to create a conditional task that routes to an approval form if a selected part has been approved, but routes to a request form if the same selected part has not been approved. This is accomplished by defining a query that asks: Has 00431/C been approved?

- If the query result is true, the workflow continues along the Condition task's true path, proceeding to a Do task containing instructions to complete an approval form.
- If the query result is false, the workflow continues along the Condition task's false path, proceeding to a Do task containing instructions to complete a Request for Change form.

Create an automatic Condition task by inserting the Condition Task template into the workflow process template and performing step 6 in Add a Condition task to a process template.

Alternatively, you can create a custom action handler that uses ITK to check for the required criteria, as long as the handler uses the EPM_set_condition_task_result ITK call to set the task result to true or false.
Chapter 3  Adding tasks to workflow process templates

Note
If the system encounters a problem with performing the query as defined for an automatic **Condition** task, it sends the task to the responsible party’s Inbox for manual completion.

Configuring Condition tasks
Do not have a true path and false path converge on the **Finish** node. Paths are explicitly **AND** tasks and need a successor task at the merge point to complete. Typically, an **Or** task, which is specifically configured to require only one predecessor path to complete for it to start, is used to join the two paths. However, you can also use a **Generic** task or another kind of task.

Do not place a **Condition** task as the last task in a workflow process. The **Finish** node is not a task and should not be linked as a successor task to the **Condition** task.

Add a Condition task to a process template

1. On the toolbar, click **Edit Mode**.

2. On the toolbar, click **Condition Task**.

3. In the process flow pane, double-click where you want to place the new **Condition** task.
   
   A new **Condition** task appears with a default name of **New Condition Task**#, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) Type a new name for the task in the **Name** box.

5. (Optional) Type any instructions for the task into the **Instructions** box. If this is a manual **Condition** task, these instructions should prompt for the configuration of the task’s true and false paths.

6. Create an automatic **Condition** task by creating a database query for the task by performing the following subtasks. Do not define a query if you want to create a manual **Condition** task.
   
   a. Click **Query**.
      
      The **Query** dialog box appears.

   b. Perform one of the following:
      
      • If the required query already exists, select the query from the query list.

      • If the required query does not exist, create a new query.

      For information about creating queries, see the **Query Builder Guide**.

   c. Select **All**, **Any**, or **None** to determine whether all, any, or none of the target attachments must meet the query criteria to set the **Condition** task’s result to **True**.
d. Click Assign to assign the query to the Condition task.

The query is assigned to the task and is performed when the task reaches a Started state.

7. Create two or more tasks to succeed the Condition task; the true/false condition paths link the Condition task to the succeeding tasks.

For information about creating true and false paths branching from the Condition task, see Set Condition task paths.

8. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

For more information about using the Task Attributes dialog box, see Task attributes.

9. Configure task handlers by clicking Task Handlers in the template manager pane.

Handlers are essential to designing flexible, complex workflows.

- Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks.

- Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.

For more information about using the Task Handlers dialog box, see Task handlers.

Set Condition task paths

Because Condition tasks are used to branch your workflow process according to defined criteria, you must always create at least two paths branching off from the task. The paths can be either success paths, failure paths, or a combination of the two.

To draw and configure success paths from a Condition task:

1. On the toolbar, click Edit Mode.

2. Create one or more tasks to succeed the Condition task.

For more information about the available tasks, see Task template definitions.

3. Select the Condition task, placing the cursor in the body of the task (not the blue bar at the top). Draw a path from the Condition task to the succeeding task by dragging the cursor to the succeeding task.

A blue path displays between the two tasks.

4. Right-click the path and select the desired path type.

- The Set Path to True Path option creates a forward-branching path. Creating this path automatically places a rule handler on the Condition task to
check the condition of the specified target. When the condition is True, the workflow process proceeds along this path.

- The Set Path to False Path option creates a forward-branching path. Creating this path automatically places a rule handler on the Condition task to check the condition of the specified target. When the condition is False, the workflow process proceeds along this path.

- The Set Custom Result option allows you to define a custom task result. Enter any string to define the task result.

For example, you could enter Production to indicate the workflow process flowing into a production-ready branch.

Note
If you select this option and want the Condition task to be automatically processed, you must ensure the task result is sent to the Condition task. You can do this either by writing custom code or using the EPM-set-task-result-to-property handler. Custom conditions can also appear as Manual condition options and appear as buttons in the Condition dialog box.

5. If you selected a true or false path, the flow path displays True or False, respectively.

If you defined a custom result, the flow path displays the string you entered. In this example, the flow path displays Production.

To draw a failure path from the Condition task, see Creating failure paths.

Create as many paths off of the Condition task as required for your workflow process. In this example, after creating a production-ready branch, you could create Design and Release branches by creating additional succeeding tasks and creating additional customized flow paths from the Condition task.

Using Validate tasks

The Validate task branches a workflow along two or more paths. The path followed is determined by whether specified errors occur during a workflow. Use this task to design workflows around anticipated errors (such as checked out targets), unexpected errors (such as failed scripts or failure of custom handlers), or to track any and all workflow errors.

Configure the Validate task by defining one or more success and failure paths flowing from the task. The success path is followed if no error occurs. The failure path is followed when errors occur.

When errors occur, you determine if the failure path is followed when:

- Any error occurs.

- Only when an error you specify on a list of error codes occurs.
Note

In the context of the Validate task, workflow error means any error generated by a workflow handler.

Configure the task to follow a failure path by pairing a workflow handler and an error code. Place a handler to be validated on the Validate task and then add the respective error code to the path's error list (or set the path to fail on any error).

For more information about locating error codes, see Find error codes.

Find error codes

All error codes are documented in the Integration Toolkit Function Reference. Error codes are grouped by module. For example, Application Encapsulation (AE) errors are listed within the AE module, Appearances errors are listed within the Appearances module, and so forth.

Most workflow errors are displayed within the Enterprise Process Modeling (EPM) module.

To display a list of error messages:
1. Go to the Help Library and open the Integration Toolkit Function Reference.
2. At the top of the page, select the Modules header.
3. In the Modules page, scroll down to the appropriate module.
   For example, to see all Enterprise Process Modeling (EPM) errors, scroll to EPM Errors and click the link.
4. The error page displays all errors for that module. Error numbers are defined as module base value + error code.
   For example, the EPM_internal_error error has an error code of EMH_EPM_error_base + 1.
5. To determine the error base value for the selected module:
   a. Return to the Modules page.
   b. Scroll down to EMH Constants and click the link.
   c. The Error Message Handler (EMH) Constants page displays the error base of each module.
      For example, the error base value of EMH_EMP_error_base is 33000.
      Thus, the error number for the EPM_internal_error error is the concatenation of the EPM modules error base (33000) and the error code (1), creating an error code of 33001.

Although using workflow (EPM) error codes with the Validate task may be the most common usage, the task works with any error code. You can add error codes from any module, or custom error codes, to the Results List.
For more information about configuring the **Validate** task with error codes, see *Add error codes*.

### Add error codes

After drawing a failure path between the **Validate** task and a successor task, you must specify how you want the failure path to respond to workflow errors.

The failure path can be configured to activate when:

- **Any** error occurs by selecting **Set To Error Path**.
  
  This option automatically configures the failure path to activate upon any error. No additional steps are required.

- **Specific** errors occur by selecting **Set Error Codes** and completing the following procedure.

1. Right-click the path you want to configure as a failure path.

2. Select **Set Error Codes** to specify which error codes you want the **Validate** task to check.

   The **Set Error Codes** dialog box appears.

3. In the **Set Error Codes** dialog box, select the **Branch on Selected Errors** option.

4. In the **Add or Remove Error Code** box, type an EPM error code. For example, type **32009 (RES_OBJECT_IS_RESERVED)** to ensure the failure path is followed whenever a target is not checked in.
Note
For more information about finding EPM error codes, see Find error codes.

5. Click **Add**  to add this error to the **Results List**.

6. Continue adding errors to the **Results List** until you have specified all the errors you want to cause the workflow process to follow the failure path.

7. Click **OK** to close the **Set Error Codes** dialog box.
   The selected path appears as a broken path, indicating it is now a failure path.
Insert and configure a Validate task

1. On the toolbar, click Edit Mode.

2. On the toolbar, click Validate Task.

3. In the process flow pane, double-click where you want to place the new Validate task.
   
   A new Validate task appears with the default name of New Validate Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type the actions the user must perform.

6. Explicitly link the predecessor task to the Validate task.
   
   For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.
   
   For more information about using the Task Attributes dialog box, see Task attributes.

8. Configure task handlers by clicking Task Handlers in the template manager pane.
   
   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.
   
   For more information about using the Task Handlers dialog box, see Task handlers.

Validate task example: Close gaps in your workflow

At Design, Inc., employees check out documents that are targets of workflows and sometimes neglect to check them back in. Teamcenter does not allow users to initiate a workflow process on a target that is checked out. However, at Design, Inc., no business rules prevent users from checking out targets after a workflow process is initiated. When the workflow reaches the review stage, and the required targets are checked out, the workflow cannot complete.

In this example, this situation is anticipated and the Validate task is used to provide a correction. The task is placed before the review stage of the workflow and configured to verify that all targets are checked in. If so, a success path is followed. If not, the workflow follows a failure path that includes an additional Do task assigned to a manager. The Do task instructs the manager to get the targets checked in, and
then complete the Do task. After the error condition is corrected, the Do task’s success path traverses back into the main workflow.

The Validate task is configured to validate whether targets are checked in by placing the CR-assert-targets-checked-in rule handler on the Start action, and specifying the target-checked-out error in the error list.

The following procedure illustrates how to configure the workflow in this example.

1. Choose File→New Root Template to create a new workflow process.

2. Type a name for the new workflow process in the New Root Template Name box and click OK.

   The workflow process template appears in the process flow pane.

3. On the toolbar, click Edit ⚑.

   This puts the application in Edit mode, allowing you to edit the workflow process template.

4. Insert a Do task into the workflow process by clicking the Do task button ⭐ on the toolbar, and then double-clicking in the process flow pane to the right of the Start node.

   The new Do task is inserted at the cursor point.

5. Draw a success path from the Start node to the Do task by placing the cursor in the body of the Start node and dragging it to the body of the Do task. By default, flow paths are success paths. No configuration is necessary to create a success path.

   For more information about drawing flow paths, see Link tasks manually.

6. Insert a Validate task 🚀 to the right of the Do task.

7. Draw a success path from the Do task to the Validate task.

8. Configure the Validate task to check whether the target is checked in by adding the CR-assert-targets-checked-in rule handler to the Start action:
Chapter 3  Adding tasks to workflow process templates

a. In the process flow pane, ensure the **Validate** task is still selected. In the **Template** view, click the **Handlers** button 📝.  
   The **Handlers** dialog box appears.

b. In the task action in the left-side of the dialog box, select the **Start** action.

c. In the right-side of the dialog box, select **Rule Handler** 📝 for the handler type.

d. In the **Rule Handler** list, select **CR-assert-targets-checked-in**. No handler arguments are required for this handler in this example.

e. Click **Create** at the bottom of the dialog box to add the handler to the **Start** action of the new **Validate** task.

f. Close the **Handlers** dialog box.

9. Insert a **Do** task 🌟 above and to the right of the **Validate** task. This is the first of the two successor tasks used in this example.

10. Rename the **Do** task by selecting the task in the task hierarchy tree, and then typing **Success** in the **Name** box in the template manager pane.

11. Draw a success path from the **Validate** task to the **Success** task.

12. Insert a **Do** task 🌟 below and to the right of the **Validate** task. This is the second of the two successor tasks uses in this example.

13. Rename this second successor task to **Failure (target checked-out)**.

14. Create a failure path between the **Validate** task and the **Failure (target checked-out)** task by placing the cursor in the body of the **Validate** task and dragging it to the body of the **Failure (target checked-out)** task.

15. Right-click the path you have just drawn. A list provides you with two options.  
   Selecting either option creates a **failure** path.

   For this example, select **Set Error Codes** to specify the specific error code you want the **Validate** task to validate.

   The **Set Error Codes** dialog box appears.

16. In the dialog box, type the EPM error code you want to cause the workflow process to follow the failure path. For this example, type **32009 (RES_OBJECT IS RESERVED)** to ensure the failure path is followed whenever a target is not checked in.

   **Note**

   For more information about finding EPM error codes, see **Find error codes**.

17. Click **Add** 🌟 to add this error to the **Results List**.
18. Click **OK** to close the **Set Error Codes** dialog box.

The selected path appears as a broken path, indicating it is now a **failure path**.

19. Insert another **Do** task ✶ after the **Failure (target checked-out)** task.

20. Rename the **Do** task to **Check in Targets**.

21. In the **Instructions** box of the **Check in Targets** task, type instructions directing the manager to ensure all workflow targets are checked in, and to then complete the task.

22. Draw a success path from the **Failure (target checked-out)** task to the **Check in Targets** task.

23. Reconcile the success and failure paths by inserting an **Or** task and linking it to the **Success** task (the final interactive task of the success path) and the **Check in Targets** task (the final interactive task of the failure path).

   - Click the **Or** task button ➔ on the toolbar, and then double-click in the process flow pane to the right of the **Success** and **Check in Targets** tasks.

   - Draw a flow path from the **Success** task to the **Or** task.

   - Draw a flow path from the **Check in Targets** task to the **Or** task.

24. Link the **Or** task to the **Finish** node to complete the workflow.

When the workflow is run, either the success or failure path is followed, depending on whether the **RES_OBJECT_IS_RESERVED** error is triggered.

For more information about the **Validate** task's behavior, see **Validate task behavior**.
**Validate task example: Improve user response time**

At Business Corporation, the product review process has become increasingly complicated. Different products require different sets of review documents and the exponential growth of the product line has generated twenty different review documents that can be chosen as workflow targets.

Over the past year, the Teamcenter administrator has had to demote and restart more than 100 review workflows because users have selected inappropriate target objects. The administrator has long used the **EPM-validate-target-objects** rule handler at the beginning of the workflow to display an error to the project initiator at the time the workflow is launched. But too often the initiator ignores or misunderstands the message. As Business Corporation review processes become more complex, more workflows stall as team members ignore the error as they launch the workflow, and team leads do not track the error logs in a timely manner.

The administrator solved this problem using the **Validate** task and backward branching. He added a **Validate** task to the workflow, with the **Validate** task configured to branch down the failure path when the **EPM_invalid_target_type** error occurs. The failure path branches backward to the **Select Proper Targets** task, prompting the workflow process initiator to select the correct target. Once the targets are correct, the workflow process continues down the success path.

The following procedure illustrates how to configure the workflow in this example:

1. Choose **File→New Root Template** to create a new workflow process.

2. Type a name for the new workflow process in the **New Root Template Name** box and click **OK**.
   
   The workflow process template appears in the process flow pane.

3. On the toolbar, click **Edit**.
   
   This puts the application in **Edit** mode, allowing you to edit the workflow process template.

4. Insert a **Do** task into the workflow process by clicking the **Do** task button on the toolbar, and then double-clicking in the process flow pane to the right of the **Start** node.
   
   The new **Do** task is inserted at the cursor point.
5. Rename the Do task by selecting the task in the task hierarchy tree, and then typing Select Proper Targets in the Name box in the template manager pane.

6. Draw a success path from the Start node to the Select Proper Targets task by placing the cursor in the body of the Start node and dragging it to the body of the Select Proper Targets task. By default, flow paths are success paths. No configuration is necessary to create a success path.

For more information about drawing flow paths, see Link tasks manually.

7. Insert a Validate task above the Select Proper Targets task and to the right of the Start node.

8. Draw a success path from the Select Proper Targets task to the Validate task by placing the cursor in the body of the Select Proper Targets task and dragging it to the body of the Validate task.

If proper targets are selected, the workflow flows from Select Proper Targets, through the Validate task, and on to the next task you create.

9. Draw a failure path back from the Validate task to the Select Proper Targets task by placing the cursor in the body of the Validate task and dragging it to the body of the Select Proper Targets task.

When proper targets are not select, the workflow branches backward to the Select Proper Targets task, prompting the user select proper targets.

10. Configure the path as a failure path by right-clicking the path you have just drawn. A shortcut menu provides you with two options. Selecting either option creates a failure path.

For this example, select Set Error Codes to specify the specific error code you want the Validate task to validate.

The Set Error Codes dialog box appears.

11. In the dialog box, type the EPM error code you want to cause the workflow process to follow the failure path. For this example, type 33127 (EPM_invalid_target_type ) to ensure the failure path is followed whenever a target is not checked in.

   Note

   For more information about finding error codes, see Find error codes.

12. Click Add to add this error to the Results List.

13. Click OK to close the Set Error Codes dialog box.

   The selected path appears as a broken path, indicating it is now a failure path.

14. Configure the Validate task to check whether correct target types have been selected by adding the EPM-validate-target-objects rule handler to the Start action:

   a. In the process flow pane, ensure the Validate task is still selected. In the Template view, click the Handlers button.
The **Handlers** dialog box appears.

b. In the task action in the left-side of the dialog box, select the **Start** action.

c. In the right-side of the dialog box, select **Rule Handler** for the handler type.

d. In the **Rule Handler** list, select **EPM-validate-target-objects**. No handler arguments are required for this handler in this example.

e. Click **Create** to add the handler to the **Start** action of the new **Validate** task.

f. Close the **Handlers** dialog box.

15. Insert a **Do** task to the right of the **Validate** task.

16. Rename the **Do** task to **Targets OK**.

17. Draw a success path from the **Validate** task to the **Targets OK** task by placing the cursor in the body of the Validate task and dragging it to the body of the **Targets OK** task.

18. Link the **Targets OK** task to the **Finish** node to complete the workflow.

When the workflow is run, it cannot progress past the **Validate** task until the workflow targets are validated as correct. The workflow raises user awareness of incorrect targets by sending an interactive task to the workflow process initiator each time the **EPM_invalid_target_type** error occurs, prompting the user to select valid targets.

**Validate task example: Track errors from custom handlers**

Corporate Ltd. uses a workflow to manage its quarterly budget analysis and review. The workflow includes a custom handler that runs a script to generate and distribute a budget report from various Excel files. The custom handler was placed on the **Start** action of a **Do** task (named **Distribute Quarterly Budget**) immediately succeeding a **Review** task.

Occasionally the script cannot complete because of computation errors. The custom handler generates an error when the script cannot complete. But as the script runs overnight, the error does not immediately display. Because the error recipient (in this case, the workflow process initiator) is not logged in at time of error, the error does not redisplay when the user logs in. The result is that the workflow has stalled one or more days before the workflow process initiator notices the delay.

The Teamcenter administrator solved this problem by inserting a **Validate** task before the **Do** task and drawing a success path between them. Then the administrator inserted another **Do** task (named **Manually Compile/Distribute Quarterly Budget**) parallel to the first, connected it to the **Validate** task with a failure path and assigned the task to the lead accountant. The **Validate** task is configured to follow the failure path when the script error is thrown. Whenever the compilation script fails, the lead accountant is prompted to recompile the budget.
Because the **Validate** task can be configured to respond to any specific error, even errors thrown by custom handlers, the failure of the custom handler can be taken into consideration and managed.

The following procedure illustrates how to configure the workflow in this example:

1. Choose **File→New Root Template** to create a new workflow process.

2. Type a name for the new workflow process in the **New Root Template Name** box and click **OK**.
   
The workflow process template appears in the process flow pane.

3. On the toolbar, click **Edit**.
   
   This puts the application in **Edit** mode, allowing you to edit the workflow process template.

4. Insert a **Review** task into the workflow process by clicking the **Review** task button on the toolbar, and then double-clicking in the process flow pane to the right of the **Start** node.
   
The new **Review** task is inserted at the cursor point.

5. Rename the **Review** task by selecting the task in the task hierarchy tree, and then typing **Review/Request Funding** in the **Name** box in the template manager pane.

6. Draw a success path from the **Start** node to the **Review/Request Funding** task by placing the cursor in the body of the **Start** node and dragging it to the body of the **Review/Request Funding** task. By default, flow paths are success paths. No configuration is necessary to create a success path.
   
   For more information about drawing flow paths, see **Link tasks manually**.
7. Insert a Validate task to the right of the Review/Request Funding task.

8. Draw a success path from the Review/Request Funding task to the Validate task by placing the cursor in the body of the Review/Request Funding task and dragging it to the body of the Validate task.

9. Configure the Validate task to check whether the script fails by adding the custom handler used to run the budget-compilation script to the Start action:
   a. In the process flow pane, ensure the Validate task is still selected. In the Template view, click the Handlers button .
      The Handlers dialog box appears.
   b. In the task action in the left-side of the dialog box, select the Start action.
   c. In the right-side of the dialog box, select Action Handler for the handler type.
   d. In the Action Handler list, type budget-compilation. No handler arguments are required for this handler in this example.
   e. Click Create at the bottom of the dialog box to add the handler to the Start action of the new Validate task.
   f. Close the Handlers dialog box.

10. Insert a Do task above and to the right of the Validate task. This is the first of the two successor tasks used in this example.

11. Rename the Do task to Distribute Quarterly Budget.

12. Draw a success path from the Validate task to the Distribute Quarterly Budget task by placing the cursor in the body of the Validate task.

13. Insert another Do task above the Distribute Quarterly Budget task. This is the second of the two successor tasks used in this example.

14. Rename this second successor task Manually Compile/Distribute Quarterly Budget.

15. In the Instructions box of the Manually Compile/Distribute Quarterly Budget task, type instructions directing the lead accountant to manually compile and distribute the budget report, then to complete the task.

16. Create a failure path between the Validate task and the Manually Compile/Distribute Quarterly Budget task by placing the cursor in the body of the Validate task and dragging it to the body of the Manually Compile/Distribute Quarterly Budget task.

17. Right-click the path you have just drawn. A list provides you with two options. Selecting either option creates a failure path.
For this example, select **Set Error Codes** to specify the specific error code you want the **Validate** task to validate.

The **Set Error Codes** dialog box appears.

18. In the dialog box, type the custom error code you want to cause the workflow process to follow the failure path. For this example, type **99001** (custom error budget-compilation).

   **Note**

   For more information about finding EPM error codes, see **Find error codes**.

19. Click **Add** + to add this to the **Results List**.

20. Click **OK** to close the **Set Error Codes** dialog box.

   The selected path appears as a broken path, indicating that it is now a **failure** path.

21. Reconcile the success and failure paths by inserting a generic task and linking it to the **Distribute Quarterly Budget** task (on the success path) and the **Manually Compile/Distribute Quarterly Budget** task (on the failure path).

   • Click the **Task** task button 📄 on the toolbar, then double-click in the process flow pane to the right of the **Distribute Quarterly Budget** and **Manually Compile/Distribute Quarterly Budget** tasks.

   The new generic task is inserted at the cursor point.

   • Rename the generic task **Quarterly Meeting**.

   • Draw a success path from the **Distribute Quarterly Budget** task to the **Quarterly Meeting** task.

   • Draw a success path from the **Manually Compile/Distribute Quarterly Budget** task to the **Quarterly Meeting** task.
22. In the **Instructions** box of the **Quarterly Meeting** task, type instructions directing the finance officer to host the cross-team finance meeting to discuss budget needs and to then complete the task.

23. Insert a **Route** task below the **Quarterly Meeting** task.

24. Rename the **Route** task to **Review and Approve Funding**.

25. In the **Instructions** box of the **Review and Approve Funding** task, type instructions directing the finance officer to route the revised budget requests to all stakeholders and interested parties.

26. Link the **Quarterly Meeting** task to the **Review and Approve Funding** task.

27. Link the **Review and Approve Funding** task to the **Finish** node to complete the workflow.

When the workflow is run, the success path is followed if the budget script successfully completes, or the failure path is followed if the script fails. This workflow raises user awareness of the script failure by having an interactive task sent to the lead accountant when this error occurs.

**Validate task behavior**

The **Validate** task’s behavior depends upon how its failure path is configured and what errors are received.

<table>
<thead>
<tr>
<th>Failure criteria you specified</th>
<th>Error thrown (if any)</th>
<th>Task behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail if any error</td>
<td>Any error</td>
<td>Failure path is followed.</td>
</tr>
<tr>
<td>Fail if error on error list occurs</td>
<td>Error on error list</td>
<td>Failure path is followed.</td>
</tr>
<tr>
<td>Fail if error on error list occurs</td>
<td>Error not on error list</td>
<td>Workflow process halts. Task remains in <strong>Started</strong> state and an error appears.</td>
</tr>
<tr>
<td>No failure path configured</td>
<td>Any error</td>
<td>Workflow process stops. Task remains in <strong>Started</strong> state and an error appears.</td>
</tr>
<tr>
<td>Regardless of whether failure path was configured, and whether errors occurred</td>
<td>No errors occur</td>
<td>Success path followed. If no success path was configured, workflow process stops.</td>
</tr>
</tbody>
</table>

**Using Or tasks**

Use an **Or** task template to continue the workflow process when any one of its multiple task predecessors is completed or promoted. There is no limit to the number of predecessors an Or task may have. Typically, **Or** tasks are used to unite parallel paths create by:
• True/false condition paths branching from **Condition** tasks.

• Parallel links branching from a single task.

This template is a visual milestone in the workflow process. There is no dialog box associated with the **Or** task.

**Insert and configure an Or task**

1. On the toolbar, click **Edit Mode**.

2. On the toolbar, click **Or task**.

3. Double-click the location in the process flow pane where you want to place the new **Or** task node.

   A new **Or** task node displays with a default name of **Or Task #**, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the **Name** box, type a new name for the task.

5. (Optional) In the **Instructions** box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.

   For more information about linking this task to predecessor and successor tasks, see **Link tasks manually**.

7. (Optional) Configure task attributes by clicking **Task Attributes** in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

   For more information about using the **Task Attributes** dialog box, see **Task attributes**.

8. Configure task handlers by clicking **Task Handlers** in the template manager pane.

   Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.

   For more information about using the **Task Handlers** dialog box, see **Task handlers**.

**Using Add Status tasks**

Use the **Add Status** task template to create and add a **Release** status to the target objects of the workflow process.
Chapter 3  Adding tasks to workflow process templates

This template is a visual milestone in the workflow process. There is no action for the user to perform, and therefore, no dialog box associated with the Add Status task.

Insert and configure an Add Status task

1. On the toolbar, click Edit Mode.

2. Click Add Status task.

3. Double-click the location in the process flow pane, where you want to place the new Add Status task node.

A new Add Status task node displays with a default name of New Add Status Task #, where # is incremented until the task name becomes unique within this workflow process template.

4. (Optional, but recommended) In the Name box, type a new name for the task.

5. (Optional) In the Instructions box, type the actions the user must perform.

6. Explicitly link the task to the predecessor tasks.

For more information about linking this task to predecessor and successor tasks, see Link tasks manually.

7. (Optional) Configure task attributes by clicking Task Attributes in the template manager pane. Use task attributes to manage task security, duration, display, and quorum behavior.

For more information about using the Task Attributes dialog box, see Task attributes.

8. Configure task handlers by clicking Task Handlers in the template manager pane.

Handlers are essential to designing flexible, complex workflows. Use action handlers to perform all types of digital actions, such as running scripts, sending e-mail, creating forms, and assigning responsibility for various workflow tasks. Use rule handlers to implement workflow rules, such as adding status, demoting tasks, displaying forms, and notifying workflow participants.

For more information about using the Task Handlers dialog box, see Task handlers.

Drag and drop a task

1. On the toolbar, click Edit.

2. In the process flow pane, identify the task you want to move. If the task has paths linking it to other tasks, delete the paths.

3. Select the task you want to move by clicking the blue title bar.
4. Drag the task to the desired location in the workflow process template.

5. Draw a path from the task you want to be the preceding task to the newly moved task. The path you draw, (also called an explicit link) determines the order in which tasks are performed.

**Note**

Moving tasks and their associated paths in the process flow pane changes the order in which tasks are performed. Using the process flow pane to manage task order is the recommended method.

It is important to note that the task hierarchy tree lists tasks in the order they were first created. This order is **not** altered as you change task order within the process flow pane. The order displayed in the task hierarchy tree does not indicate task execution order.

**Cut and paste a task**

1. On the toolbar, click **Edit**.

2. In the process flow pane, select the task you want to move by clicking the body of the task.

3. Click one of the following, as needed:
   - Click **Cut** if you want to remove the task from its current location and paste it elsewhere.
     
     The system removes the task from its location in the workflow process template and sends it to the clipboard.
   - Click **Copy** if you want a copy of the existing task to be pasted elsewhere.
     
     A copy of the task is sent to the clipboard.

4. Click **Paste**.

   The task is pasted to the upper left-hand corner of the process flow pane.

5. Select the newly pasted task by clicking the blue title bar.

6. Drag the task to the desired location in the workflow process template.

   For more information about linking the added task to the existing task nodes, see **Linking tasks in a workflow process template**.
Chapter 3  Adding tasks to workflow process templates

Note

Moving tasks and their associated paths in the process flow pane changes the order in which tasks are performed. Using the process flow pane to manage task order is the recommended method.

It is important to note that the task hierarchy tree lists tasks in the order they were first created. This order is not altered as you change task order within the process flow pane. The order displayed in the task hierarchy tree does not indicate task execution order.

Delete a task

1. On the toolbar, click Edit Mode.
2. Click the task node you want to delete.
   Once selected, the task bar turns blue.
3. Click Delete.
   The selected task and any attached links are deleted.

   Note
   If you do not replace the deleted links with explicit links, Workflow Designer creates assumed links for you.
Chapter

4  Linking tasks in a workflow process template

Linking tasks in a workflow process template .............................. 4-1
Refreshing Workflow Designer ...................................................... 4-1
Link tasks manually .................................................................. 4-2
Deleting tasks and links ............................................................... 4-2
Delete links ................................................................................ 4-2
Creating failure paths ................................................................. 4-3
4 Linking tasks in a workflow process template

Linking tasks in a workflow process template

A link establishes the sequence by which peer-level tasks are executed, indicating that the task on the arrow end of the path cannot start until the task on the start end is completed.

Explicit links   Manually created links, drawn from the predecessor task to the successor task.

Assumed links   Automatically created by the system if no explicit links have been created from the Start node by the time the template is set to the Available stage.

When you put a workflow template in Edit mode and draw a single link from the Start node to another task node, assumed link behavior is disabled. The system does not draw assumed links, even if you leave tasks unlinked and change the workflow template to the Available stage. Any unlinked tasks are skipped when a workflow process based on the workflow template is initiated, and no error messages appear.

Caution

When you place workflow templates created before Teamcenter 8.3 and 8.1.1.1 in Edit mode, the system removes all links originating from the Start node. If this occurs, manually redraw any removed links.

Refreshing Workflow Designer

You can refresh the display by:

• Moving up or down a level.

• Going to the top level.

• Choosing View→Refresh All.

• Setting the template to the Available stage.
Link tasks manually

Drawing a path between two tasks establishes the sequence in the execution of the tasks by declaring that the task on the arrow end of the link cannot start until the task on the start end of the link has been completed.

Manually drawing either success or failure paths between tasks creates explicit links between your tasks. Always explicitly link your tasks to ensure predictable results. Draw your success or failure path immediately after inserting tasks into the workflow process, before saving the workflow process or switching away from the Workflow Designer application. Saving the workflow process or switching applications before manually drawing paths prompts Teamcenter to automatically insert implicit links.

1. On the Workflow Designer toolbar, click Edit Mode.

2. Click the task node you want to be the predecessor task.
   Do not click the title bar of the task node: doing so begins a drag process.

3. Drag your cursor to the task node you want to be the successor task.
   A link arrow follows the cursor as you drag. When your cursor moves over a task node, the node is highlighted.

4. Release the mouse button.
   A link arrow connects the predecessor and successor nodes.

Deleting tasks and links

When you delete a task from a template, the system deletes its links along with the task. If you do not reestablish explicit links among the remaining tasks, the system creates assumed links.

Delete links

1. On the toolbar, click Edit Mode.

2. In the process flow pane, click the link you want to delete. The link turns blue.

3. Click Delete.
   The system deletes the selected link.

   **Note**
   If you do not replace a deleted link with an explicit link, Workflow Designer automatically creates a link from the Start node to each unlinked task.
Creating failure paths

A failure path gives an alternate course that a workflow process can follow in any of the following scenarios:

- A task is rejected.
- The user determines that the task cannot be completed.
- There is an error.

When creating a workflow, each path is configured as either a success path or a failure path. A failure path must be configured into the workflow process template at design time. A task follows the appropriate path based on the task’s outcome. A success path is traversed when a task’s state transitions to Complete or when a task is promoted and it transitions to a Skipped state. A task completes upon the successful execution of the task’s handlers on the Complete action.

Backward branching allows a path to be routed backward to some previous task in the workflow process flow, including the Start node. Both success and failure paths are capable of branching in a backward direction. Backward branching allows the re-execution of a task with a Complete or Skipped task state.

To create a failure path, right-click an arrow and select the appropriate failure option. Failure path options display differently for different tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Failure option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do</td>
<td>Set to Unable to Complete</td>
</tr>
<tr>
<td>Review</td>
<td>Set to Reject</td>
</tr>
<tr>
<td>Route</td>
<td>Set to Reject</td>
</tr>
<tr>
<td>Condition</td>
<td>Set to Unable to Complete</td>
</tr>
<tr>
<td>Validate</td>
<td>Set to Error Path</td>
</tr>
<tr>
<td>EPM</td>
<td>Set to Unable to Complete</td>
</tr>
</tbody>
</table>

This example shows the options for an existing Condition task failure path.
Chapter 4  

Linking tasks in a workflow process template
Chapter

5  Modifying task behavior

Modifying task behavior .................................................. 5-1
Edit task attributes ......................................................... 5-2
What are task handlers? ................................................. 5-4
View task handlers ........................................................ 5-4
Create task handlers based on existing handlers .................. 5-5
Create new task handlers ................................................. 5-6
Edit task handlers .......................................................... 5-7
Delete task handlers ....................................................... 5-7
Chapter

5  Modifying task behavior

Modifying task behavior

You can modify the behavior of a task within a workflow process template by using:

- **Attributes**
  Allows you to set requirements and/or restrictions on a task. Possible task attributes are:
  - Named ACL
  - Template name
  - Signoff quorum
  - Release status
  - Icons
  
  For more information, see *Edit task attributes*.

- **Handlers**:
  Small ITK programs or functions. Handlers are the lowest-level building blocks in EPM. You use handlers to extend and customize tasks. The following is a list of the types of functions you can add to a task:
  - Set protections
  - Assign reviewers
  - Demote a task
  - Perform a signoff
  - Change a status
  
  There are two kinds of handlers:
  - **Action handlers**:
    Extend and customize task actions. Action handlers perform such actions as displaying information, retrieving the results of previous tasks (inherit), notifying users, setting object protections and launching applications.

  - **Rule handlers**:
    Integrate workflow business rules into EPM workflow processes at the task level. Rule handlers attach conditions to an action.

    Many conditions defined by a rule handler are binary (that is, they are either true or false). However, some conditions are neither true nor false. EPM allows two or more rule handlers to be combined using logical **AND/OR** conditions. When several rule handlers are combined using a logical **OR** condition, rule handler quorums specify the number of rule handlers that must return go for the action to complete.
Chapter 5  Modifying task behavior

For more information, see What are task handlers?.

Edit task attributes

You can customize a task by editing its attributes.

1. On the Workflow Designer toolbar, click Edit Mode.

2. Click Task Properties in the toolbar.
   The system displays the Task Properties dialog box.
   The Name box lists the name of the selected workflow process template or task template.

3. (Optional) Type task instructions into the Instructions box.

4. Click the Attributes Panel tab.
   The system displays the Attributes Panel dialog box.

5. Click Named ACL to add permissions for the task and target objects.
   a. Use one of the following methods to select an ACL to apply to the task.
      • In the ACL Name box, select an existing ACL.
         o Click the system Named ACL button to list ACL names created in Access Manager.
         o Click the workflow Named ACL button to list ACL names created in Workflow Designer.
      • In the ACL Name box, type a new ACL name and click Create.
         The new ACL is added to the list of workflow named ACLs.
         A. Add access control entries (ACEs) to define the permissions for the named ACL.
         B. Click Save to save the ACEs for the named ACL.

         For information about creating a named ACL, see the Access Manager Guide.

         For information about workflow accessors and privileges, see the Security Administration Guide.

      b. Click Assign to ACL Name to update the Assigned ACL Name box.
         This action creates the EPM-set-rule-based-protection handler on the Start action for the task.
      c. (Optional) To verify the assignment, view the Task Handler panel.

6. If the selected task is a Condition task, you can:
   • Select a graphic from the Icons list.
• Click **Condition Query** to define a query.
  The system displays the **Condition Query** dialog box.

• Define a query for the Condition task. For information about defining
  queries, see *Query Builder Guide*.
  The **Duration** box displays the length of time allowed for the completion
  of the project. You can define the duration length in the template of the
  selected task. You can also define duration length in the **Attributes** dialog
  box when the selected task is in a **Pending** state.

7. To set the **Duration** box:
• Type an integer value for any or all of the following boxes to indicate
  the length of time that can pass before the selected tasks needs to reach
  completion:
  o Years
  o Weeks
  o Days
  o Hours
  o Minutes

• Click one of the following, as needed:
  o **OK** Saves the changes to the database and closes the dialog box.
  o **Clear** Clears all boxes.
  o **Cancel** Closes the dialog box without making any changes.

The **Recipients** list displays the names of users selected to receive program
mail when the selected task becomes overdue. You can set the **Recipients**
list from this dialog box.

8. To set the **Recipients** list:
• Click **Set** to the right of the **Recipient** box.
  The system displays the **Select Recipients** dialog box.

• Type the user, group, or address list search criteria for users you want
  to select.

• Based on the search criteria you entered, click either **User**, **Group**, or
  **Address List**.
  The search results display in the box below. To display all users in the
  selected grouping, type * and click the appropriate button. All users in the
  selected grouping display in the box.

• Select the users you want to define as recipients from the search results. You
  can choose multiple users by pressing Ctrl and clicking the desired names.

• Click **Users**.
  The selected users display in the box in the right side of the dialog box.
  These are the selected recipients.
• To delete a recipient, click Delete.
• Close the Named ACL dialog box.

Note
When a named ACL is applied to a task and the Named ACL dialog box is closed, the Show Task in Process Stage List property on the Tasks Attributes Panel is automatically selected.

- The Show Task in Process Stage List displays the task in the Process Stage List property for the target object.
- Tasks in the Process Stage List are used to determine the ACL for the target objects.

9. Select Show Task in Process Stage List to display the task in the Process Stage List property for the target object.
• Select the Show Task in Process Stage List property when a named ACL is defined for a task.
• Clear the Show Task in Process Stage List when there are no named ACL and EPM-set-rule-based-protection handler defined for this task, and the task does not need to appear in the target object Process Stage List. For example, clear this box for subtasks or parent tasks.

Note
The Process Stage List also determines the task’s attributes, such as responsible party or signoff approvers, factored into the currently active named ACL.

10. Click Close to save the changes to the database and close the dialog box.

What are task handlers?
You can customize task behavior by creating and modifying task handlers. A task handler is a small ITK program or function. Handlers are the lowest level building blocks in EPM and are used to extend and customize tasks.

Note
Never add a CM handler to a workflow process. CM handlers are designed to be used only in change processes. For information about creating change processes, see the Change Viewer Guide.

View task handlers
You can display the task handlers of a selected task from Workflow Designer or from Workflow Viewer while in design mode by performing the following steps:

1. Click Browse Mode.
Modifying task behavior

2. Select the task whose handlers you want to view. To view handler information for the root task of the workflow process (the initial Start task) select the workflow process.

3. Click the Task Handlers pane.

   The system displays the Task Handlers dialog box. In the left pane, the handler tree lists the handlers assigned to the selected task.

   To more easily view the contents of the handler tree, you can click Expand All Folders or Collapse All Folders.

Create task handlers based on existing handlers

You can create new task handlers based on an existing handler. Use this procedure when one or more attributes of the new handler are contained in an existing handler. To create a handler, perform the following steps from the Task Handlers dialog box in either Workflow Designer or when in design mode in Workflow Viewer:

1. On the toolbar, click Edit Mode

2. Select the handler from the handler tree that you want to use as a template for the new handler.

   The Handler Type, Quorum, Task Action, and Action/Rule Handler boxes display the current settings for the selected handler.

3. Edit the data in the boxes as required for the new handler.

   If the selected task involves selecting signoff teams or performing signoffs, select and enter type the number or percentage required for the quorum in the Quorum box.

4. Edit existing arguments in the Argument table by selecting the value cell to the right of the argument cell and deleting the existing values. Add new value information by double-clicking in the cell to initiate the text-field editor, and then entering the required values.

   Separate multiple values by a comma.

5. Add a new argument row by clicking the Argument table. Type the new argument name into the argument cell by double-clicking in the cell to initiate the text-field editor, then entering the required argument name. Type the corresponding values into the value cell to the right of the argument cell by double-clicking in the cell to initiate the text-field editor, then entering the required values.

   Separate multiple values by a comma. You can display documentation for the selected handler by clicking Help.

6. Change the argument order by selecting an argument row and clicking Up or Down (located to the right of the table) to move the argument row up or down, respectively.
Chapter 5  *Modifying task behavior*

7. Change the handler order by selecting a handler in the handler tree and clicking **Up ▲** or **Down ▼** (located below the tree) to move the argument row up or down, respectively.

8. Click **Create** to create a new handler based on the data now displayed in the dialog box.

   The system creates the new handler and displays it in the handler tree.

**Create new task handlers**

You can create new task handlers with no preexisting data. Use this procedure when no existing handlers contain the necessary attributes. To create a new handler, perform the following steps from the Task Handlers dialog box in either Workflow Designer or when in design mode in Workflow Viewer:

1. Decide the type of handler you want to create:
   - **Rule handler**
     - Click **Rule Handler**.
   - **Action handler**
     - Click **Action Handler**.

2. If the selected task involves selecting signoff teams or performing signoffs, select and type the number or percentage required for the quorum in the **Quorum** box.

3. Select a handler from the **Action Handler** or **Rule Handler** list.

4. Add a new argument row by clicking **Add** next to the **Argument** table. Type the new argument name into the argument cell by double-clicking in the cell to initiate the text-field editor, then typing in the required argument name. Type the corresponding values into the value cell to the right of the argument cell by double-clicking in the cell to initiate the text-field editor, then entering the required values.

   Separate multiple values by a comma. You can display documentation for the selected handler by clicking **Help**.

5. Change the argument order by selecting an argument row and clicking **Up ▲** or **Down ▼** (located to the right of the table) to move the argument row up or down, respectively.

6. Change the handler order by selecting a handler in the handler tree and clicking **Up ▲** or **Down ▼** (located below the tree) to move the argument row up or down, respectively.

7. Click **Create** to create a new handler based on the data currently displayed in the handler's display area.

   The system creates the new handler and displays it in the handler tree.
Edit task handlers

To modify task handlers, you must edit the argument table. To edit a handler, perform the following steps from the Task Handlers dialog box in either Workflow Designer or when in design mode in Workflow Viewer:

1. Select the handler you want to edit from the handler tree.

The Handler Type, Quorum, Task Action and Action/Rule Handler boxes display the current settings for the selected handler.

2. If the selected task involves selecting signoff teams or performing signoffs, select and type the number or percentage required for the quorum in the Quorum box.

3. Edit existing arguments in the Argument table by deleting the existing values from the value cell to the right of the argument cell, and then double-clicking in the cell to initiate the text-field editor and entering the required values.

Separate multiple values by a comma. You can display documentation for the selected handler by clicking Help.

4. Change the argument order by selecting an argument row and clicking Up ▲ or Down ▼ (located to the right of the table) to move the argument row up or down, respectively.

5. Change the handler order by selecting a handler in the handler tree and clicking Up ▲ or Down ▼ (located below the tree) to move the argument row up or down, respectively.

6. Add a new argument to the Argument table.
   a. Type the new argument name in the argument cell by double-clicking in the cell to initiate the text-field editor, then entering the required argument name.
   
   b. Type the corresponding values in the value cell to the right of the argument cell by double-clicking in the cell to initiate the text-field editor, and then entering the required values.

Separate multiple values by a comma.

7. Click Modify to update the selected handler to reflect the data currently displayed in the handler's display area.

The system modifies the selected handler.

Delete task handlers

When a handler is no longer required, you can delete it as explained in this section. To delete a handler, perform the following steps from the Task Handlers dialog box in either Workflow Designer or when in design mode in Workflow Viewer:

- Select the desired handler from the handler tree and click Delete.

The system deletes the selected handler and no longer displays it in the tree.
Chapter

6 Using workflow templates at multiple Teamcenter sites

Using workflow templates at multiple Teamcenter sites ..................... 6-1

Replicating workflow templates ..................................................... 6-1
Replicate a workflow template ....................................................... 6-1
Synchronize replicated templates ....................................................... 6-2

Importing and exporting workflow templates using Workflow Designer .... 6-3
Import workflow templates .............................................................. 6-3
Export workflow templates ............................................................... 6-5
Chapter 6  Using workflow templates at multiple Teamcenter sites

Using workflow templates at multiple Teamcenter sites
There are two methods to distribute your workflow templates to different Teamcenter sites:

- Replicating templates using Multi-Site Collaboration
- Importing and exporting templates in an XML format

Replicating workflow templates
You can replicate your workflow templates, including those under construction, on several Teamcenter sites by using the data_share utility and update them with the data_sync utility. You cannot edit the replicas, only the template at the owning site. Also, handlers attached to the templates must exist at all sites where the templates are replicated.

Replicate a workflow template
1. If necessary, create the template you want to replicate.

   For more information, see Create templates in Workflow Designer.

2. Run the data_share utility with the following arguments:

   data_share -u=user-id -p=password -g=group -f=send
   -site=remote-site-name1 -name=workspace-object-class=class-name

For example, if you want to replicate the demotemplate workflow template at the teamcentersite2 site, run the following utility command (the required logon information is omitted from the example):

   data_share -f=send -site=teamcentersite2 -name=demotemplate
   -class=EPMTaskTemplate
Note

- If you want to transfer ownership to the specified site, add the -transfer argument to the command.

- If you want to import the template at another site to the current site, change the -f argument to -f=remote_import.

- If you want to replicate the template at more than one site, add more -site arguments to the command.

- If you want to replicate several templates, type the template names in a text file and replace the -name and -class arguments with the -filename and -classoffile arguments, respectively.

The replicate template appears at the new site with the ** symbol.

**Synchronize replicated templates**

1. Update the template at the owning site that is replicated at another site.

   For more information, see *Configure ability to apply template edits to active processes*.

   **Note**

   If you want active workflow processes based on the synchronized template to be updated at the replica site, set the WRKFLW_multisite_apply_template_changes preference to true.

   For more information, see the Preferences and Environment Variables Reference.

2. Run the data_sync utility with the following arguments:

   \[
   \text{data\_sync} \ -u=\text{user\_id} \ -p=\text{password} \ -g=\text{group} \ -f=\text{sync} \\
   \text{-site=remote-site-name1} \ -\text{class=class-name} \ -\text{update}
   \]

   For example, if you changed the demotemplate workflow template and wanted to update the replica at the teamcentersite2 site, run the following utility command (the required logon information is omitted from the example):

   \[
   \text{data\_sync} \ -f=\text{sync} \ -\text{site=teamcentersite2} \ -\text{class=EPMTaskTemplate} \ -\text{update}
   \]

   **Note**

   If you want to synchronize the template at more than one site, add more -site arguments to the command.

   The replicate template is updated at the specified sites.
Importing and exporting workflow templates using Workflow Designer

Importing and exporting workflow process and task templates from the Teamcenter database is useful for transferring workflow templates between different Teamcenter sites.

You can import workflow process and task templates into the Teamcenter database from an exported workflow template file. Importing templates is useful for transferring workflow templates between different Teamcenter sites. The templates must first be exported from a Teamcenter database into an export file, after which you can import the file into the Teamcenter database at another site.

You can export workflow process and task templates from the Teamcenter database in XML format, storing the templates in a single export file. After exporting the templates, you can import the file into the Teamcenter database at another site. You can also easily search the XML to determine handler and argument usage.

Best practice

If your enterprise encompasses more than one site, always make workflow template changes at the master site, and then propagate the changes by exporting the workflow template from the master site to other sites. If additional changes are required at a later date, again make the workflow template changes at the master site, export the workflow template from the master site, and then import it at all other sites.

This method ensures that the origin_uid value of each workflow template continues to match from site to site. If you export/import a workflow template between nonmaster sites, its origin_uid value eventually becomes mismatched between versions, resulting in the following error when you choose to overwrite during import:

The origin_uid’s of the importing template(s) do not match with the origin_uid’s of the existing template(s). The import of template(s) in overwrite mode failed. Matching origin_uid’s are required to apply template changes to active workflow processes. You can replace the existing template by deleting it, and then re-importing, but this will prevent you from applying template changes to active workflow processes.

If you receive this error, you can manually replace the existing template with the importing template by first deleting the importing template, then repeating the import. However, using this method breaks the link between origin_uid values. If you use this method, the system cannot apply template changes to active workflow processes, as described in Applying template edits to active workflow processes.

Import workflow templates

1. Choose Tools→Import.
   The system displays the Import Workflow Templates dialog box.

2. Type the path to the directory containing the export file in the Import File box, or click the Browse button to locate the directory.

3. (Optional) If you want the system to continue the transfer if one or more workflow templates fail to transfer, select the Continue On Error check box. If one or more workflow templates fail to transfer, the system records transfer errors in its log files, bypasses the failed workflow templates, and transfers the remaining workflow templates.
Chapter 6  Using workflow templates at multiple Teamcenter sites

If you do not select this option, the system stops the transfer process if one workflow template fails to transfer and only includes in the transfer those workflow templates that transferred successfully.

4. (Optional) If you want the system to overwrite any workflow template of the same name that already exists in the database, select the **Overwrite Duplicate Templates** check box. The system does not display or log any errors.

Select this option when the imported workflow template contains changes that you want applied to the database. For example, you have added two custom tasks to the **QuarterlyReview** workflow template and thoroughly tested the revised template in your test database. Now you are ready to import the changes to the production database. By choosing to overwrite duplicate templates when importing the workflow template to the production database, you are effectively editing the **QuarterlyReview** workflow template. On import, the original **QuarterlyReview** workflow template is overwritten by the importing workflow template; it now contains the two custom tasks.

If you do not select this option, any importing template with the same name as an existing template is ignored and the import process continues. A message is logged that a workflow template of the same name exists.

5. (Optional) If you chose to overwrite duplicate templates, you can also choose to apply the differences in the imported templates to all active workflow processes based on the original version of the workflow template. In other words, you can choose to apply the edits you have made to the importing template to active workflow processes.

To continue the example in the previous step, if you select the **Apply template changes to all active workflow processes** check box while importing the **QuarterlyReview** workflow template into the production database, the two custom tasks added during import are also applied to all active workflow processes that were based on the original version of the **QuarterlyReview** workflow template.

Updates are applied as described in **Applying template edits to active workflow processes**.

6. (Optional) If you chose to apply edits to active workflow processes, you can also choose to process the edits in the background by selecting the **Update processes in background** check box.

Your edits are applied in the background. The updates run asynchronously and you are notified by Teamcenter mail when the updates complete. Typically, you only want to update workflow processes in real time when your changes impact 10–20 active workflow processes, as in testing scenarios.

**Caution**

Asynchronous processing must be configured.

For more information about the required configuration procedures, see **Configuring background processing of processes and tasks**.

7. Click **OK** to import the templates contained within the file you selected into the Teamcenter database.
The imported template names now exist in the database and appear in the Process Template list.

Export workflow templates

   The Export Workflow Templates dialog box appears.

2. Type the path to the directory containing the objects you want to export in the Export Directory box, or click the Browse button to locate the directory.

3. Specify the name of the export file in the File Name box, for example, template_export.

4. In the Templates section of the dialog box, select the templates you want to export from the All Templates list. (Use the Ctrl key to select multiple templates.)

5. Add the selected templates to the Selected Templates list. These are the templates the system exports.

6. If you want the system to continue the transfer if one or more templates fail to transfer, select Continue On Error. If one or more templates fail to transfer, the system records transfer errors in its log files, bypasses the failed templates, and transfers the remaining templates.

   If you do not choose this option, the system stops the transfer process if one template fails to transfer and only includes in the transfer those templates that transferred successfully.

7. Click OK to export the templates in the Selected Templates list and close the dialog box.

   The selected templates are exported in XML format to the file name you defined in step 3 in the directory you defined in step 2.
# Appendix

## A Workflow handlers

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow handlers</td>
<td>A-1</td>
</tr>
<tr>
<td>Syntax for handler arguments</td>
<td>A-1</td>
</tr>
<tr>
<td>Handler keywords</td>
<td>A-1</td>
</tr>
<tr>
<td>Common keywords</td>
<td>A-2</td>
</tr>
<tr>
<td>Handler-specific keywords</td>
<td>A-3</td>
</tr>
<tr>
<td>Use keywords to implement dynamic participants in handlers</td>
<td>A-7</td>
</tr>
<tr>
<td>Using lists of values (LOVs) as handler arguments</td>
<td>A-8</td>
</tr>
<tr>
<td>LOV syntax</td>
<td>A-8</td>
</tr>
<tr>
<td>LOV syntax example</td>
<td>A-9</td>
</tr>
<tr>
<td>Differentiating between classes and types</td>
<td>A-10</td>
</tr>
<tr>
<td>Defining multilevel object paths</td>
<td>A-10</td>
</tr>
<tr>
<td>Specifying relations</td>
<td>A-11</td>
</tr>
<tr>
<td>Debugging handler data</td>
<td>A-13</td>
</tr>
<tr>
<td>Action handlers</td>
<td>A-13</td>
</tr>
<tr>
<td>add-status</td>
<td>A-14</td>
</tr>
<tr>
<td>adhoc-signoffs</td>
<td>A-15</td>
</tr>
<tr>
<td>AI-process-export</td>
<td>A-24</td>
</tr>
<tr>
<td>AI-process-import</td>
<td>A-25</td>
</tr>
<tr>
<td>APPR-update-from-targets</td>
<td>A-26</td>
</tr>
<tr>
<td>approve-service-structure</td>
<td>A-27</td>
</tr>
<tr>
<td>auto-assign</td>
<td>A-28</td>
</tr>
<tr>
<td>auto-assign-rest</td>
<td>A-32</td>
</tr>
<tr>
<td>auto-relocate-file</td>
<td>A-35</td>
</tr>
<tr>
<td>CAE-batch-meshing-handler</td>
<td>A-36</td>
</tr>
<tr>
<td>CAE-simulation-process-launch-handler</td>
<td>A-37</td>
</tr>
<tr>
<td>change-all-started-to-pending</td>
<td>A-39</td>
</tr>
<tr>
<td>CR-assign-team-selector</td>
<td>A-40</td>
</tr>
<tr>
<td>CR-change-group-owner</td>
<td>A-43</td>
</tr>
<tr>
<td>CR-change-target-group</td>
<td>A-44</td>
</tr>
<tr>
<td>CR-change-target-group-owner</td>
<td>A-45</td>
</tr>
<tr>
<td>CR-fill-in-reviewers</td>
<td>A-46</td>
</tr>
<tr>
<td>CR-notify</td>
<td>A-50</td>
</tr>
<tr>
<td>create-status</td>
<td>A-59</td>
</tr>
<tr>
<td>debug</td>
<td>A-60</td>
</tr>
<tr>
<td>demote</td>
<td>A-61</td>
</tr>
<tr>
<td>demote-on-reject</td>
<td>A-62</td>
</tr>
<tr>
<td>DOCMGT-render-document-revision</td>
<td>A-64</td>
</tr>
<tr>
<td>DPV-export-device-to-ai</td>
<td>A-65</td>
</tr>
<tr>
<td>DPV-export-plant-to-ai</td>
<td>A-66</td>
</tr>
<tr>
<td>DPV-export-routine-to-ai</td>
<td>A-67</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ECM-add-affected-irs-as-target</td>
<td>A-68</td>
</tr>
<tr>
<td>ECM-att-new-status-for-aff-revs</td>
<td>A-69</td>
</tr>
<tr>
<td>ECM-attach-components-to-change</td>
<td>A-70</td>
</tr>
<tr>
<td>ECM-copy-end-item-effectivity</td>
<td>A-71</td>
</tr>
<tr>
<td>ECM-create-base-revrule-form</td>
<td>A-72</td>
</tr>
<tr>
<td>ECM-notify-competing-changes</td>
<td>A-73</td>
</tr>
<tr>
<td>ECM-set-base-revrule</td>
<td>A-75</td>
</tr>
<tr>
<td>ECM-start-new-sub-processes</td>
<td>A-76</td>
</tr>
<tr>
<td>EPM-attach-assembly-components</td>
<td>A-77</td>
</tr>
<tr>
<td>EPM-attach-item-revision-targets</td>
<td>A-82</td>
</tr>
<tr>
<td>EPM-attach-related-objects</td>
<td>A-83</td>
</tr>
<tr>
<td>EPM-auto-check-in-out</td>
<td>A-91</td>
</tr>
<tr>
<td>EPM-change-ownership</td>
<td>A-92</td>
</tr>
<tr>
<td>EPM-check-signoff-comments</td>
<td>A-94</td>
</tr>
<tr>
<td>EPM-create-form</td>
<td>A-95</td>
</tr>
<tr>
<td>EPM-create-relation</td>
<td>A-97</td>
</tr>
<tr>
<td>EPM-create-sub-process</td>
<td>A-98</td>
</tr>
<tr>
<td>EPM-delete-ugcgm-markup</td>
<td>A-109</td>
</tr>
<tr>
<td>EPM-display-form</td>
<td>A-110</td>
</tr>
<tr>
<td>EPM-export-AI-AH</td>
<td>A-113</td>
</tr>
<tr>
<td>EPM-export-to-plmxmfile</td>
<td>A-114</td>
</tr>
<tr>
<td>EPM-generate-image</td>
<td>A-116</td>
</tr>
<tr>
<td>EPM-generate-ugcgm-drawing</td>
<td>A-117</td>
</tr>
<tr>
<td>EPM-make-mature-design-primary</td>
<td>A-118</td>
</tr>
<tr>
<td>EPM-mark-archive</td>
<td>A-119</td>
</tr>
<tr>
<td>EPM-perform-offline-export</td>
<td>A-121</td>
</tr>
<tr>
<td>EPM-publish-target-objects</td>
<td>A-123</td>
</tr>
<tr>
<td>EPM-remove-objects</td>
<td>A-124</td>
</tr>
<tr>
<td>EPM-run-external-command</td>
<td>A-127</td>
</tr>
<tr>
<td>EPM-send-target-objects</td>
<td>A-139</td>
</tr>
<tr>
<td>EPM-send-to-oramfg</td>
<td>A-141</td>
</tr>
<tr>
<td>EPM-set-condition-by-check-validation-result</td>
<td>A-142</td>
</tr>
<tr>
<td>EPM-set-form-value-AH</td>
<td>A-143</td>
</tr>
<tr>
<td>EPM-set-job-protection</td>
<td>A-144</td>
</tr>
<tr>
<td>EPM-set-property</td>
<td>A-145</td>
</tr>
<tr>
<td>EPM-set-rule-based-protection</td>
<td>A-154</td>
</tr>
<tr>
<td>EPM-set-task-result-to-property</td>
<td>A-155</td>
</tr>
<tr>
<td>EPM-tessellation-handler</td>
<td>A-157</td>
</tr>
<tr>
<td>EPM-unpublish-target-objects</td>
<td>A-159</td>
</tr>
<tr>
<td>ERP-att-logfile-as-dataset-RH</td>
<td>A-160</td>
</tr>
<tr>
<td>ERP-attach-targets-AH</td>
<td>A-161</td>
</tr>
<tr>
<td>ERP-delete-log-dataset-AH</td>
<td>A-162</td>
</tr>
<tr>
<td>ERP-download-AH</td>
<td>A-163</td>
</tr>
<tr>
<td>ERP-post-upload-AH</td>
<td>A-164</td>
</tr>
<tr>
<td>ERP-set-pathnames-in-logds-AH</td>
<td>A-165</td>
</tr>
<tr>
<td>ERP-transform-AI-contents-AH</td>
<td>A-166</td>
</tr>
<tr>
<td>execute-follow-up</td>
<td>A-167</td>
</tr>
<tr>
<td>inherit</td>
<td>A-169</td>
</tr>
<tr>
<td>invoke-system-action</td>
<td>A-170</td>
</tr>
<tr>
<td>ISSUEMGIT-check-review-decision</td>
<td>A-176</td>
</tr>
<tr>
<td>ISSUEMGIT-update-issue-status</td>
<td>A-177</td>
</tr>
<tr>
<td>late-notification</td>
<td>A-179</td>
</tr>
<tr>
<td>OBJJIO-release-and-replicate</td>
<td>A-182</td>
</tr>
<tr>
<td>notify</td>
<td>A-184</td>
</tr>
</tbody>
</table>
notify-signoffs .................................................. A-191
release-asbuilt-structure ...................................... A-192
release-asmaintained-structure ............................... A-193
require-authentication ........................................ A-194
RM-attach-SM-tracelink-requirement ......................... A-195
RM-attach-tracelink-requirement ............................ A-196
SAP-set-valid-date-AH ........................................ A-197
SAP-upload-AH ................................................ A-198
schmgt-approve-timesheetentries ............................ A-199
schmgt-revise-timesheetentries .............................. A-200
set-condition .................................................. A-201
set-duration ................................................... A-203
set-parent-result ............................................. A-204
set-status ...................................................... A-205
suspend-on-reject ............................................. A-208
system ......................................................... A-209
TCX-auto-approve-first-step ................................ A-210
TCX-create-form ............................................... A-211
TCX-Create-Print-Requests .................................. A-212
TCX-create-snapshot .......................................... A-213
TCX-Create-Translation-Request ............................. A-214
TCX-delete-dataset ............................................ A-216
TCX-delete-log-datasets ...................................... A-217
TCX-export-signoff-data ...................................... A-218
TCX-IRM-cleanfields ......................................... A-220
TCX-purge-dataset ............................................ A-221
TCX-release-previous-itemrevs ............................... A-222
TCX-remove-targets-with-status .............................. A-223
TCX-set-bom-precise ......................................... A-224
TCX-setstatus-EO-folder ..................................... A-225
TCX-store-cr-data .............................................. A-226
TCX-trigger-approve-first-step .............................. A-228
trigger-action ................................................. A-229
trigger-action-on-related-process-task ..................... A-230
TSTK-CreateTranslationRequest .............................. A-234
VAL-approve-result-overrides ................................ A-234
VAL-reject-result-overrides ................................ A-236
VAL-set-condition-result-overrides ......................... A-237

Legacy syntax for selected action handlers .................. A-237
adhoc-signoffs (legacy) ....................................... A-238
auto-assign (legacy) .......................................... A-241
auto-assign-rest (legacy) ..................................... A-243
CR-assign-team-selector (legacy) ............................ A-245
CR-fill-in-reviewers (legacy) ................................. A-247
CR-notify (legacy) ............................................. A-251
notify (legacy) ................................................ A-257

Rule handlers .................................................. A-262
assert-signoffs-target-read-access ......................... A-263
check-condition .............................................. A-264
check-process-completion .................................... A-266
check-responsible-party .................................... A-267
check-signoff ............................................... A-268
CR-assert-targets-checked-in ........................................ A-269
CR-check-item-status .................................................. A-270
debug-rule ................................................................. A-271
disallow-adding-targets .............................................. A-272
disallow-removing-targets .......................................... A-273
ECM-check-target-is-ec ............................................. A-274
ECM-is-all-affected-irs-released ................................ A-275
ECM-is-all-affected-irs-target ..................................... A-276
ECM-is-valid-ec-process ............................................ A-277
EPM-assert-target-classified ....................................... A-278
EPM-check-action-performer-role ................................ A-279
EPM-check-assembly-status-progression ...................... A-281
EPM-check-object-properties ....................................... A-284
EPM-check-occ-notes .................................................. A-287
EPM-check-oramfg-transfer-status ............................. A-288
EPM-check-related-objects ......................................... A-289
EPM-check-status-progression ................................... A-293
EPM-check-target-attachments ................................... A-296
EPM-check-target-object ........................................... A-301
EPM-check-validation-result ...................................... A-303
EPM-check-validation-result-with-rules ..................... A-304
EPM-hold ................................................................. A-306
EPM-validate-target-objects ....................................... A-308
ERP-check-effective-date-RH ..................................... A-310
ERP-check-target-status-RH ....................................... A-311
ERP-validate-data-RH ................................................ A-312
invoke-system-rule .................................................... A-313
SAP-check-forms-attached-RH ................................ A-319
SAP-check-forms-to-download-RH ............................. A-321
TCX-check-approver ................................................ A-322
TCX-check-bom-precise ............................................. A-323
TCX-check-bomchild-statualist ................................. A-324
TCX-check-comps-against-pattern ............................... A-325
TCX-check-datasets ................................................. A-326
TCX-check-itemrev-status .......................................... A-328
TCX-check-jobowner ................................................ A-329
TCX-check-prev-itemrev-status ................................ A-330
TCX-check-signoff .................................................... A-331
TCX-check-status ..................................................... A-332
TCX-has-target-drawing ............................................ A-333
validate-for-checkedout-asmaintained-physicalpartrevision A-334
validate-for-checkedout-physicalpartrevision ............... A-335
validate-for-class ..................................................... A-336
validate-for-latest-asmparticalongrevision ................. A-337
validate-for-physicalpartrevision ............................... A-338
validate-for-unserviceable-physicalpartrevision .......... A-339
validate-missing-asmaintained-structure .................. A-340
validate-missing-structure ........................................ A-341

Handler examples ..................................................... A-341
Replace status of target objects workflow example ........ A-341
Start node ............................................................... A-342
ACMERP (Status task) ............................................... A-344
Appendix

A  Workflow handlers

Workflow handlers

Handlers are the lowest-level building blocks in workflow. They are small ITK programs used to extend and customize tasks. There are two kinds of handlers:

• Action handlers perform an action, such as attaching objects or sending an e-mail.

• Rule handlers confirm that a defined rule has been satisfied. If the rule is met, the handler returns the EPM_go command, allowing the task to continue. If the rule is not met, it returns the EPM_nogo command, preventing the task from continuing.

For more information and a list of available handlers, see Workflow handlers.

Syntax for handler arguments

Define handler arguments and values using the Handlers dialog box.

When you select a handler name, the existing arguments and values for the selected handler populate the argument table. You can enter additional arguments by typing argument and value data into the table cells. To assign multiple values to a single argument, separate the values with commas. For example:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-type</td>
<td>UGMASTER, UGPART</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

Handler keywords

Keywords are special arguments that extract values from the system, inserting the data into the handler's argument values in place of the keyword. Keyword syntax is the dollar sign ($) followed by the keyword name. For example, $USER extracts the logon ID of the current user and inserts that value into the handler argument.

Some keywords are common keywords. You can use common keywords with many Teamcenter handlers. You can use some common keywords with custom handlers by using the EPM_substitute_keyword and EPM_substitute_task_keyword ITK functions. Use of these functions is illustrated within some of the sample workflow handlers delivered in the sample directory.
Other keywords are **handler-specific keywords**. You can handler-specific keywords only with specific handlers. The documentation for each handler lists any handler-specific keywords that you can use with that handler.

### Common keywords

Table The following table lists common keywords that you can use with many Teamcenter handlers and with custom handlers by using the **EPM_substitute_keyword** ITK function.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$USER</td>
<td>Extracts the user ID of the current user.</td>
</tr>
<tr>
<td>$GROUP</td>
<td>Extracts the group ID of the current user.</td>
</tr>
<tr>
<td>$ROLE</td>
<td>Extracts the role of the current user.</td>
</tr>
</tbody>
</table>

The following table lists common keywords that you can use with many Teamcenter handlers and with custom handlers by using the **EPM_substitute_task_keyword** ITK function.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PROCESSOWNER</td>
<td>Extracts the user ID of the owner of the current workflow process.</td>
</tr>
<tr>
<td>$PROCESSGROUP</td>
<td>Extracts the group ID of the owner of the current workflow process.</td>
</tr>
</tbody>
</table>
| $TARGETOWNER[[(Class) | Extracts the user ID of the owner of the current workflow process's target.  
| Type]]        | You can define an optional type or bracketed class in square brackets to specify the type or class of target object from which to extract the owner ID. If you do not define a class or type, the system uses the class of **ItemRevision** by default. |
|               | If the system finds more than one object, it returns the owner ID from the first object. |
|               | For example, $TARGETOWNER[(Dataset)] extracts the owning user ID from the first dataset target found, and $TARGETOWNER[UGMASTER] extracts the owning user ID from the first **UGMASTER** target found. |
| $TARGETGROUP[[(Class) | Extracts the group ID of the owner of the current workflow process's target. Only the first owner is returned.  
| Type]]        | As with $TARGETOWNER, you can provide a type or bracketed class in square brackets to specify the type or class of target object from which to extract the owning group ID. |
### Workflow handlers

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TARGETOWNERS[[(Class)</td>
<td>Type1[,Type2,...]]]</td>
</tr>
<tr>
<td>$TARGETGROUPS[[(Class)</td>
<td>Type1[,Type2,...]]]</td>
</tr>
<tr>
<td>$ROLEINGROUP</td>
<td>Extracts the user’s current logged-on group ID and role in the format of a resource string, for example, group::role.</td>
</tr>
</tbody>
</table>

**Handler-specific keywords**

The following table lists keywords that you can only use with specific handlers.

The documentation for each action handler and rule handler lists any handler-specific keywords that you can use with that handler. You can search the handler documentation for a particular handler-specific keyword to find all handlers that accept that keyword and to read a description of its functionality.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CHANGE_IMPLEMENTATION_BOARD</td>
<td>adhoc-signoffs, CR-fill-in-reviewers, CR-notify, notify</td>
</tr>
</tbody>
</table>
### Workflow handlers

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$CHANGE_REVIEW_BOARD</code></td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td><code>$CHANGE_SPECIALIST1</code></td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>auto-assign</td>
</tr>
<tr>
<td></td>
<td>auto-assign-rest</td>
</tr>
<tr>
<td></td>
<td>CR-assign-team-selector</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td><code>$CHANGE_SPECIALIST2</code></td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>auto-assign</td>
</tr>
<tr>
<td></td>
<td>auto-assign-rest</td>
</tr>
<tr>
<td></td>
<td>CR-assign-team-selector</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td><code>$CHANGE_SPECIALIST3</code></td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>auto-assign</td>
</tr>
<tr>
<td></td>
<td>auto-assign-rest</td>
</tr>
<tr>
<td></td>
<td>CR-assign-team-selector</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td><code>$COMPETING_EC_OWNER</code></td>
<td>ECM-notify-competing-changes</td>
</tr>
<tr>
<td><code>$COMPETING_REV_OWNER</code></td>
<td>ECM-notify-competing-changes</td>
</tr>
<tr>
<td><code>$CURRENT_DATE</code></td>
<td>EPM-set-property</td>
</tr>
<tr>
<td><code>$EC.Owner</code></td>
<td>ECM-notify-competing-changes</td>
</tr>
<tr>
<td><code>$OWNER</code></td>
<td>EPM-check-action-performer-role</td>
</tr>
<tr>
<td></td>
<td>late-notification</td>
</tr>
</tbody>
</table>
## Workflow handlers

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PROCESS</td>
<td>check-process-completion</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td></td>
<td>notify-signoffs</td>
</tr>
<tr>
<td></td>
<td>notify (legacy) (legacy syntax)</td>
</tr>
<tr>
<td>$PROJECT_ADMINISTRATOR</td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>auto-assign</td>
</tr>
<tr>
<td></td>
<td>auto-assign-rest</td>
</tr>
<tr>
<td></td>
<td>CR-assign-team-selector</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td>$PROJECT_AUTHOR</td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td>$PROJECT_MEMBER</td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
<tr>
<td>$PROJECT_TEAM_ADMINISTRATOR</td>
<td>adhoc-signoffs</td>
</tr>
<tr>
<td></td>
<td>auto-assign</td>
</tr>
<tr>
<td></td>
<td>auto-assign-rest</td>
</tr>
<tr>
<td></td>
<td>CR-assign-team-selector</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers</td>
</tr>
<tr>
<td></td>
<td>CR-notify</td>
</tr>
<tr>
<td></td>
<td>notify</td>
</tr>
</tbody>
</table>
### Appendix A  
**Workflow handlers**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Handlers</th>
</tr>
</thead>
</table>
| $\texttt{PROPOSED\_RESPONSIBLE\_PARTY}$ | adhoc-signoffs  
|                                  | auto-assign  
|                                  | auto-assign-rest  
|                                  | CR-assign-team-selector  
|                                  | CR-fill-in-reviewers  
|                                  | CR-notify  
|                                  | notify  |
| $\texttt{PROPOSED\_REVIEWERS}$   | adhoc-signoffs  
|                                  | CR-fill-in-reviewers  
|                                  | CR-notify  
|                                  | notify  |
| $\texttt{REFERENCE}$             | check-process-completion  
|                                  | EPM-attach-related-objects  
|                                  | EPM-create-form  
|                                  | EPM-create-relation  
|                                  | EPM-display-form  
|                                  | EPM-remove-objects  
|                                  | EPM-set-property  
|                                  | notify  
|                                  | notify-signoffs  
|                                  | notify (legacy) (legacy syntax)  |
| $\texttt{RELEASE\_STATUS}$       | EPM-create-form  
|                                  | EPM-create-relation  
|                                  | EPM-display-form  |
| $\texttt{RESPONSIBLE\_PARTY}$    | CR-notify  
|                                  | EPM-check-action-performer-role  
|                                  | late-notification  
<p>|                                  | notify  |</p>
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$REQUESTOR</td>
<td>adhoc-signoffs, auto-assign, auto-assign-rest, CR-assign-team-selector,</td>
</tr>
<tr>
<td></td>
<td>CR-fill-in-reviewers, CR-notify, notify</td>
</tr>
<tr>
<td>$REV_OWNER</td>
<td>ECM-notify-competing-changes</td>
</tr>
<tr>
<td>$REV REVIEWERS</td>
<td>CR-fill-in-reviewers, CR-notify, late-notification, notify</td>
</tr>
<tr>
<td>$SIGNOFF</td>
<td>EPM-create-form, EPM-create-relation, EPM-display-form</td>
</tr>
<tr>
<td>$TARGET</td>
<td>EPM-attach-related-objects, EPM-check-target-attachments, EPM-create-form,</td>
</tr>
<tr>
<td></td>
<td>EPM-create-relation, EPM-display-form, EPM-remove-objects, EPM-set-property,</td>
</tr>
<tr>
<td></td>
<td>notify, notify-signoffs, notify (legacy) (legacy syntax)</td>
</tr>
<tr>
<td>$TEMPLATE</td>
<td>check-process-completion</td>
</tr>
<tr>
<td>$UNDECIDED</td>
<td>CR-notify, late-notification</td>
</tr>
</tbody>
</table>

Use keywords to implement dynamic participants in handlers

You can use the following keywords to invoke dynamic participants:
Workflow handlers

$ANALYST  $PROJECT_ADMINISTRATOR
$CHANGE_SPECIALIST1  $PROJECT_TEAM_ADMINISTRATOR
$CHANGE_SPECIALIST2  $PROJECT_AUTHOR
$CHANGE_SPECIALIST3  $PROJECT_MEMBER
$CHANGE_REVIEW_BOARD  $REQUESTOR
$CHANGE_IMPLEMENTATION_BOARD

For more information about which handlers use these keywords, see Handler-specific keywords.

If you want to use your custom dynamic participants, follow these steps:

1. In Business Modeler IDE, create a child of the Participant business object.
2. For each child you create, associate a keyword in Business Modeler IDE.
3. In Workflow Designer, use the keyword you associated with a Participant business object child in a handler.

   The handler associates the keyword with the dynamic participant defined in Business Modeler IDE and users with the specified role.

For more information about creating Participant business object children and associating keywords, see the Business Modeler IDE Guide.

Using lists of values (LOVs) as handler arguments

Some handlers have the ability to work on many objects, or may require many pieces of information to fully define what it is required of them. In these cases, it is cumbersome to supply all of the information as arguments or to add the handler several times to the same task, defining multiple arguments each time.

In cases when a handler is placed several times in a workflow process on different tasks (or in different workflow processes), adding many arguments to each instance of the handler is time consuming. If arguments later need to be modified, they may need to be changed in every instance of the handler, which is also time consuming.

Using LOVs as handler arguments is an efficient alternative. Standard LOVs supply a list of possible values to form attributes. LOVs used in handler arguments are created in the same way, using the standard LOV interface; however they do not need to be attached to any attributes. Each line in the LOV supplies configuration information relevant to the specific handler it is used for and in the format required by the handler. For more information about creating LOVs, see List of values help.

LOV syntax

The format of the data in a handler LOV is dependent on the information required by the handler, therefore, it is not the same across all handlers that accept LOV arguments. Where similar types of information are required, however, a consistent format is used. For example, when multiple fields of information are required in an LOV line, the fields are separated by tildes (~). The individual handler documentation describes the LOV line format required for that handler.
Any handler using an LOV accepts the `-lov=lov-name` argument, which specifies the LOV to be used. LOVs cannot have duplicate lines, and there is a line length limit of 240 characters. Use these formatting commands to manage LOV syntax:

- A backslash (\) continuation character can be used at the end of a line, allowing very long lines to be broken into more manageable lines.

- Insert an optional line number within square brackets ([I]) to avoid creating duplicate lines when you need to enter duplicate data on multiple lines. Using line numbers can also help with sorting lines in the LOV, if the order of the lines in the LOV is important for a specific handler.

- Any content contained within square brackets ([I]) at the start of a line is ignored, as are spaces after the closing square brackets.

- A line starting with a number sign (#) character is treated as a comment and ignored.

- Any blank lines are ignored.

**Note**

The name of an LOV used with a handler can be anything, but using a naming convention, for example, `SYS_handler-name`, can help in identifying LOVs used by handlers in the LOV dialog box.

**LOV syntax example**

This LOV example is copied from the EPM-attach-related-objects handler documentation. Notice how blank lines and the number sign (#) are used to create comment lines, which describe the behavior of the formatted lines.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-lov</code></td>
<td><code>SYS_EPM_attach_objects</code></td>
</tr>
</tbody>
</table>

The `SYS_EPM_attach_objects` LOV contains this data:

```
#========================================================
# LOV: SYS_EPM_attach_objects
# Used with EPM-attach-related-objects
#========================================================
# Attach all objects in target revision Specification relation
#========================================================
$TARGET.(ItemRevision).Specification.*

#========================================================
# Attach all forms attached to datasets in target revision
# Specification relation
#========================================================
$TARGET.(ItemRevision).Specification.(Dataset).Form.(Form)!UGPartAttr

#========================================================
# Attach all BOM View Revisions in target revision
#========================================================
$TARGET.(ItemRevision).PSBOMViewRevision.*

#========================================================
# Attach all forms in target revision Manifestation relation
```

Any handler using an LOV accepts the `-lov=lov-name` argument, which specifies the LOV to be used. LOVs cannot have duplicate lines, and there is a line length limit of 240 characters. Use these formatting commands to manage LOV syntax:

- A backslash (\) continuation character can be used at the end of a line, allowing very long lines to be broken into more manageable lines.

- Insert an optional line number within square brackets ([I]) to avoid creating duplicate lines when you need to enter duplicate data on multiple lines. Using line numbers can also help with sorting lines in the LOV, if the order of the lines in the LOV is important for a specific handler.

- Any content contained within square brackets ([I]) at the start of a line is ignored, as are spaces after the closing square brackets.

- A line starting with a number sign (#) character is treated as a comment and ignored.

- Any blank lines are ignored.

**Note**

The name of an LOV used with a handler can be anything, but using a naming convention, for example, `SYS_handler-name`, can help in identifying LOVs used by handlers in the LOV dialog box.

**LOV syntax example**

This LOV example is copied from the EPM-attach-related-objects handler documentation. Notice how blank lines and the number sign (#) are used to create comment lines, which describe the behavior of the formatted lines.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-lov</code></td>
<td><code>SYS_EPM_attach_objects</code></td>
</tr>
</tbody>
</table>

The `SYS_EPM_attach_objects` LOV contains this data:

```
#========================================================
# LOV: SYS_EPM_attach_objects
# Used with EPM-attach-related-objects
#========================================================
# Attach all objects in target revision Specification relation
#========================================================
$TARGET.(ItemRevision).Specification.*

#========================================================
# Attach all forms attached to datasets in target revision
# Specification relation
#========================================================
$TARGET.(ItemRevision).Specification.(Dataset).Form.(Form)!UGPartAttr

#========================================================
# Attach all BOM View Revisions in target revision
#========================================================
$TARGET.(ItemRevision).PSBOMViewRevision.*

#========================================================
# Attach all forms in target revision Manifestation relation
```
Appendix A  Workflow handlers

Differentiating between classes and types

The purpose of many handlers is to locate and/or act on specified types or classes. Specifying a type directs the system to identify an object type. But specifying a class directs the system to identify any of the many types within that class. Therefore, it can be difficult to distinguish between types and classes.

For example, in the case of item revisions, some handlers perceive ItemRevision as a class of item revisions, making it difficult to designate the ItemRevision type.

Some handlers have the ability to distinguish between a class and type definitively. These handlers accept syntax that uses round brackets () to specify a class. For example, (ItemRevision) specifies the class and ItemRevision specifies the type. When this bracket notation is accepted, an exclamation point (!) can be used to exclude specific types, using this format:

(Class)[!Type1[!Type2[...]]

For example, given the four item types defined:

- Item
- Document
- Design
- Software

then:

(Item) Matches any object of the Item class.
(Item) ! Software Matches any object of the Item class except for the type Software.
(Item) ! Document ! Item Matches any object of class Item except for the Document and Item types.

Design Matches only the Design type.

The individual handler documentation indicates which handlers accept this syntax.

Defining multilevel object paths

With some handlers, you can specify a multilevel path for locating objects using relation type/object type pairs, or relation type/class pairs. Typically, you use this method when working with LOVs.

The general syntax is:

relation.(type[!,type])[(class)[!type]].relation.(type[!,type])[(class)[!type]]

You specify multiple types in a comma-separated list. For any relation or type field in the path, you can use either an asterisk (*) or ALL as a wildcard to mean any relation, type, or class.

You can specify target and reference relations within a workflow process using the $TARGET and $REFERENCE keywords.
Workflow handlers

For example, use multilevel object paths to find forms of a specific type attached to revisions within revisions. Consider this scenario:

A change item revision is currently in a change process. The change object contains item revisions with the Solution Items relation. Each of these solution revisions contain an Affected Item Form type in a reference relation that needs to be attached to the change process. You can identify these forms using this syntax:

\$TARGET.(ItemRevision).Solution Items.(ItemRevision).
  Reference.Affected Item Form

The previous example uses three relation pairs, as follows:

<table>
<thead>
<tr>
<th>Pair</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$TARGET.(ItemRevision)</td>
<td>Finds objects of the class ItemRevision attached as workflow process targets.</td>
</tr>
<tr>
<td>Solution Items.(ItemRevision)</td>
<td>For each of the revisions found by the first pair, the system searches the Solution Items relation to find objects of the class ItemRevision.</td>
</tr>
<tr>
<td>Reference.Affected Item Form</td>
<td>For each of the revisions found by the second pair, the system searches the Reference relations to find objects of the type Affected Item Form.</td>
</tr>
</tbody>
</table>

The individual handler documentation indicates which handlers accept this syntax.

Specifying relations

Some relations for certain objects cannot be specified with standard generic relationship management (GRM) relation types. For example, you cannot specify to select all the revisions in an item. The following table lists available types of relations, including GRM relations and special relations.

<table>
<thead>
<tr>
<th>Class</th>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>Any GRM relation</td>
<td>Identifies any GRM-related objects attached to items. For example: (Item).IMAN_reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, to find all the datasets in the IMAN_specification relation of all revisions in any items found: (Item).Revisions.*.IMAN_specification. (Dataset)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note The type of revision is not relevant as there is only one type of revision</td>
</tr>
</tbody>
</table>


### Attachment A

**Workflow handlers**

<table>
<thead>
<tr>
<th>Class</th>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSBOMView or BV</td>
<td>Any GRM relation</td>
<td>Identifies any GRM-related objects in any item; therefore, an asterisk (*) is used to specify any type.</td>
</tr>
<tr>
<td>Revision</td>
<td>Any GRM relation</td>
<td>Identifies all BOM views from items.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, to select all BOM views:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Item). PSBOMView</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Select only the view BOM views:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Item).BV.BOMView Revision</td>
</tr>
<tr>
<td>PSBOMViewRevision or BVR</td>
<td>Any GRM relation</td>
<td>Identifies all BOM view revisions from revisions.</td>
</tr>
<tr>
<td>Dataset</td>
<td>Any GRM relation</td>
<td>Identifies any GRM-related objects attached to revisions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(ItemRevision).IMAN_reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identifies any GRM-related objects in document revisions that are attached as requirements to design revisions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Design Revision.IMAN_requirement.Document Revision.IMAN_specification.*</td>
</tr>
<tr>
<td>Folder</td>
<td>*</td>
<td>Identifies objects in folders.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, to identify all revisions in a folder:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Folder).*.(ItemRevision)</td>
</tr>
<tr>
<td>Job</td>
<td>$TARGET or Targets</td>
<td>Identifies targets attached to a job.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Job).$TARGET</td>
</tr>
</tbody>
</table>
Workflow handlers

<table>
<thead>
<tr>
<th>Class</th>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$REFERENCE or</td>
<td>References</td>
<td>Identifies targets attached to a job.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Job).$REFERENCE</td>
</tr>
</tbody>
</table>

### Debugging handler data

The following handlers offer debugging functionality, enabled through the TC_HANDLERS_DEBUG environment variable:

- **EPM-check-target-object**
- **EPM-validate-target-objects**
- **EPM-check-target-attachments**
- **EPM-attach-related-objects**
- **EPM-remove-objects**

The debugging data displays in the system log file. Use the debugging information to solve small usability issues, such as incorrect argument usage. You can also submit the data in incident reports to customer service.

You can enable debugging functionality for all the above handlers and their subfunctions by setting the TC_HANDLERS_DEBUG environment variable to **ALL**.

Alternatively, you can enable debugging functionality for specific handlers by entering one or more of the above handler names as the value.

For information about this environment variable, see the Preferences and Environment Variables Reference.

### Action handlers

Action handlers extend and customize task actions. They perform such actions as displaying information, retrieving the results of previous tasks (inherit), notifying users, setting object protections and launching applications.

This section provides information about each action handler.
add-status

DESCRIPTION
Finds all status types attached to the root task and attaches them to each target object. The main objective of a release process is for this handler to be executed, thereby attaching a status to the workflow process. Target objects are officially released after this handler executes. For information about creating the status object, see create-status.

SYNTAX
add-status [RETAIN_RELEASE_DATE] [SET_EFFECTIVITY]

ARGUMENTS

RETAIN_RELEASE_DATE
Retains the original release date of the target object if previously released.

SET_EFFECTIVITY
System creates the open-ended effectivity with release date as start date.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
Use this handler with the create-status action handler. Assumes all target objects, reference objects, and status types are attached to the root task.
**adhoc-signoffs**

**DESCRIPTION**

Determine the behavior of the **Ad-hoc done** check box that displays within the **select-signoff-team** task’s interface, allowing the initializing user, address list members, and resource pool members to add users to the signoff team in an ad hoc manner. If the task template contains predefined signoff profiles, the ad hoc selections add one-time-only additions to the required signoff team. Alternatively, if the task template contains no predefined signoff profiles, the ad hoc additions comprise the whole of the signoff team.

When this handler is attached to the **select-signoff-team** task, the check box is not selected by default. You can modify this behavior using the **AUTO_COMPLETE** argument.

Note

When this handler is not attached to the **select-signoff-team** task, the check box displays by default as checked, in expectation that ad hoc additions are not required. Users can still clear the check box, add additional signoff team members to the signoff team, and then select the check box again.

Remember that the check box only indicates that the user has completed any ad hoc additions to the signoff team; it does not signify that the required profiles have been added to the signoff team. Even if the user fits into any of the signoff profiles, it is added only as an ad hoc user and not as the signoff profile member.

Using the **AUTO_COMPLETE** argument with this handler allows the **select-signoff-team** task to complete automatically. If the system’s **ad hoc done** query is returned as **true** and any predefined signoff profiles have been selected, the task automatically completes without user interaction. Therefore, the **select-signoff-team** task template can be configured to automatically choose a signoff team and decide whether or not to allow users to modify this predefined signoff team at execution of the task.

This handler’s arguments are listed in order of precedence, meaning that the system attempts to find a match for the argument as a user before it tries to find a match as an address list, and so on. When a **select-signoff-team** task is created, based on a task template that uses this handler, it parses these arguments and add those signoffs to the task. After that point, the ad hoc signoff functionality allows subsequent modifications to the signoff list. Therefore, what is specified in this handler is only used to initialize this task; during execution of the workflow process, the ad hoc signoff functionality accepts further changes.

By default, this handler is executed at workflow process initiation, rather than at the task where it is assigned. It initializes the signoff lists at workflow process initiation, allowing the workflow process initiator to view signoff assignments early in the workflow process and set the assignments as desired. However, this also means that assignments are based on target/assignment data as it exists at the time of initiation. For instance, if you use the **$TARGETGROUP** keyword argument with this handler and the handler is executed at workflow process initiation, it looks at the group that owns the targets when the workflow process is initiated, not
when the task using this handler is executed. When you use this method, keyword arguments always resolve to the workflow process initiator.

Use the -ce argument to ensure the handler is executed when the select-signoff-team task starts, rather than at workflow process initiation.

**SYNTAX**

**adhoc-signoffs**

[AUTO_COMPLETE]

[-assignee= [user:user | person:person | addresslist:list]
| resourcepool:group:role | allmembers:group:role
| $PROPOSED_RESPONSIBLE_PARTY | $PROPOSED_REVIEWERS | $USER
| $PROCESSOWNER | $TARGETOWNER[type]
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $PROJECT_AUTHOR | $PROJECT_MEMBER
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1 | $CHANGE_SPECIALIST2 | $CHANGE_SPECIALIST3
| $CHANGE_REVIEW_BOARD | $CHANGE_IMPLEMENTATION_BOARD]

[-quorum= quorum-value]

[-ce] [-clear_signoffs]

**ARGUMENTS**

**AUTO_COMPLETE** (optional)

(Optional.) Allows the task to complete without user interaction. Automatically selects the Ad-hoc done check box in the select-signoff-team task interface. The task is assumed to be populated; no select-signoff-team task needs to be performed through the interface (providing at least one of the signoff profiles have been fulfilled).

When this argument is not used, the system does not automatically select the Ad-hoc done check box, preventing the select-signoff-team task from completing until the user manually checks it, typically after ad hoc signoffs have been added. Absence of the adhoc-signoffs handler implies the presence of this argument, and the Ad-hoc done check box is selected and behaves accordingly.

**-assignee**

(Optional.) Assigns signoff members to select-signoff-team or Notify task under a Route task. It can take more than one value if you specified them using a comma-separated list. The following value formats are allowed:

- **user:user**
  
  Adds the user specified to the signoff member list for the task to which it is attached. Accepts a valid Teamcenter user ID.

- **person:person**
  
  Adds the user whose name is specified to the signoff member list for the task to which it is attached. Accepts a valid Teamcenter person name.

**Note**

If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use wayne, joan:

-assignee=person:wayne, joan
• **addresslist**: Adds all members of the address list specified to the signoff member list.

• **resourcepool**: Results in a single assignment which can be performed by any single member of this group/role.

  You can define resource pools in the form of `group::, group::role, or role`. Accepts valid Teamcenter resource pool names and these keywords:
  
  o **$GROUP**
    Current user’s current group.

  o **$ROLE**
    Current user’s current role.

  o **$TARGETGROUP[type]**
    Owning group of the first target object of the specified type. The `type` value is optional. If not specified, the first target is used.

  o **$PROCESSGROUP**
    Owning group of the workflow process.

• **allmembers**: Adds all members of a group/role combination to the signoff member list. You can define role in groups in the form of `group::, group::role, or role`. Accepts valid Teamcenter resource pool names and these keywords:

  o **$GROUP**
    Current user’s current group.

  o **$ROLE**
    Current user’s current role.

  o **$TARGETGROUP[type]**
    Owning group of the first target object of the specified type. The `type` value is optional. If not specified, the first target is used.

  o **$PROCESSGROUP**
    Owning group of the workflow process.

• **$PROPOSED_RESPONSIBLE_PARTY**
  Affects assignments based on the user assigned as the responsible party for the first target object.

• **$PROPOSED_REVIEWERS**
  Affects assignments based on members assigned as reviewers for the first target object.
Appendix A  Workflow handlers

- **$USER**
  Adds the current user to the signoff member list.

- **$PROCESSOWNER**
  Adds the workflow process owner to the signoff member list.

- **$TARGETOWNER [type]**
  Adds the owner of the first target of specified type to the signoff member list. The type value is optional. If not specified, the first target is used.

- **$PROJECT_ADMINISTRATOR, $PROJECT_TEAM_ADMINISTRATOR, $PROJECT_AUTHOR, $PROJECT_MEMBER**
  Dynamically adds the project team members belonging to the role specified in the argument value. The project team is determined by the project team associated with the first target object.

- **$REQUESTOR, $ANALYST, $CHANGE_SPECIALIST1, $CHANGE_SPECIALIST2, $CHANGE_SPECIALIST3 $CHANGE_REVIEW_BOARD, $CHANGE_IMPLEMENTATION_BOARD**
  Dynamically resolves to the user or resource pool associated with the first Change target object in the workflow process. The particular user or resource pool is determined by the role specified in the argument value.

**Note**

Change-related keywords apply only to change objects. If the workflow process does not contain a change object as a target, the argument resolves to null.

Change Manager does not need to be enabled before these keywords take effect, but during installation, Change Management must be selected under Extensions→Enterprise Knowledge Foundation in Teamcenter Environment Manager.

- **-quorum** (Optional.) Determines the quorum for the perform-signoffs task. The value can either be a percentage or a number. For example, if it is set to 51% then of all the signoff members, 51% of members need to approve for the task to move ahead. If it is set to 5, then 5 members need to approve for the task to move ahead. The value specified here will override the quorum specified on the select-signoff-team task template. If no value is specified, the quorum specified on the select-signoff-team task template is used. This argument is ignored if the handler is placed on a Notify task.

- **-ce** or **-conventional-execution** (Optional.) Disables the handler from being executed when the workflow process is initiated. Instead, the handler is executed in the conventional manner at the point of handler placement on the task action.

- **clear_signoffs** (Optional.) If specified, all existing signoffs are removed from the select-signoff-team subtask before the new signoffs are added.
Workflow handlers

PLACEMENT

Place on the Start action of a select-signoff-team subtask.

This handler is executed at workflow process initiation if the -ce argument is not specified. If -ce is specified, the handler is executed in a conventional manner at the point of handler placement on the task action.

Place on the Undo action of a select-signoff-team subtask and specify the -ce argument to clear the Ad-hoc done check box when the subtask is demoted. In this situation, the next time the subtask reaches the Start action of the select-signoff-team subtask, the user is again prompted to select a signoff team.

RESTRICTIONS

Ignores any invalid arguments without reporting an error.

The keywords always refer to the initiating user because all instances of this handler in a workflow process are executed when the workflow process is initiated, not when tasks are approved.

If the -ce argument is not specified, all instances of this handler are executed when the workflow process is initiated and in this case the keywords refer to the initiating user.

EXAMPLES

- This example places the handler on the Undo action of the select-signoff-team subtask. If the select-signoff-team subtask is demoted, the -ce argument clears the Ad-hoc done check box. When the workflow process returns to the select-signoff-team subtask, the responsible party is again prompted to select the signoff team because the Ad-hoc done check box is clear, indicating the task is not yet complete.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ce</td>
<td></td>
</tr>
</tbody>
</table>

- This example has a valid user, resource pool, address list and handler-specific keywords as argument values. So Smith, the current logged on users group/role resource pool, members of the List1 address list, and the members assigned as reviewers are added as signoff attachments to the select-signoff-team task on which this handler is added. The handler is executed at the time of workflow process initiation.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:Smith, resourcepool:$GROUP::$ROLE, addresslist:List1, $PROPOSED_REVIEWERS</td>
</tr>
<tr>
<td>-quorum</td>
<td>80%</td>
</tr>
</tbody>
</table>

If the handler with the above arguments is specified on the Notify task under the Route task, the signoff attachments are added to the Notify task and used for sending notifications. The quorum is set to 80% which means that of all the signoff members, 80% need to approve for the task to move ahead.

- This example has a valid user, resource pool, address list, and handler-specific keywords as argument values. So Smith, the current logged on users group/role
resource pool, members of List1 address list, and the members assigned as reviewers are added as signoff attachments to the select-signoff-team task on which this handler is added. Because of the -ce option, the handler is executed when the task action on which it is attached is executed. The quorum is set to 80% which means that of all the signoff members, 80% need to approve for the task to move ahead.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:Smith, resourcepool:$GROUP::$ROLE,</td>
</tr>
<tr>
<td></td>
<td>addresslist:List1, $PROPOSED_REVIEWERS</td>
</tr>
<tr>
<td>-quorum</td>
<td>80%</td>
</tr>
<tr>
<td>-ce</td>
<td></td>
</tr>
</tbody>
</table>

If the handler with the above arguments is specified on the Notify task under the Route task, the signoff attachments are added to the Notify task and used for sending notifications.

- This example assigns the user whose ID is Smith to the signoff team

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:Smith</td>
</tr>
</tbody>
</table>

- This example assigns the owning user ID of the first UGMASTER target found by the system to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:$TARGETOWNER[UGMASTER]</td>
</tr>
</tbody>
</table>

- This example assigns the project team administrator of the project team associated with the first target found by the system to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:$PROJECT_TEAM_ADMINISTRATOR</td>
</tr>
</tbody>
</table>

- This example assigns all members of the jhList address list to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>addresslist:jhList</td>
</tr>
</tbody>
</table>

- This example assigns the manufacturing resource pool (any role within the manufacturing group) to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:manufacturing:</td>
</tr>
</tbody>
</table>
- This example assigns the $PROCESSGROUP resource pool (any role within the xyz group, where xyz is the owning group of the workflow process) to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:$PROCESSGROUP::</td>
</tr>
</tbody>
</table>

- This example assigns the $TARGETGROUP resource pool (any roles within the abc group, where abc is the group of the first item revision target) to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:$TARGETGROUP::</td>
</tr>
</tbody>
</table>

- This example assigns the engineer role within the manufacturing group resource pool to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:manufacturing::engineer</td>
</tr>
</tbody>
</table>

- This example assigns the current logged on role within the current logged on group resource pool to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:$GROUP::$ROLE</td>
</tr>
</tbody>
</table>

- This example assigns the engineer role within any group resource pool to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:::engineer</td>
</tr>
</tbody>
</table>

- This example adds user smith and all reviewers of the first target item revision object to the signoff team. The quorum is set to 51% which means that at least more than half of the signoff members need to approve for the perform-signoffs task to move ahead. Because of the -ce option, the handler is executed when the task action on which it is attached is executed.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:smith, $PROPOSED_REVIEWERS</td>
</tr>
<tr>
<td>-quorum</td>
<td>51%</td>
</tr>
<tr>
<td>-ce</td>
<td></td>
</tr>
</tbody>
</table>

- This example adds all members of the Engineering group and Engineer role to the signoff team. The members are dynamically evaluated when the select-signoff-team task completes. The quorum is set to 80% which means that of all the signoff members, 80% need to approve for the task to move ahead.
Because of the -ce option, the handler is executed when the task action on which it is attached is executed.

**Argument** | **Values**
---|---
-assignee | allmembers:Engineering::Engineer
-quorum | 80%

- This example adds all members of the list1 address list and the **Engineering:Engineer** resource pool to the signoff team. The quorum is set to 5 which mean that of all the signoff members, 5 need to approve for the task to move ahead. Because of the -ce option, the handler is executed when the task action on which it is attached is executed.

**Argument** | **Values**
---|---
-assignee | resourcepool:Engineering::Engineer, addresslist:list1
-quorum | 5

- This example has a valid user, resource pool, address list, and handler specific keywords as argument values. So **smith**, the current logged on users group/role resource pool, members of the list1 address list, and the members assigned as reviewers are assigned to the signoff team. Because of the -ce option, the handler is executed when the task action on which it is attached is executed.

**Argument** | **Values**
---|---
-assignee | user:smith,resourcepool:$GROUP::$ROLE, addressList:list1,$PROPOSED_REVIEWERS

If the handler with these arguments is specified on the Notify task under the **Route** task, the signoff attachments are added to the Notify task and used for sending notifications.

- This example has a valid user, resource pool, and handler-specific keywords as values. So **smith**, the current logged on users group/role resource pool, members of the project associated with the first target object, and members assigned as reviewers are added to the signoff team. Because of the -ce option, the handler is executed when the task action on which it is attached is executed.

**Argument** | **Values**
---|---
-assignee | user:smith,resourcepool:$GROUP::$ROLE, $PROJECT_MEMBER,$PROPOSED_REVIEWERS

If the handler with these arguments is specified on the Notify task under the **Route** task, the signoff attachments are added to the Notify task and used for sending notifications.
• This example has a valid user, resource pool, and handler-specific keywords as values. So smith, the current logged-on user group/role resource pool, and CHANGE_REVIEW_BOARD and ANALYST associated with the first change target object are added to the signoff team. Because of the -ce option, the handler is executed when the task action on which it is attached is executed.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:smith,resourcepool:$GROUP::$ROLE, $CHANGE_REVIEW_BOARD,$ANALYST</td>
</tr>
<tr>
<td>-ce</td>
<td></td>
</tr>
</tbody>
</table>

If the handler with these arguments is specified on the Notify task under the Route task, the signoff attachments are added to the Notify task and used for sending notifications.

• This example removes all existing members of the signoff team and adds PROPOSED_RESPONSIBLE_PARTY. Because of the -ce option, the handler is executed when the task action on which it is attached is executed. The AUTO_COMPLETE option allows the task to complete without user interaction by automatically selecting the Ad-hoc done check box in the select-signoff-team subtask interface, and the task does not need to be performed through the interface.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ce</td>
<td></td>
</tr>
<tr>
<td>-clear_signoffs</td>
<td></td>
</tr>
<tr>
<td>-assignee</td>
<td>$PROPOSED_RESPONSIBLE_PARTY</td>
</tr>
<tr>
<td>AUTO_COMPLETE</td>
<td></td>
</tr>
</tbody>
</table>

If the handler with these arguments is specified on the Notify task under the Route task, the signoff attachments are added to the Notify task and used for sending notifications.
AI-process-export

DESCRIPTION
Creates a new RequestObject object under the target ApplicationInterface (AI) object without changing the base references of the AI object.

An AI object is a persistent workspace object that is the repository for the import and export transactions between Teamcenter and an external application for a predefined and configured structure. It contains:

- An ordered list of request objects.
- The transfer mode (import or export).
- The root or top-level object of the structures to exchange. This can be any object that is valid to export from Teamcenter using PLM XML, for example, a structure context, item, or BOM view revision.
- Tracking information to allow updates of changed data (deltas).

For more information about AIs, see the My Teamcenter Guide.

Use this handler in workflows containing at least one AI object as a target, and containing reference attachments such as StructureContext or CollaborationContext objects, or objects accepted by PLM XML export (such as BOM views, BOM view revisions, items, and item revisions).

Note
Without a StructureContext or CollaborationContext object, the PLM XML cannot export a structure, because there is no configuration; only the workspaceObject is exported. Typically, a StructureContext or CollaborationContext object is used as a reference attachment.

SYNTAX

AI-process-export

ARGUMENTS
None.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
The attachments must be placed under the root task.

EXAMPLES
To share an existing CollaborationContext object with another application using PLM XML format, use a workflow template containing this handler. Initiate the workflow against an AI object, selecting the AI object as the target attachment and the CollaborationContext object as the reference attachment. The workflow creates a new RequestObject object. The AI can now be shared with another application.
AI-process-import

DESCRIPTION
Imports the PLM XML associated with the target RequestObject objects. RequestObject objects are contained within ApplicationInterface (AI) objects.
For more information about working with AIs, see the My Teamcenter Guide.

SYNTAX
AI-process-import

ARGUMENTS
None.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
The attachments must be placed under the root task.

EXAMPLES
To import the PLM XML associated with a new RequestObject object created by any client application under an existing AI object, use a workflow template containing this handler. Initiate the workflow against the AI and select one or more RequestObject objects as target attachments, including the new RequestObject. Optionally, also select an ICRevision object as a reference attachment. The structure is updated with the contents of the PLM XML contained within the RequestObject object.
APPJ-update-from-targets

DESCRIPTION
Notifies the Update Manager that the target item revisions have achieved a release status. Notification is performed through update packages, which are discrete packages of change information to be applied to appearance sets. The packages are queued in the database and processed serially, as one update can affect the next. For more information about updating appearances, see Appearance Configuration Guide.

One update package is created for each release status contained within the workflow process. The package references all target item revisions of the workflow processes.

Note
Typically, a workflow process contains only a single release status; therefore, only a single package is created. Workflow functionality does permit multiple release statuses to be contained within a single workflow process, however. Adding this handler to such a workflow process creates multiple update packages.

This handler is intended to run after the add-status handler, which is used to assign release statuses to the target object.

SYNTAX
APPR-update-from-targets

ARGUMENTS
None.

PLACEMENT
Place after the add-status handler (this handler applies status to the targets). Therefore, placement is typically at the end of the workflow process.

RESTRICTIONS
Use only with workflow processes that assign one or more release statuses to one or more target item revisions.
approve-service-structure

DESCRIPTION
Executes an approval process for MRO service structures.

SYNTAX
approve-service-structure

ARGUMENTS
None.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
Use only for approval of MRO service structures inheriting from a transaction element.
auto-assign

DESCRIPTION
Makes the specified user or resource pool the responsible party for the task to which the handler is added. Optionally, you can make the same specified user/resource pool the responsible party for all subtasks of the parent task.

Note
If you use keyword arguments to dynamically generate this assignment, and the system resolve the argument to a user or resource pool, then the argument is ignored.

SYNTAX
auto-assign [-subtasks] {-user=user-id | -person=person-name | resourcepool
|-assignee= {user:user | person:person | resourcepool:group::role
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESSOWNER | $TARGETOWNER [type]
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3]]

ARGUMENTS
-subtasks
Propagates task assignments to subtasks of the current task (nonrecursively). Optional.

-user
Makes the user whose ID is specified the responsible party for the task to which this handler is added.
Accepts a single valid Teamcenter user ID or one of these keywords: $USER or $TARGETOWNER.

-person
Makes the user whose name is specified the responsible party for the task to which this handler is added.
Accepts a single valid Teamcenter person name.

Note
If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use wayne, joan:

-person=wayne, joan

-assignee
Makes the user or resource pool the specified keyword evaluates to the responsible party for the task to which this handler is added. Accepts one of the following in the format specified below:
Workflow handlers

- **user:user**
  Adds the specified user to the signoff member list and as the responsible party for the task to which the handler is attached. Accepts a valid Teamcenter user ID.

- **person:person**
  Adds the person whose name is specified to the signoff member list and as the responsible party for the task to which the handler is attached. Accepts a valid Teamcenter person name.

  **Note**
  If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use **wayne, joan**:
  
  `-assignee=person:wayne\, joan`

- **resourcepool:group::role**
  Results in a single assignment which can be performed by any single member of this group/role.
  
  You can define resource pools in the form of **group::, group::role, or role**. Accepts valid Teamcenter resource pool names and these keywords:
  
  o **$GROUP**
    Current user’s current group.
  
  o **$ROLE**
    Current user’s current role.
  
  o **$TARGETGROUP[type]**
    Owning group of the first target object of the specified type. The `type` value is optional. If not specified, the first target is used.
  
  o **$PROCESSGROUP**
    Owning group of the workflow process.

- **$PROPOSED_RESPONSIBLE_PARTY**
  Affects assignments based on the user assigned as the responsible party for the first target object.

- **$USER**
  Adds the current user to the signoff member list and as the responsible party.

- **$PROCESSOWNER**
  Adds the workflow process owner to the signoff member list and as the responsible party.

- **$TARGETOWNER [type]**
Appendix A  Workflow handlers

Adds the owner of the first target of the specified type to the signoff member list and as the responsible party. The type value is optional. If not specified, the first target is used.

- **$PROJECT_ADMINISTRATOR, $PROJECT_TEAM_ADMINISTRATOR**

  Dynamically adds the project team members belonging to the role specified in the argument value to the signoff member list and as the responsible party. The project team is determined by the project team associated with the first target object.

- **$REQUESTOR, $ANALYST, $CHANGE_SPECIALIST1, $CHANGE_SPECIALIST2, $CHANGE_SPECIALIST3**

  Dynamically resolves to the user or resource pool associated with the first change target object in the workflow process. The particular user or resource pool is determined by the role specified in the argument value.

  **Note**

  Change-related keywords apply only to change objects. If the workflow process does not contain a change object as a target, the argument resolves to null.

  Change Manager does not need to be enabled before these keywords take effect, but during installation, Change Management must be selected under Extensions→Enterprise Knowledge Foundation in Teamcenter Environment Manager.

**PLACEMENT**

Place on the Start action.

**RESTRICTIONS**

None.

**EXAMPLES**

- This example makes Smith the responsible party for the task to which this handler is assigned and all of the task’s subtasks.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-subtasks</td>
<td></td>
</tr>
<tr>
<td>-assignee</td>
<td>user:Smith</td>
</tr>
</tbody>
</table>

- This example makes the workflow process owner the responsible party for the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROCESSOWNER</td>
</tr>
</tbody>
</table>

- This example makes the engineer role within manufacturing group resource pool the responsible party for the task to which this handler is assigned.
### Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>resourcepool:manufacturing::engineer</td>
</tr>
</tbody>
</table>

- This example makes the responsible party group the responsible party for the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROPOSED_RESPONSIBLE_PARTY</td>
</tr>
</tbody>
</table>

- This example makes the project administrator of the project associated with the first target the responsible party for the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROJECT_ADMINISTRATOR</td>
</tr>
</tbody>
</table>

- This example makes the user or resource pool associated as ANALYST with the first change target the responsible party for the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$ANALYST</td>
</tr>
</tbody>
</table>
auto-assign-rest

DESCRIPTION

Automatically makes the specified user or resource pool the responsible party for any unassigned subtasks of the parent task to which this handler is added. You specify the user or resource pool by entering a comma-delimited list in the Arguments column for this handler.

This handler first assumes that the list contains user IDs and attempts to match the entries (in the order listed) to valid user IDs. The first entry matching a user ID is made the responsible party for any subtasks of the task to which this handler is assigned.

If no entries in the list match a valid user ID, the system attempts to match the entries (in the order listed) to valid resource pool names. The first entry matching a resource pool name (group, group/role, or role) is made the responsible party for any subtasks of the task to which this handler is assigned.

If this handler is attached to the root task with no argument specified, the workflow process initiator is made the responsible party for all tasks in the workflow process.

If this handler is attached to the root task and one or more entries are contained in the list, the first valid user or resource pool is made the responsible party for all tasks in the workflow process.

SYNTAX

auto-assign-rest
-assignee [user:user | person:person | resourcepool:group::role]
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESSOWNER | $TARGETOWNER [type]
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3

ARGUMENTS

-assignee

Makes the user or resource pool the specified keyword evaluates to the responsible party for the task to which this handler is added. Accepts one of the following in the format specified below:

• user:user

  Adds the user specified to the signoff member list and as the responsible party for the task to which the handler is attached. Accepts a valid Teamcenter user ID.

• person:person

  Adds the person whose name is specified to the signoff member list and as the responsible party for the task to which the handler is attached. Accepts a valid Teamcenter person name.
Note
If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use **wayne, joan**:

- `assignee=person:wayne\, joan`

- **resourcepool:group::role**
  Results in a single assignment which can be performed by any single member of this group/role.
  You can define resource pools in the form of `group::`, `group::role`, or `role`. Accepts valid Teamcenter resource pool names and these keywords:
  - `$GROUP`
    Current user’s current group.
  - `$ROLE`
    Current user’s current role.
  - `$TARGETGROUP[type]`
    Owning group of the first target object of the specified type. The `type` value is optional. If not specified, the first target is used.
  - `$PROCESSGROUP`
    Owning group of the workflow process.

- **$PROPOSED_RESPONSIBLE_PARTY**
  Affects assignments based on the user assigned as the responsible party for the first target object.

- **$USER**
  Adds the current user to the signoff member list and as the responsible party.

- **$PROCESSOWNER**
  Adds the workflow process owner to the signoff member list and as the responsible party.

- **$TARGETOWNER [type]**
  Adds the owner of the first target of the specified type to the signoff member list and as the responsible party. The `type` value is optional. If not specified, the first target is used.

- **$PROJECT_ADMINISTRATOR, $PROJECT_TEAM_ADMINISTRATOR**
  Dynamically adds the project team members belonging to the role specified in the argument value to the signoff member list and as the responsible party. The project team is determined by the project team associated with the first target object.
Appendix A  Workflow handlers

- **$REQUESTOR, $ANALYST, $CHANGE_SPECIALIST1, $CHANGE_SPECIALIST2, $CHANGE_SPECIALIST3**

Dynamically resolves to the user or resource pool associated with the first change target object in the workflow process. The particular user or resource pool is determined by the role specified in the argument value.

**Note**

Change-related keywords apply only to change objects. If the workflow process does not contain a change object as a target, the argument resolves to null.

Change Manager does not need to be enabled before these keywords take effect, but during installation, **Change Management** must be selected under **Extensions→Enterprise Knowledge Foundation** in Teamcenter Environment Manager.

**PLACEMENT**

Place on the **Start** action. Typically placed on the root task after the **CR-assign-team-selector** handler.

**RESTRICTIONS**

None.

**EXAMPLES**

- In this example, a five-task workflow process containing the task templates below is initiated by user **Jones**. The **auto-assign-rest** handler is placed on the root task, and the **auto-assign** handler is placed on the fourth task, set with the **-assignee=$PROCESSOWNER** argument.

  The workflow consists of a **Do** task, **Review** task, **Checklist** task, **Review** task, and **Do** task.

  Because the **auto-assign-rest** handler is placed on the root task and **Smith** is specified with the **-assignee** argument, **Smith** is the responsible party for the first three tasks (and their subtasks). Because the **auto-assign** **-assignee=$PROCESSOWNER** handler is placed on the fourth task, **Jones** is the responsible party for the fourth task and its subtasks. **Smith** is the owner of the fifth task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:Smith</td>
</tr>
</tbody>
</table>

- This example assigns the user or resource pool assigned as the responsible party for the subtasks of the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROPOSED_RESPONSIBLE_PARTY</td>
</tr>
</tbody>
</table>

- This example assigns the user or resource pool associated as **ANALYST** with the first change target object the responsible party for the subtasks of the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$ANALYST</td>
</tr>
</tbody>
</table>
auto-relocate-file

DESCRIPTION
Relocates all released datasets of a job to a specified directory. Teamcenter does not automatically register this handler. Users have to register and modify the handler code to suit their requirements, using the sample code provided. For more information about using this handler and to reference the sample code, see the Server Customization Programmer’s Guide.
CAE-batch-meshing-handler

DESCRIPTION
Launches the specified batch meshing tool from a workflow.

SYNTAX
CAE-batch-meshing-handler -tool=toolname

ARGUMENTS
-tool
The name of the batch meshing tool to launch. The name must match the batch meshing tool name defined in the Meshing Tools list in the Options dialog box (Edit→Options→CAE Tools→Batch Meshing). The -tool argument is required.

RESTRICTIONS
None.
### CAE-simulation-process-launch-handler

**DESCRIPTION**
Launches the specified simulation tool.

**SYNTAX**

```
```

**ARGUMENTS**

- **-tool**
The name of the simulation tool to launch.

  **Note**
The simulation tool name you specify here must match the simulation tool name defined in the Simulation Tool Configuration dialog box in CAE Manager.

- **-launch**
This argument is used only when both the Remote Launch and the Local Launch options are selected in the Simulation Tool Configuration dialog box in CAE Manager.

  **Note**
  If this value is not specified, the handler assumes the launch type to be local, this is, the machine on which Teamcenter server is running.

- **-nosync**
If specified, a synchronous process running in the background does not inform the task about its completion. As a result, the control from the current task goes to the next task (if any) as soon as the current task starts.

  If not specified, the task waits for the execution of the process to complete before moving to the next task.

  **Note**
  This argument is valid for local launch only. Remote launch is always executed in non-synchronous mode.

- **-continue**
If specified, the current task moves to the next task after completion even if the current task fails.

  If not specified, the task stops on failure.
Appendix A  

Workflow handlers

Note
This argument is valid for local launch only. Remote launch is always executed in nonsynchronous mode.

This argument is not valid if you specify the -nosync argument.

-noref
If specified, the handler does not add output objects as reference attachments.

If not specified, the handler adds output objects as reference attachments in the Reference folder.

Note
This argument is valid for local launch only. Remote launch is always executed in nonsynchronous mode and output objects are never added as reference attachments.

This argument is not valid if you specify the -nosync argument.

-param::paramName
Used to assign run-time parameter values for any parameters already defined as part of the tool configuration in the Simulation Tool Configuration dialog box in CAE Manager.

Launches the tool with the paramName value for the paramName parameter as defined in the tool configuration. The specified parameters are processed according to the defined configuration.

Note
The paramName value must be defined as a run-time parameter for the tool configuration in the Simulation Tool Configuration dialog box. Any run-time parameters defined in the tool configuration that are not indicated as action handler arguments get the default values defined in the tool configuration.

The paramName value can be an empty string, in which case the default value of the corresponding paramName is overridden with an empty value.

Restrictions
None.
change-all-started-to-pending

DESCRIPTION
Ensures that all tasks that are started, but not completed, are cleaned up at the conclusion of the workflow process.

SYNTAX
change-all-started-to-pending

ARGUMENTS
None.

PLACEMENT
Place on the Complete action of the root task.

RESTRICTIONS
None.
Appendix A  Workflow handlers

CR-assign-team-selector

DESCRIPTION
Assigns all select-signoff-team tasks in the entire workflow process to the specified user, person, initiator (owner), or resource pool of the workflow process. Only one argument can be defined; all arguments are mutually exclusive of each other.

SYNTAX
CR-assign-team-selector
-assignee= [user:user | person:person | resourcepool:group::role
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESSOWNER | $TARGETOWNER [type]
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR]
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3

ARGUMENTS
-assignee
Makes the user or resource pool the specified keyword evaluates to the responsible party for the task to which this handler is added. Accepts one of the following in the format specified below:

• user:user
  Adds the user specified to the signoff member list for the task to which it is attached. Accepts a valid Teamcenter user ID.

• person:person
  Adds the user whose name is specified to the signoff member list for the task to which it is attached. Accepts a valid Teamcenter person name.

  Note
  If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use wayne, joan:

  -assignee=person:wayne\, joan

• resourcepool:group::role
  Results in a single assignment which can be performed by any single member of this group/role.
  You can define resource pools in the form of group::, group::role, or role. Accepts valid Teamcenter resource pool names and these keywords:

  o $GROUP
    Current user’s current group.

  o $ROLE
    Current user’s current role.
o $TARGETGROUP [type]
   Owning group of the first target object of the specified type. The type value is optional. If not specified, the first target is used.

o $PROCESSGROUP
   Owning group of the workflow process.

- $PROPOSED_RESPONSIBLE_PARY
  Affects assignments based on the user assigned as the responsible party for the first target object.

- $USER
  Adds the current user to the signoff member list.

- $PROCESSOWNER
  Adds the workflow process owner to the signoff member list.

- $TARGETOWNER [type]
  Adds the owner of the first target of specified type to the signoff member list. The type value is optional. If not specified, the first target is used.

- $PROJECT_ADMINISTRATOR, $PROJECT_TEAM_ADMINISTRATOR
  Dynamically adds the project team members belonging to the role specified in the argument value. The project team is determined by the project team associated with the first target object.

- $REQUESTOR, $ANALYST, $CHANGE_SPECIALIST1, $CHANGE_SPECIALIST2, $CHANGE_SPECIALIST3
  Dynamically resolves to the user or resource pool associated with the first change target object in the workflow process. The particular user or resource pool is determined by the role specified in the argument value.

   **Note**

   Change-related keywords apply only to change objects. If the workflow process does not contain a change object as a target, the argument resolves to null.

   Change Manager does not need to be enabled before these keywords take effect, but during installation, Change Management must be selected under Extensions→Enterprise Knowledge Foundation in Teamcenter Environment Manager.

**PLACEMENT**

Place on the **Start** action of the root task.

**RESTRICTIONS**

None.

**EXAMPLES**

- This example assigns the user **jim** all **select-signoff-team** tasks in that workflow process.
## Appendix A  Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:jim</td>
</tr>
</tbody>
</table>

- This example assigns the person **Jim Smith** all `select-signoff-team` tasks in that workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>person:Jim Smith</td>
</tr>
</tbody>
</table>

- This example assigns the owner of the workflow process all `select-signoff-team` tasks in that workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROCESSOWNER</td>
</tr>
</tbody>
</table>

- This example assigns the user or resource pool assigned as the responsible party for all `select-signoff-team` tasks in that workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROPOSED_RESPONSIBLE_PARTY</td>
</tr>
</tbody>
</table>

- This example makes the project administrator of the project associated with the first target the responsible party for all `select-signoff-team` tasks in that workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROJECT_ADMINISTRATOR</td>
</tr>
</tbody>
</table>

- This example makes the user or resource pool associated as **REQUESTOR** with the first change target the responsible party for all `select-signoff-team` tasks in the workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$REQUESTOR</td>
</tr>
</tbody>
</table>
CR-change-group-owner

**DESCRIPTION**
Changes the owning group for the item master of any item type whose revision is attached as target.

**SYNTAX**

```
CR-change-group-owner -group=group-id
```

**ARGUMENTS**

- **-group**
  A valid Teamcenter group_id.

**PLACEMENT**
Place on the Complete action.

**RESTRICTIONS**
None.

**EXAMPLES**

- This example is used with a workflow initiated with an item revision and document revision attached as targets. It sets the owning group of the respective master item and master document to **engineering**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-group</td>
<td>engineering</td>
</tr>
</tbody>
</table>
Appendix A  Workflow handlers

CR-change-target-group

DESCRIPTION
Changes the group ownership of the target objects to the current $\text{group.id}$ of the user. If the target is an item revision object, the group of its item master is set to the current group ID of the user as well.

SYNTAX

```
CR-change-target-group
```

ARGUMENTS

None.

PLACEMENT

Place on the Complete action.

RESTRICTIONS

None.
CR-change-target-group-owner

DESCRIPTION
Changes the owner and/or the owning group for the target objects.

Note
The handler does not validate if the owning user belongs to the owning group. It makes the change even if the user does not belong to the group.

SYNTAX
CR-change-target-group-owner [-owner=\texttt{user-id}][\texttt{-group=group-id}]  

ARGUMENTS
- \	exttt{-owner}
Valid Teamcenter \texttt{user_id}.

- \	exttt{-group}
Valid Teamcenter \texttt{group_id}.

PLACEMENT
Place on the Complete action.

RESTRICTIONS
None.

EXAMPLES
• This example changes the group and owner of the targets to \texttt{engineering} and \texttt{jim}, respectively.

\begin{tabular}{ll}
\textbf{Argument} & \textbf{Values} \\
\texttt{-owner} & \texttt{jim} \\
\texttt{-group} & \texttt{engineering} \\
\end{tabular}

• This example changes the only group of the targets to \texttt{production}.

\begin{tabular}{ll}
\textbf{Argument} & \textbf{Values} \\
\texttt{-group} & \texttt{production} \\
\end{tabular}

• This example changes only the owner of the targets to \texttt{smith}.

\begin{tabular}{ll}
\textbf{Argument} & \textbf{Values} \\
\texttt{-owner} & \texttt{smith} \\
\end{tabular}
CR-fill-in-reviewers

DESCRIPTION

Automatically assigns signoff reviewers that meet specified user, group, or role criteria for the specified Review task. This criteria populates the signoff profiles.

This handler compares the assigned user with the profile definition in the corresponding select-signoff-team task. If the assigned user does not match the profile definition, automatic assignment does not occur and the select-signoff-team task must be performed manually.

SYNTAX

CR-fill-in-reviewers
-assignee= [user:user | person:person | addresslist:list
 | resourcepool:group::role | allmembers:group::role
 | $PROPOSED_RESPONSIBLE_PARTY | $PROPOSED_REVIEWERS | $USER
 | $PROCESSOWNER | $TARGETOWNER [type]
 | $PROJECT_ADMINISTRATOR
 | $PROJECT_TEAM_ADMINISTRATOR
 | $PROJECT_AUTHOR | $PROJECT_MEMBER
 | $REQUESTOR | $ANALYST
 | $CHANGE_SPECIALIST1 | $CHANGE_SPECIALIST2 | $CHANGE_SPECIALIST3
 | $CHANGE_REVIEW_BOARD | $CHANGE_IMPLEMENTATION_BOARD]
 [-add_excess_as_adhoc]
 [-review_task_name=review-task-name]

ARGUMENTS

-assignee
Assigns the specified users, role members, group members, and/or resource pool members to the signoff team.

• user:user
  Adds the user specified to the signoff member list for the task to which it is attached. Accepts a valid Teamcenter user ID.

• person:person
  Adds the user whose name is specified to the signoff member list for the task to which it is attached. Accepts a valid Teamcenter person name.

  Note
  If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use wayne, joan:

  -assignee=person:wayne\\, joan

• addresslist:list
  Adds all members of the address list specified to the signoff member list.

• resourcepool:group::role
  Results in a single assignment which can be performed by any single member of this group/role.
You can define resource pools in the form of \textit{group::}, \textit{group::role}, or \textit{role}. Accepts valid Teamcenter resource pool names and these keywords:

- $\textit{GROUP}$
  - Current user’s current group.

- $\textit{ROLE}$
  - Current user’s current role.

- $\textit{TARGETGROUP}[\textit{type}]$
  - Owning group of the first target object of the specified type. The \textit{type} value is optional. If not specified, the first target is used.

- $\textit{PROCESSGROUP}$
  - Owning group of the workflow process.

- \textit{allmembers:group::role}
  - Adds all members of a \textit{group::role} combination to the signoff member list. You can define role in groups in the form of \textit{group::}, \textit{group::role}, or \textit{role}. Accepts valid Teamcenter resource pool names and these keywords:

  - $\textit{GROUP}$
    - Current user’s current group.

  - $\textit{ROLE}$
    - Current user’s current role.

  - $\textit{TARGETGROUP}[\textit{type}]$
    - Owning group of the first target object of the specified type. The \textit{type} value is optional. If not specified, the first target is used.

  - $\textit{PROCESSGROUP}$
    - Owning group of the workflow process.

- $\textit{PROPOSED\_RESPONSIBLE\_PARTY}$
  - Affects assignments based on the user assigned as the responsible party for the first target object.

- $\textit{PROPOSED\_REVIEWERS}$
  - Affects assignments based on members assigned as reviewers for the first target object.

- $\textit{USER}$
  - Adds the current user to the signoff member list.

  If $\textit{USER}$ is used, and the current user belongs to several groups and roles, the behavior of the $\textit{USER}$ keyword depends on the value of the \textit{SIGNOFF\_fill\_in\_reviewers} site preference, as follows:
Appendix A  Workflow handlers

- 1
Attempts to match the current user’s group/role values with the profile first, default values second, then any other groups/roles of which the current user is a member. This is the default setting.

- 2
Attempts to match the current user’s group/role values first, default values of which the current user is a member second.

- 3
Attempts to match the current user’s group/role values.

- $PROCESSOWNER
Adds the workflow process owner to the signoff member list.

- $TARGETOWNER [type]
Adds the owner of the first target of specified type to the signoff member list. The type value is optional. If not specified, the first target is used.

- $PROJECT_ADMINISTRATOR, $PROJECT_TEAM_ADMINISTRATOR, $PROJECT_AUTHOR, $PROJECT_MEMBER
Dynamically adds the project team members belonging to the role specified in the argument value. The project team is determined by the project team associated with the first target object.

- $REQUESTOR, $ANALYST, $CHANGE_SPECIALIST1, $CHANGE_SPECIALIST2, $CHANGE_SPECIALIST3, $CHANGE_REVIEW_BOARD, $CHANGE_IMPLEMENTATION_BOARD
Dynamically resolves to the user or resource pool associated with the first Change target object in the process. The particular user or resource pool is determined by the role specified in the argument value.

Note
Change-related keywords apply only to change objects. If the process does not contain a change object as a target, the argument resolves to null.

Change Manager does not need to be enabled before these keywords take effect, but during installation, Change Management must be selected under Extensions→Enterprise Knowledge Foundation in Teamcenter Environment Manager.

-add_excess_as_adhoc
(Optional.) Adds the rest of the assignees as ad hoc users if the profile is satisfied.

-review_task_name
(Optional.) Specifies the Review task name to which the reviewers are added.

PLACEMENT
Place either on the Start action of the relevant select-signoff-team task or on the root task with the -review_task_name argument.
RESTRICTIONS

Use only with the **select-signoff-team** task or on the root task.

EXAMPLES

- This example designates the user **tom** and all members of the **engineering** group as reviewers for the **Review** task called **Review Task 1**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:tom, allmembers:engineering::</td>
</tr>
<tr>
<td>-review_task_name</td>
<td>$ROOTTask.Review Task 1</td>
</tr>
</tbody>
</table>

- This example shows the current user added as a reviewer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:$USER</td>
</tr>
<tr>
<td>-review_task_name</td>
<td>Review Task 1</td>
</tr>
</tbody>
</table>

- This example designates members assigned as reviewers for the first target object as reviewers for the **Review** task called **Review Task 1**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>$PROPOSED_REVIEWERS</td>
</tr>
<tr>
<td>-review_task_name</td>
<td>Review Task 1</td>
</tr>
</tbody>
</table>

- This example designates user **tom**, all the members of the **Engineering** group, and the **REQUESTOR** associated with the first change target object as reviewers for the **Review** task named **Review Task 1**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-assignee</td>
<td>user:tom, allmembers:engineering::$REQUESTOR</td>
</tr>
<tr>
<td>-review_task_name</td>
<td>Review Task 1</td>
</tr>
</tbody>
</table>

If the handler with these arguments is specified on the **Notify** task under the **Route** task, the signoff attachments are added to the **Notify** task and used for sending notifications.
CR-notify

DESCRIPTION
Sends a report through the operating system (OS) mail to all task reviewers. CR-notify does not notify users through Teamcenter e-mail. If you want to send the report using Teamcenter e-mail, use the notify handler.

The -report argument differentiates CR-notify handler from the notify handler. In notification e-mail, the -report argument appends a report describing the signoff data associated with the perform-signoffs task. CR-notify is designated for use on the perform-signoffs task. The notify handler is used on any type of task.

SYNTAX
CR-notify
-report={review|rejection|progress|level}
-recipient=
{OS:|user-name | user:user | person:person | addresslist:value
| resourcepool:group::role
| allmembers:group::role
| $USER | $REVIEWERS | $PROPOSED_REVIEWERS
| $RESPONSIBLE_PARTY | $PROPOSED_RESPONSIBLE_PARTY
| $PROCESSOWNER | $TARGETOWNER [type]
| $UNDECIDED | $RESOURCE_POOL_ALL
| $RESOURCE_POOL_NONE | $RESOURCE_POOL_SUBSCRIBED
| $PROJECT_ADMINISTRATOR | $PROJECT_MEMBER
| $PROJECT_TEAM_ADMINISTRATOR
| $PROJECT_AUTHOR| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1 | $CHANGE_SPECIALIST2 | $CHANGE_SPECIALIST3
| $CHANGE_REVIEW_BOARD | $CHANGE_IMPLEMENTATION_BOARD
[-subject=string]
[-comments=string]
[-url=(rich | dhtml)]

ARGUMENTS
-report
Indicates the report type sent to recipients. Accepts one of these values:

• review
Notifies all recipients when they must review target objects. The report lists target and reference object IDs and types.

• rejection
Notifies recipients that the Review task has been rejected. The report lists target and reference object IDs, as well as the Review task reviewers, decisions, dates, and comments for each Review task. Do not use this value unless you want the workflow process to always send a rejection notice.

• progress
Notifies recipients of the current state of the workflow process. The report lists the target and reference object names, object IDs (if applicable for the object), as well as the Review task reviewers, decisions, dates, and comments for each Review task.
Workflow handlers

- level
  Notifies recipients when the **Review** task completes. The report lists the target and reference object IDs, as well as the current **Review** task reviewers, decisions, dates, and comments.

- subject
  Specifies the subject of the report.
  Each type of report is formatted and mailed with the default subject. This is an additional user-defined subject.

- comments
  Specifies an additional user-defined comment.

- recipient
  Specifies the task reviewers to receive notification. Accepts one of these values:
  - **OS**:user-name
    Sends a notification to the OS user name specified.
    `user-name` is a single valid OS user name.
  - **user**:user
    Sends notification to the user specified.
    `user` is a single valid Teamcenter user ID.
  - **person**:person
    Sends a notification to user whose name is specified.
    `person` is a single valid Teamcenter person.

    **Note**
    If the person’s name includes a comma, you must include an escape character (`\`) to add the correct person. For example, to use `wayne, joan`:
    
    `-recipient=person:wayne\, joan`

  - **addresslist**:list
    Adds all members of the address list specified to the signoff member list. Sends notification to all members of a group/role combination.
    `list` is a valid Teamcenter address list.

  - **resourcepool**:group::role
    Sends notification to members of a group/role combination. Notification is sent to all members, subscribed members, or none based on the **EPM_resource_pool_recipients** preference.
    The preference value can be overridden with:
    - **$RESOURCE_POOL_ALL**
    - **$RESOURCE_POOL_SUBSCRIBED**
### Appendix A  
*Workflow handlers*

- **$RESOURCE_POOL_NONE**
  You can define role in groups in the form of `group::, group::role, or role`.
  Accepts valid Teamcenter resource pool names and these keywords:
  - **$GROUP**
    The current user's current group.
  - **$ROLE**
    The current user's current role.
  - **$TARGETGROUP [type]**
    The owning group of the first target object of the specified type. The `type` value is optional. If not specified, the first target is used.
  - **$PROCESSGROUP**
    The owning group of the workflow process.

- **allmembers::group::role**
  Sends notification to all members of a group/role combination.
  You can define role in groups in the form of `group::, group::role, or role`.
  Accepts valid Teamcenter group and role names and these keywords:
  - **$GROUP**
    The current user's current group.
  - **$ROLE**
    The current user's current role.
  - **$TARGETGROUP [type]**
    The owning group of the first target object of the specified type. The `type` value is optional. If not specified, the first target is used.
  - **$PROCESSGROUP**
    The owning group of the workflow process.

- **$USER**
  Send notification to the current user.

- **$REVIEWERS**
  Builds a list of all users who are reviewers in the same task level as the current reviewer, and sends e-mail to them all.

- **$PROPOSED_REVIEWERS**
  Builds a list of all users who are reviewers in the same task level as the current reviewer, and sends notification to all of them.
• $RESPONSIBLE_PARTY
  Sends the notification to the designated responsible party for the task.

• $PROPOSED_RESPONSIBLE_PARTY
  Sends the notification to the designated responsible party for the task.

• $PROCESSOWNER
  Sends notification to the workflow process owner.

• $TARGETOWNER [type]
  Adds the owner of the first target of specified type to the signoff member list.
  The type value is optional. If not specified, the first target is used.

• $UNDECIDED
  Sends notification to the users who have not set the decision for the task.

• $RESOURCE_POOL_ALL
  Identifies all members of the resource pool.
  This argument has an affect only when it is used along with resourcepool, $REVIEWERS, $PROPOSED_REVIEWERS, $UNDECIDED, $RESPONSIBLE_PARTY, or $PROPOSED_RESPONSIBLE_PARTY.

  When this argument is used along with resourcepool>, e-mail is sent to all the members of the resource pool.

  When this argument is used along with $REVIEWERS or $PROPOSED_REVIEWERS, and if a resource pool is assigned as a reviewer, e-mail is sent to all the members of that resource pool.

  When this argument is used with $UNDECIDED, and if a resource pool is assigned as a reviewer, and no signoff decision has been made for this resource pool assignment, all members of that resource pool are notified.

  When this argument is used along with $RESPONSIBLE_PARTY or $PROPOSED_RESPONSIBLE_PARTY, and if a resource pool is assigned as responsible party, e-mail is sent to all members of that resource pool.

• $RESOURCE_POOL_NONE
  This argument has an effect only when it is used along with resourcepool, $REVIEWERS, $PROPOSED_REVIEWERS, $UNDECIDED, $RESPONSIBLE_PARTY, or $PROPOSED_RESPONSIBLE_PARTY.

  When this is used along with resourcepool, e-mail is not sent to members of the resource pool. (This combination is allowed, but of no value.)

  When this argument is used along with $REVIEWERS, $PROPOSED_REVIEWERS, or $UNDECIDED, and if a resource pool is assigned as a reviewer, e-mail is not sent to members or subscribers of the resource pool.

  When this argument is used along with $RESPONSIBLE_PARTY or $PROPOSED_RESPONSIBLE_PARTY, and if a resource pool is assigned as a responsible party, e-mail is not sent to members or subscribers of resource pool.
Appendix A  Workflow handlers

- **$RESOURCE_POOL_SUBSCRIBED**
  Identifies the users who have subscribed to resource pool.
  This argument has an effect only when it is used along with resourcepool, $REVIEWERS, $PROPOSED_REVIEWERS, $UNDECIDED, $RESPONSIBLE_PARTY, or $PROPOSED_RESPONSIBLE_PARTY.

  When this is used along with resourcepool, e-mail is sent to users who are subscribed to the resource pool.

  When this argument is used with $REVIEWERS or $PROPOSED_REVIEWERS, and if a resource pool is assigned as a reviewer, e-mail is sent to users who are subscribed to the resource pool.

  When this argument is used with $UNDECIDED, and if a resource pool is assigned as a reviewer and no signoff decision has been made for this resource pool assignment, e-mail is sent to users who subscribed to the resource pool.

  When this argument is used with $RESPONSIBLE_PARTY or $PROPOSED_RESPONSIBLE_PARTY, and if a resource pool is assigned as a responsible party, e-mail is sent to users who subscribed to the resource pool.

- **$PROJECT_ADMINISTRATOR**
  $PROJECT_MEMBER
  $PROJECT_TEAM_ADMINISTRATOR
  $PROJECT_AUTHOR

  Dynamically evaluates project team members belonging to the role specified in the argument value and sends notification to them. The project team is determined by the project team associated with the target object.

- **$REQUESTOR**
  $ANALYST
  $CHANGE_SPECIALIST1
  $CHANGE_SPECIALIST2
  $CHANGE_SPECIALIST3
  $CHANGE_REVIEW_BOARD
  $CHANGEIMPLEMENTATION_BOARD

  Dynamically resolves to the user or resource pool associated with the first change target object in the process. The particular user or resource pool is determined by the role specified in the argument value.

  **Note**

  Change-related keywords apply only to change objects. If the process does not contain a change object as a target, the argument resolves to null.

  Change Manager does not need to be enabled before these keywords take effect, but during installation, Change Management must be selected under Extensions→Enterprise Knowledge Foundation in Teamcenter Environment Manager.
Note

If the $RESOURCE_POOL_XXXX argument is not defined and the $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY arguments are used for a case where assignments are made to resource pools, the e-mail is sent using the EPM_resource_pool_recipients preference.

The EPM_resource_pool_recipients preference can have one of the following values:

- **all**
  Sends e-mail to all members of resource pool.

- **none**
  Does not send an e-mail to members or subscribers of resource pool.

- **subscribed**
  Sends e-mail to Teamcenter users who have subscribed to resource pool.

If the $RESOURCE_POOL_XXXX argument is defined, the argument takes precedence over the preference value. If this argument is not defined and the EPM_resource_pool_recipients preference is not set, then subscribed is the default preference.

The -recipient argument can have multiple values by using a delimiter specified by the EPM_ARG_target_user_group_list_separator preference. The default value for this preference is a comma.

**-subject**
(Optional.) Inserts the specified string in the subject line of the e-mail.

**-comments**
(Optional.) Inserts the specified string in the body of the e-mail.

**-url**
(Optional.) Inserts a DHTML link to the workflow process into the notification e-mail, based on the value for -url. If no value is specified for -url, both links are added into the notification e-mail.

If the -url argument is not defined, the notification e-mail contains links depending on the values set in the EPM_notify_url_format preference.

**EPM_notify_url_format** can take the following values:

- **rich**
  Inserts a rich client link to the workflow process into the notification e-mail.

- **dhtml**
  Inserts a thin client (DHTML) link to the workflow process into the notification e-mail.

If the -url argument is not defined and the EPM_notify_url_format preference is not set in the preference file, rich client and thin client links are added to
the notification e-mail as a default. The URL is generated only when the
WEB_default_site_server preference is set to the thin client server node name.

Note
Rich client URL functionality must be enabled for links to rich client workflow
processes to launch the rich client.

PLACEMENT

review
Place on the Start action of the perform-signoffs task when using this argument.

rejection
Place on the Perform or Undo actions of the perform-signoffs task when using
this argument.

When placed on a Perform action, an e-mail is sent on a Reject action.

Only place on an Undo action when the next task is a Review task, and the design
of the workflow process requires that the task is demoted on a Reject action. This is
achieved by placing the demote-on-reject handler on the Perform action of the
perform-signoffs task. A Reject action causes a demotion to the previous task,
which invokes the Undo action, and the CR-notify handler sends out the required
notification.

progress
The recommended placement when using this argument is attached to the Start or
Complete actions of a perform-signoffs task.

level
The recommended placement when using this argument is attached to the Complete
action of a perform-signoffs task.

RESTRICTIONS

Use only on the perform-signoffs task.

EXAMPLES

• This example designates the user smith, members of the manufacturing
group, the OS users peters and john, users with the manager role, members of
the VendorList address list, and project members as recipients of a progress
report with the subject Manufacturing Release Process Completed.

The CR-notify handler should be placed on Complete action of
perform-signoffs task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>progress</td>
</tr>
<tr>
<td>-subject</td>
<td>Manufacturing Release Process Completed</td>
</tr>
<tr>
<td>-recipient</td>
<td>users:smith, os:peters, os:john, allmembers:manufacturing, allmembers:::manager, addresslist:VendorList, $PROJECT_MEMBER</td>
</tr>
</tbody>
</table>
• This example designates the workflow process owner as the recipient of a progress report with the subject **Manufacturing Release Process Completed**.

The **CR-notify** handler should be placed on **Complete** action of **perform-signoffs** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>progress</td>
</tr>
<tr>
<td>-subject</td>
<td>Manufacturing Release Process Completed</td>
</tr>
<tr>
<td>-recipient</td>
<td>$PROCESSOWNER</td>
</tr>
</tbody>
</table>

• This example builds a list of all users assigned as reviewers for the **perform-signoffs** task.

The **CR-notify** handler should be placed on **Start** action of **perform-signoffs** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>progress</td>
</tr>
<tr>
<td>-recipient</td>
<td>$PROPOSED_REVIEWERS</td>
</tr>
</tbody>
</table>

• This example designates the task owner and task reviewers as recipients of a review report with the subject **TASK REVIEW NOTIFICATION**.

If any resource pool is assigned as a reviewer, then all users who have subscribed to that resource pool receive notification e-mail.

Place the **CR-notify** handler on the **Start** action of the **perform-signoffs** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>review</td>
</tr>
<tr>
<td>-subject</td>
<td>TASK REVIEW NOTIFICATION</td>
</tr>
<tr>
<td>-comments</td>
<td>Please review the task</td>
</tr>
<tr>
<td>-recipient</td>
<td>$PROCESSOWNER, $PROPOSED_REVIEWERS, $RESOURCE_POOL_SUBSCRIBED</td>
</tr>
</tbody>
</table>

• This example illustrates creating a workflow process template with a **Review** task. Add the **CR-notify** handler in the **Undo** action of the **perform-signoffs** task. Place a **demote-on-reject** handler on the **Perform** action of the **perform-signoffs** task.

The notification is sent to task owner, responsible party, and reviewers. If any resource pool is assigned as a responsible party and/or as a reviewer, then notification is sent to all group members of that resource pool.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>rejection</td>
</tr>
</tbody>
</table>
This example designates the **REQUESTOR** of the first change target object the recipient of a progress report with the subject **Manufacturing Release Process Completed**.

Place the **CR-notify** handler on the **Complete** action of the **perform-signoffs** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-subject</td>
<td>TASK REJECTION &amp; DEMOTE NOTIFICATION</td>
</tr>
<tr>
<td>-recipient</td>
<td>$RESOURCE_POOL_ALL, $PROCESSOWNER, $PROPOSED_RESPONSIBLE_PARTY, $PROPOSED_REVIEWERS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>Progress</td>
</tr>
<tr>
<td>-subject</td>
<td>Manufacturing Release Process Completed</td>
</tr>
<tr>
<td>-recipient</td>
<td>$REQUESTOR</td>
</tr>
</tbody>
</table>
create-status

DESCRIPTION
Attaches the specified status type to the root task.

For more information about applying the status to the target data, see add-status.

SYNTAX
create-status statustype

ARGUMENTS
statustype
Adds the specified status type to the root task. If this argument is not supplied, the workflow process name is used. The name provided should be the name of a status type already defined in the Business Modeler IDE, not the display name.

For more information about defining status types, see the Business Modeler IDE Guide.

If it is not, a status object is created that is not based on a status type, which means that effectivity and configuration may not work against it.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
• This example attaches the Released status to the root task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Released</td>
<td></td>
</tr>
</tbody>
</table>
**debug**

**DESCRIPTION**
Allows you to print information (for example, state, action, and arguments) about the last action triggered. Typically used for debugging.

**SYNTAX**
```
dbg
```

**ARGUMENTS**
None.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
None.
demote

DESCRIPTION
Clears all signoff decisions from the current and previous Review tasks. An optional argument allows the user to specify the task name that the workflow process is demoted to.

SYNTAX
demote [-level=levelname]

ARGUMENTS
- **-level**
  Specifies to which previous task the workflow process is demoted. Must specify a valid task in the current workflow process.

  If this argument is not specified, the workflow process is demoted to the previous task.

PLACEMENT
None.

RESTRICTIONS
None.

EXAMPLES
This example shows how to demote the workflow process to the task named design.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-level</td>
<td>design</td>
</tr>
</tbody>
</table>
demote-on-reject

DESCRIPTION
Demotes the current task to the previous task, or to the task specified on the -level argument of the demote handler placed on the Undo action of the current task.

By default, the handler checks the quorum requirements at each rejection and demotes the task when the quorum limit cannot be met. Consider a perform-signoffs task assigned to five reviewers with a quorum of three. The first two rejections do not demote the task. The third rejection, which would prevent the requirement of quorum of three from being met, demotes the task.

You can override the default behavior and specify the number of rejections required to demote the workflow process using the -num_rejections argument. Using the above example, override the quorum of three by setting this argument at 2. The task demotes on the second rejection.

To set the required number of rejections at the original quorum amount, type -1. Using the above example, setting the argument at -1 sets the required number of rejections at 3, which is the quorum.

**Note**
This handler takes precedence if success and failure paths exist.

SYNTAX
demote-on-reject [-num_rejections=number-of-rejections]

ARGUMENTS
-num_rejections
Number of rejections required to demote the task.

Specifying -1 reads the quorum value and sets that value as the number of rejections required to demote the current task.

This argument is optional.

PLACEMENT
Place on the Perform action of the perform-signoffs subtask of a Review task.

RESTRICTIONS
- Use only for CR. Do not use this on other EPM applications or workflow process templates.
- This handler assumes that all target objects, reference objects and status types are attached to the root task.

EXAMPLES
- This example demotes a process when the number of rejections exceed the quorum limit:
  demote-on-reject
- This example demotes a process when the second rejection is received:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-num_rejections</td>
<td>2</td>
</tr>
</tbody>
</table>
• This example demotes a process when the number of rejections equals the defined quorum limit:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-num_rejections</td>
<td>-1</td>
</tr>
</tbody>
</table>
DOCMGT-render-document-revision

DESCRIPTION

Translates datasets associated with (the target) item revisions to derived visualization datasets, for example, MS Word to PDF. The Item Revision Definition Configuration (IRDC) and Dispatcher Service Configuration settings are used to determine the source and output file formats. The item revision must be valid and checked in.

The translation process is asynchronous and the workflow process continues after the translation is initiated. This handler is dependent on Dispatcher for translation. The translated files are stored in the Teamcenter database and may be related to the source dataset or item revision.

SYNTAX

DOCMGT-render-document-revision -existing_file=[replace | preserve]

ARGUMENTS

-existing_file

• replace

Replaces the existing (visualization) dataset with the new (translated) dataset.

• preserve

Translates the existing dataset provided the IRDC specified visualization dataset is not associated with the item revision. If the visualization dataset is already associated with the item revision, the new visualization file does not replace the old visualization file.

PLACEMENT

Requires no specific placement.

Do not place on the perform action of the perform-signoffs task; otherwise, this handler is executed multiple times.

RESTRICTIONS

• Requires Dispatcher for dataset translation.

• Item revision with attached datasets like Microsoft Word, Microsoft Excel, and so on, must be included as targets of the workflow process.
**DPV-export-device-to-ai**

**DESCRIPTION**
Exports the device (and station) selected from the bill of resource (BOR) in Manufacturing Process Planner to an application interface object (**AIOBJECT**). This is used for exporting Dimensional Planning and Validation (DPV) devices to application interface objects that are then downloaded by Extract, Translate, and Load (ETL).

**SYNTAX**

```
DPV-export-device-to-ai -type=ai-type -RevisionRule=revision-rule
```

**ARGUMENTS**

- `-type`
  Sets the application interface (AI) type to use to export the selected device (and station) objects.

- `-RevisionRule`
  Sets the revision rule to use when exporting the device (and station) objects.

**PLACEMENT**

This action handler can be configured in a DPV workflow task and must be placed on the **Complete** action of the specified task.

**RESTRICTIONS**

None.

**EXAMPLES**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-type</code></td>
<td>DPV_AIType</td>
</tr>
<tr>
<td><code>-RevisionRule</code></td>
<td>Latest Working</td>
</tr>
</tbody>
</table>
DPV-export-plant-to-ai

**DESCRIPTION**
Exports the plant selected from the bill of process (BOP) in Manufacturing Process Planner to an application interface object (AIObject). This is used for exporting Dimensional Planning and Validation (DPV) plants to application interface objects that are then downloaded by Extract, Translate, and Load (ETL).

**SYNTAX**
```
DPV-export-plant-to-ai -type=plant-ai-type -RevisionRule=revision-rule
```

**ARGUMENTS**
- **-type**
  Sets the application interface (AI) type to use to export the selected plant objects.
- **-RevisionRule**
  Sets the revision rule to use when exporting the device plant objects.

**PLACEMENT**
This action handler can be configured in a DPV workflow task and must be placed on the Complete action of the specified task.

**RESTRICTIONS**
None.

**EXAMPLES**
<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>DPV_PlantAIType</td>
</tr>
<tr>
<td>-RevisionRule</td>
<td>Latest Working</td>
</tr>
</tbody>
</table>
DPV-export-routine-to-ai

DESCRIPTION
Exports the routine selected from the bill of process (BOP) in Manufacturing Process Planner to an application interface object (AIObject). This is used for exporting Dimensional Planning and Validation (DPV) routines to application interface objects that are then downloaded by Extract, Translate, and Load (ETL).

SYNTAX
DPV-export-routine-to-ai -type=routine-ai-type -RevisionRule=revision-rule

ARGUMENTS
- **-type**
  Sets the application interface (AI) type to use to export the selected routine objects.

- **-RevisionRule**
  Sets the revision rule to use when exporting the device routine objects.

PLACEMENT
This action handler can be configured in a DPV workflow task and must be placed on the Complete action of the specified task.

RESTRICTIONS
None.

EXAMPLES
<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>DPV_AIType</td>
</tr>
<tr>
<td>-RevisionRule</td>
<td>Latest Working</td>
</tr>
</tbody>
</table>
ECM-add-affected-irs-as-target

DESCRIPTION
Attaches all affected revisions of the targeted EC revision as the target object of the EC process. You must add the EPM-attach-item-revision-targets handler after this handler to attach BOM view revisions (and other specification attachments) of affected revisions as target objects of the EC process.

Note
- Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.
- If you want a new process initiated for each affected revisions of the targeted EC revisions, use the ECM-start-new-sub-processes handler.

SYNTAX
ECM-add-affected-irs-as-target

ARGUMENTS
None.

PLACEMENT
May be placed multiple times in a process, depending on the requirements of the EC process. If the process allows the user to add affected revisions, this handler must be placed accordingly to ensure newly added revisions are attached as target objects of the EC process.

RESTRICTIONS
- Revisions of Affected/Solution item revisions are expected to be in only one process at a time; this handler does not add a new status when either an Affected or Solution item revision is selected as a target attachment.
- Do not use this handler with non-EC processes.
**ECM-att-new-status-for-aff-revs**

**DESCRIPTION**
Attaches a separate release status object for each affected revision of the targeted EC revision. Based on the design of the workflow, all targets of a process share the same release status object, and therefore, share the same effectivity.

This handler detaches the shared released status by creating and attaching a new status of the same name to each of the affected revisions. However, if the EC Type is defined so that effectivity is shared among affected revisions, this handler does not take effect.

You cannot input different effectivity for affected revisions if the affected revisions are being released as part of the EC process using the **ECM-add-affected-irs-as-target** handler.

**Note**
- Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.
- Effectivity is not copied from the shared status to the new status.

**SYNTAX**
**ECM-att-new-status-for-aff-revs**

**ARGUMENTS**
None.

**PLACEMENT**
Place after a released status has been added to the target objects of the EC process.

**RESTRICTIONS**
Do not use this handler with non-EC processes.
ECM-attach-components-to-change

DESCRIPTION
Attaches all unreleased logical components (for example, GDE lines, connections, and signal objects) of an assembly to a change object or as a target to the workflow process.

Note
Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

SYNTAX
ECM-attach-components-to-change -include_types=object-type1, [object-type2, ...] -target_folder=change-folder

ARGUMENTS
-include_types
Defines the types to be included as targets or reference. These can be GDE lines, connections, or signal objects. You can specify more than one type.

-target_folder
Specifies the folder where the attachments are stored.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
Do not use this handler with non-EC processes.

EXAMPLES
This example adds items of the PSConnectionRevision and Terminal types to the Solution Items folder of the target EC revision.

1. Create a CR change and add an assembly with a PSConnectionRevision and Terminal object to the Solution Items folder of the change revision.

2. Create a change process template with one Do task and one Add Status task and name it Attach to Change Test.

3. Add the ECM-attach-components-to-change handler to the Start action in the root task with the following arguments:
   -include_types=PSConnectionRevision, Terminal
   -target_folder=solution_items

4. Select the assembly and start a new workflow process using the Attach to Change Test template.

   You see the PSConnectionRevision and Terminal items added to the Solution Items folder in the change revision.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-include_types</td>
<td>PSConnectionRevision, Terminal</td>
</tr>
<tr>
<td>-target_folder</td>
<td>solution_items</td>
</tr>
</tbody>
</table>
ECM-copy-end-item-effectivity

DESCRIPTION
Copies effectivity from one affected revision of the target engineering change (EC) revision to all other affected revisions that do not share the same release status. Use this handler when various affected revisions each have a separate status object, yet need to share the same effectivity. You only need to set the effectivity in one affected revision, then use this handler to copy the effectivity to the release statuses of other revisions. If more than one affected revision contains effectivity information, this handler does not take effect.

Note
Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

SYNTAX
ECM-copy-end-item-effectivity [-set_from_prop=property-name | relation-type-name.secondary-object-name.property-name]

ARGUMENTS
-set_from_prop
If not used, end item effectivity is copied from one affected revision to all other affected revisions.

If used, the value must be the property name of the EC revision or the property of the form attached to the EC revision with a GRM relation. This value should be a real name of the property.

This argument takes the values in two valid formats:

• property-name: The property name of the EC revision.

• relation-type-name.secondary-object-name.property-name: The property name from the object that is attached to the target EC revision.

PLACEMENT
Place after a released status has been added to target objects of the EC process.

RESTRICTIONS
Do not use this handler with non-EC processes.

EXAMPLES
This example sets the effectivity start date on the latest release status of all the affected items from the value of the eff_date property of the target EC revision.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-set_from_prop</td>
<td>eff_date</td>
</tr>
</tbody>
</table>
Appendix A  Workflow handlers

ECM-create-base-revrule-form

DESCRIPTION
Displays the CM Base Configuration form and attaches it to the change revision using the relation type specified in the ECM_form_relation site preference.

Note
- Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.
- The CM Base Configuration form must be manually created before this handler can be used.

SYNTAX
ECM-create-base-revrule-form

ARGUMENTS
None.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
This handler has been specially developed for the Change Viewer application. Do not use this handler with non-EC processes.
ECM-notify-competing-changes

DESCRIPTION
Notifies a site-defined list of recipients about competing changes for each affected revision. The recipient list can be made dynamic by using four special keywords for the recipient argument. A revision is considered to be going through competing change if that revision is an affected revision for a non-released engineering change (EC) revision.

The body of the e-mail is different for each competing change found. If no subject argument is given, the e-mail is sent with the default subject: Competing change is being released.

If no recipient list is supplied, the e-mail is sent to the owner of the competing EC.

Note
Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

SYNTAX
ECM-notify-competing-changes -subject=subject -recipient=
[User:user-id | OS:OS-user | AliasList:name |
Custom:$EC_OWNER|$COMPETING_EC_OWNER|
$REV_OWNER|$COMPETING_REV_OWNER] ]

ARGUMENTS
-subject
Subject of the e-mail to be sent to all the recipients

-recipient
• User:user-id
  A Teamcenter user ID.
  user is a single valid Teamcenter user ID.

• OS:OS-user
  An operating system user ID.

• AliasList:name
  A Teamcenter alias list.

• Custom:$EC_OWNER
  The owner of the engineering change (EC) revision being released.

• Custom:$COMPETING_EC_OWNER
  The owner of the competing EC revision.

• Custom:$REV_OWNER
  The owner of the affected revision being released.

• Custom:$COMPETING_REV_OWNER
Appendix A  

Workflow handlers

The owner of the competing affected revision.

**PLACEMENT**

Place on either the last task in the EC process, or in the **Complete** action of the root task.

**RESTRICTIONS**

Do not use this handler with non-EC processes.

**EXAMPLES**

This example sends e-mail about competing changes to user **Jim**, to all members of the **Vendors** alias list, and to the owner of the competing EC revision. The e-mail uses the default subject.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-recipient</td>
<td>User:Jim, AliasList:Vendors,</td>
</tr>
<tr>
<td></td>
<td>Custom:$COMPETING_EC_OWNER</td>
</tr>
</tbody>
</table>

The following is a sample e-mail generated by the **ECM-notify-competing-changes** handler:

A100/D is being released with EC revision EC100/A.  
The Problem Revision is A100/B. The EC is owned by name.  
The revision is owned by name.  
A competing change is found where A100/E is an affected revision of the open  
EC Rev EC200/A. The Problem Revision is A100/C.  
The EC and the revision are owned by name.  
Based on the above design, the e-mail message with default arguments displays as follows:  
To: name  
Subject:Competing change is being released  
BODY  
The competing change of A100/E is being released using EC Rev EC100/A.  
Revision being released: A100/D  
EC used: EC100/A  
EC owner: name  
Details of the change still open:  
Revision: A100/E  
EC used: EC200/A  
EC owner: name
ECM-set-base-revrule

DESCRIPTION
Sets a revision rule when a change type requires a revision rule other than the default. By default, the site revision rule specified in the TC_config_rule_name site preference is used as the BOM tracking revision. Use this handler to attach a different revision rule to the target engineering change (EC) revision using the Teamcenter ECM_base_revrule_relation relation.

Note
- Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.
- Before using this handler, you must create a revision rule in the Structure Manager application.

SYNTAX
ECM-set-base-revrule -r=rule-name

ARGUMENTS
-r
Specify the base revision rule to be used for comparing the Affected and Problem assemblies associated with the target change revision.

PLACEMENT
Place on the Start action of the root task. There is no need to use this handler if the default revision rule is used for the change process.

RESTRICTIONS
Do not use this handler with non-EC processes.
ECM-start-new-sub-processes

DESCRIPTION

Starts a new process for all affected revisions of the targeted engineering change (EC) revision. The name of the process is generated from ID-Rev-Name information of the target item revision. If the generated process name exceeds the maximum length of 32 characters, the name is truncated to maximum length.

Note

- Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.
- If you want the affected revisions of the targeted EC revisions added to the existing process, use the ECM-add-affected-irs-as-target handler.

SYNTAX

ECM-start-new-sub-processes process-template-name

ARGUMENTS

process-template-name

The process template name that is used to start a new process for each affected revision of the target EC revision.

PLACEMENT

May be used several times in a process, depending on the requirements of the EC process. If the process allows the user to add affected revisions, this handler must be placed accordingly to ensure that new processes are created.

RESTRICTIONS

- Revisions of Affected/Solution item revisions are expected to be in only one process at a time; this handler does not start a new process when either an Affected or Solution item revision is selected as a target attachment.
- Do not use this handler with non-EC processes.

EXAMPLES

This example shows how to start a new process for each affected revision of a target EC revision.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Release</td>
<td></td>
</tr>
</tbody>
</table>
EPM-attach-assembly-components

DESCRIPTION
Attaches all the components of the target assembly as the targets of the same workflow process. This handler is intended for use only with item revisions.

When a workflow process is initiated for an item revision, this handler derives the components of the targeted item revision by traversing item revisions attached BOM.

By default, the handler traverses only one level deep. Set the -depth argument to all to traverse all levels. In this case, if any of the derived objects are subassemblies, they are also traversed and their component item revisions are also added as targets to the workflow process. If any remote item revisions are encountered, a warning is displayed and the remote item revisions are attached as references to the workflow process.

By default, all component item revisions currently in workflow process are ignored. If the EPM_multiple_processes_targets site preference is set to ON, you can use the -include_in_process_targets argument to attach components that are currently in workflow process.

Note
If the target item revision contains attachments such as BOM view revisions, datasets should be released along with the assembly, the EPM-attach-related-objects handler should be used in conjunction with this handler.

SYNTAX

ARGUMENTS
-depth
Defines the depth to which the traversal should take place. Specify 1 to traverse one level deep. Specify all to traverse all levels.

If not specified, traverses one level deep.

-owned_by_initiator
Adds all the component item revisions owned by the initiator as targets to the workflow process.

-owned_by_initiator_group
Adds all the component item revisions owned by the initiator's group as targets to the workflow process.

-initiator_has_write_prev
Adds all the component item revisions to which the initiator has write access as targets to the workflow process.
-exclude_released [-traverse_released_component]
Excludes released component item revisions from being added as targets. If the released component is a subassembly, the handler does not traverse the components of the released component unless traverse_released_component is also specified. The traverse_released_component argument can only be used in conjunction with the exclude_released argument.

The -traverse_released_component argument can only be used in conjunction with the -exclude_released argument.

If the -traverse_released_component is used, the handler traverses the structure of the released component, and adds the components as targets to the workflow process.

If the -depth argument is set to 1, -traverse_released_component only traverses one level deep.

If the -depth argument is set to all, the -traverse_released_component traverses all levels of the subassembly.

-rev_rule
Defines the name of the revision rule to be applied for BOM traversal. If not supplied, the default revision rule is used.

-saved_var_rule
Defines the name of the saved variant rule to be applied on BOM window for BOM traversal.

-exclude_types
Defines the types to be excluded from being added as targets.

The -exclude_types and -include_types arguments are mutually exclusive. Only one of these can be specified as arguments to the handler. If both arguments are specified, an error is displayed when running a workflow process using this handler.

/include_types
Defines the types to be included as targets.

The -exclude_types and -include_types arguments are mutually exclusive. Only one of these can be specified as arguments to the handler. If both arguments are specified, an error is displayed when running workflow process using this handler.

-add_excluded_as_ref
Adds components that are not included as targets as reference to the workflow process.

/include_in_process_targets
Can be used only if the site preference EPM_multiple_processes_targets is set to ON. In this case, this argument attaches components that are currently in process as targets.

PLACEMENT
Can place on any action. Typically placed on the Start action of the root task so that the initial list is expanded at the start of the workflow process.

RESTRICTIONS
Do not place the disallow_adding_targets handler before this handler or it fails. The disallow_adding_targets handler can be used after the placement of this handler.
EXAMPLES

- This example releases an assembly when only one level of traversal is required. Only the components of the top-level assembly are released, not the components of any subassemblies:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-depth</td>
<td>1</td>
</tr>
</tbody>
</table>

- This example releases an assembly using a specific revision rule and a saved variant rule. For this example, the **Working** revision rule and the **GMC 300 Rule** variant rule are used:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rev_rule</td>
<td>Working</td>
</tr>
<tr>
<td>-saved_var_rule</td>
<td>GMC 300 Rule</td>
</tr>
</tbody>
</table>

- This example releases an assembly using the default revision rule and the default saved variant rule, releasing only the components owned by the workflow process initiator:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-owned_by_initiator</td>
<td></td>
</tr>
</tbody>
</table>

- This example releases an assembly using the default revision rule and the default saved variant rule, releasing only the components to which the workflow process initiator belongs:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-owned_by_initiator_group</td>
<td></td>
</tr>
</tbody>
</table>

- This example releases an assembly using the default revision rule and the default saved variant rule, releasing only the components to which the workflow process initiator has write access:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-initiator_has_write_prev</td>
<td></td>
</tr>
</tbody>
</table>

- This example releases an assembly, including all components traversed to all depths, using the **Latest Released** revision rule, excluding released components from the assembly but attaching them as references:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-depth</td>
<td>all</td>
</tr>
<tr>
<td>-rev_rule</td>
<td>Latest Released</td>
</tr>
<tr>
<td>-exclude_released</td>
<td></td>
</tr>
<tr>
<td>-add_excluded_as_ref</td>
<td></td>
</tr>
</tbody>
</table>
This example releases an assembly, including all components traversed to all depths using the Latest Released revision rule, excluding released components from the assembly but attaching them as references, yet traversing the excluded released components to all depths for subcomponents to be added as targets:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-depth</td>
<td>all</td>
</tr>
<tr>
<td>-rev_rule</td>
<td>Latest Released</td>
</tr>
<tr>
<td>-exclude_released</td>
<td></td>
</tr>
<tr>
<td>-traverse_released_component</td>
<td></td>
</tr>
<tr>
<td>-add_excluded_as_ref</td>
<td></td>
</tr>
</tbody>
</table>

In this example, consider an assembly containing these revisions: CORP_Part, CORP_Tool, CORP_Vehicle, CORP_Product, CORP_Analysis, CORP_Proc_Plan, CORP_Facility, and CORP_Build.

To release the top-level assembly, excluding all the CORP_Build revisions, define the arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-exclude_types</td>
<td>CORP_Build</td>
</tr>
</tbody>
</table>

In this example, consider an assembly containing the revisions: CORP_Part, CORP_Tool, CORP_Vehicle, CORP_Product, CORP_Analysis, CORP_Proc_Plan, CORP_Facility, and CORP_Build.

To release the top-level assembly, including only the CORP_Build revisions, define the arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-include_types</td>
<td>CORP_Build</td>
</tr>
</tbody>
</table>

This example releases an assembly containing targets already in process. This argument can only be used if the EPM_multiple_processes_targets site preferences is set to ON.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-include_in_process_targets</td>
<td></td>
</tr>
</tbody>
</table>

This example releases an assembly, including all components traversed to all depths using the Latest Released revision rule, excluding released components from the assembly but attaching them as references, yet traversing the excluded released components to all depths for subcomponents to be added as targets, and all CORP_Build item revisions must be excluded:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-depth</td>
<td>all</td>
</tr>
</tbody>
</table>
This handler attaches component item revisions of the assembly to the workflow process. Therefore, you should not place the **disallow-adding-targets** handler before this handler.

Care should be taken when using this handler in conjunction with the **EPM-check-status-progression** and **EPM-check-assembly-status-progression** handlers; possible placement conflicts could arise, including:

- If you place the above rule handlers in a **Task** action ahead of this handler, there is a possibility that the assembly may never be released, as some business rules may fail, and the rule handlers may return an **EPM_nogo**.

- If you place this handler in a **Task** action ahead of the above rule handlers, there is a possibility that the assembly may be released, but may not follow the business rules. For example, the assembly may have a status which may not follow the progression path.

Teamcenter provides another method of releasing an entire assembly. You can use the **Advanced Paste** button to compile a list of objects to be pasted into the assembly. These objects can be appended to the list from multiple sources, including query results, active rich client applications, and BOM views.
**EPM-attach-item-revision-targets**

**DESCRIPTION**
Obsolete. Use the **EPM-attach-related-objects** handler instead.
Attaches all objects with specification relation to the item revision as target objects. BOM view revisions are also attached as targets to the workflow process.

**SYNTAX**

```
EPM-attach-item-revision-targets
```

**ARGUMENTS**
None.

**PLACEMENT**
Place on the **Start** action of the root task.

**RESTRICTIONS**
Place on the **Start** action of the root task so the list of target attachments is updated at workflow process initiation.

**ADDITIONAL INFORMATION**
With the addition of the **EPM-attach-related-objects** handler, the **EPM-attach-item-revision-targets** handler is obsolete.

As the **EPM-attach-item-revision-target** handler attaches BOM view revisions and objects with **IMAN_specification** relation, this handler can be replaced by adding **EPM-attach-related-objects** two times (one for specification relation and one for BOM view revisions) with the syntax:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>PSBOMViewRevision</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>
EPM-attach-related-objects

DESCRIPTION

Attaches the specified related objects of the target objects as target/reference attachments to the workflow process. This handler searches all target objects, finds the secondary objects with the specified relation or in the specified reference property and type (if specified), then adds them as target/reference attachments.

Note

If the secondary object is already part of the target list, it is ignored.

The advantage of this handler over the EPM-attach-item-revision-targets handler is that the latter can be used to attach objects (as target attachments only but not as reference attachments) only with specification relation, BOM view revisions and is useful only in the case where item revisions are targets. This handler is more generic and can also be used to attach objects with customized relations.

Note

Enable debugging functionality for this handler with the TC_HANDLERS_DEBUG environment variable.

For more information about implementing this environment variable, see the Preferences and Environment Variables Reference.

SYNTAX

EPM-attach-related-objects [-relation=relation-name | -property=property-name]
[-type=object-type1[,object-type2,...] | | -exclude_type=object-type1[,object-type2,...]] | [-lov=lov-name]
-att_type=attachment-type[-status_allow=status1
[,null,status2,...] | * | all | any | null | none] [-status_disallow=status1
[,null,status2,...] | * | all | any | null | none]

ARGUMENTS

-relation=relation-name | -property=property-name

Specifies whether a relation or property is used to locate secondary objects. Specifies the relation of the objects to be attached to the target object. It must be a valid relation.

- For manifestation, use IMAN_manifestation.
- For specification, use IMAN_specification.
- For requirement, use IMAN_requirement.
- For reference, use IMAN_reference.
- For BOM views, use PSBOMViewRevision.

-type=object-type1[,object-type2]

Specifies object types to be attached.
Appendix A  Workflow handlers

They must be valid object types. This argument is optional.
This argument should not be used with the -exclude_type argument.

-exclude_type=object-type1[,object-type2]
Specifies object types to be excluded.
They must be valid object types. This argument is optional.
This argument should not be used with the -type argument.

-lov=lov-name
Specifies an LOV to use to define which objects to attach.
Use only with the -att_type, -status_allow and -status_disallow arguments.
This argument is mutually exclusive of the -relation, -type, and -exclude_type arguments.
For an overview of using LOVs in handlers, see Using lists of values (LOVs) as handler arguments. See the LOV section for the required LOV format.

-att_type=attachment-type
Attachment type with which the objects are attached to the workflow process (target/reference).

-status_allow=status1[,null,status2,...] | * | all | any | null | none
Defines allowed statuses. Only objects with a release status matching a status defined in the list are attached.

null | NULL | none | NONE matches no status (or WIP).
* | all | ALL | any | ANY matches any status set, excluding null.

-status_disallow=status1[,null,status2,...] | * | all | any | null | none
Defines statuses that are not allowed. Only objects with a release status not matching a status defined in the list are attached.

null | NULL | none | NONE matches no status (or WIP).
* | all | ALL | any | ANY matches any status set, excluding null.

LOV

The LOV can contain multiple optional lines containing filter options followed by multiple lines containing multilevel object paths. Each multilevel object path line can optionally have a filter option added as a second field after a tilde (~).

OPTION=value

OPTION=value

[$TARGET|REFERENCE].multi.level.object.path[- OPTION=value]

[+$TARGET|REFERENCE].multi.level.object.path[- OPTION=value]

OPTION=value

Defines a configurable option to filter object selection.

If you supply an option on an LOV line on its own, it applies to all subsequent lines containing multilevel object paths. The option does not affect any multilevel object paths listed before the option.

If you supply an option on the same line as a multiple level object path, as a second field after a tilde (~) character, it only applies to that line.
Valid values are:

- **RULE={LATEST | Rule}**
  Specifies the revision rule used to select the revision attached to the workflow process if initiated on an item. Use the keyword **LATEST** to select only the latest revision.

- **INCLUDE PARENTS=YES**
  Specifies that all objects found by traversing a multilevel path are attached to the workflow process, not just the last set of objects in a path. For example, when a multilevel path is used to first find items in a workflow process, then find revisions in the item, then find datasets in the revisions, it is only the datasets that are attached by default. Setting this argument to **YES** causes both the revisions and the datasets to be attached.

This argument reduces the number of lines required in the LOV and improves performance.

**$TARGET | $REFERENCE**
Defines the starting point from which to look for objects. Valid values are:

- **$TARGET**
  Defines the starting point as the workflow process target attachments.

- **$REFERENCE**
  Defines the starting point as the workflow process reference attachments.

**multi.level.object.path**
Defines a multilevel object path to traverse to find the required objects to attach to the workflow process.

For example, *(ItemRevision).IMAN_specification.(Dataset).*

Attaches any datasets attached to the specification relation to any revisions found.

For more examples, see the **Examples** section. For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

### PLACEMENT

Place on the **Start** action of any task. Typically placed on the **Start** action of the root task so the list of target attachments is updated at workflow process initiation.

To allow targets to be added to a workflow process containing a task on which this handler has been placed (other than the root task), verify that the **disallow-adding-targets** handler does not exist on the root task of the respective workflow process template and ensure that the affected users have change access to the workflow process object. You may use the **EPM-set-job-protection** handler to ensure that the required change access is asserted.
Appendix A  Workflow handlers

Note
If this handler is placed on the root task to attach all required targets, remove the default EPM-attach-item-revision-targets review process handler. Otherwise, objects that are not required may be attached, such as any released objects that are attached to the Specification relation in any target revisions.

Alternatively, use the default EPM-attach-item-revision-targets handler to attach all Specification relations and all BOM view revisions and use the EPM-attach-related-objects handler to attach objects from other relations.

RESTRICTIONS
• Requires one or more target objects to find the related objects. Placement should allow at least one target object before the execution of this handler takes place.

• To attach both targets and references using LOVs requires two occurrences of the handler: one to attach the targets by setting the -att_type argument to target, and one to attach the references using the -att_type argument to reference.

• The LOV argument cannot be used to attach objects based on properties.

EXAMPLES
• This example attaches all objects with a specification relation as target objects to the workflow process, when a workflow process is initiated on an item revision:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

Note
If an object is already attached as target, it is not added.

• This example attaches all objects with a specified property as target objects to the workflow process, when a workflow process is initiated on an item revision:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-property</td>
<td>altid_list</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

Note
If an object is already attached as target, it is not added.

• To attach all objects with a reference relation as reference objects, add this handler one more time with the syntax:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_reference</td>
</tr>
<tr>
<td>-att_type</td>
<td>reference</td>
</tr>
</tbody>
</table>

• This example attaches the BOM view revision type View as a target:
Alternatively, you can use these LOV settings:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>PSBOMViewRevision</td>
</tr>
<tr>
<td>-type</td>
<td>view</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

where the SYS_EPM_attach_view_bvr LOV contains the data:

```
# LOV SYS_EPM_attach_view_bvr

$TARGET.(ItemRevision).PSBOMViewRevision.BOMView Revision
```

- This example attaches the **UGMASTER** and the **UGPART** datasets (associated by the **IMAN_specification** relation to the item revision) to the item revision as target objects.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-type</td>
<td>UGMASTER, UGPART</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

Alternatively, you can use these LOV settings:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM_attach_UGMASTER_UGPART</td>
</tr>
</tbody>
</table>

where the SYS_EPM_attach_UGMASTER_UGPART LOV contains the data:

```
$TARGET.(ItemRevision).IMAN_specification.UGMASTER,UGPART
```

- This example uses the **-exclude_type** argument to specify object types that are not to be attached as targets to the workflow process. It attaches all objects attached to the **Specification** relation in any target revisions as targets to the workflow process, except for the dataset types **UGMASTER** and **Text**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-exclude_type</td>
<td>UGMASTER, Text</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

Alternatively, you can use these LOV settings:
### Appendix A  Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM.exclude_UGMASTER</td>
</tr>
</tbody>
</table>

where the `SYS_EPM_exclude_UGMASTER` LOV contains the data: 
# Use an * for any class, then exclude UGM
tER and Text:

```
$TARGET.(ItemRevision).IMAN_specification.(*)!UGMASTER!Text
```

- This example attaches all specification objects, all BOM view revisions, all forms attached to datasets through a **Form** reference (except **UGPartAttr** forms), and all forms attached through a **manifestation** relation. Only attach objects that not released.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM_attach_main_objects</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
<tr>
<td>-status_allow</td>
<td>null</td>
</tr>
</tbody>
</table>

where the `SYS_EPM_attach_main_objects` LOV contains the data:

```
#=========================================================
# Attach all objects in target revision Specification relation
#=========================================================
$TARGET.(ItemRevision).IMAN_specification.*
```

```
#=========================================================
# Attach all forms attached to datasets in target revision
# Specification relation as a Form reference, but excluding the
# form type UGPartAttr.
#=========================================================
$TARGET.(ItemRevision).IMAN_specification.UGMASTER.UGPART-ATTR.UGPartAttr
```

```
# Attach all BOM View Revisions in target revision
#=========================================================
$TARGET.(ItemRevision).PSBOMViewRevision.*
```

```
# Attach all forms in target revision Manifestation relation
#=========================================================
$TARGET.(ItemRevision).Manifestation.(Form)
```

- This example attaches all required revision attachments, such as specification objects and BOM view revisions, regardless of whether the workflow process is initiated on revisions, items or folders containing the items or revisions. If the method of initiating workflow processes on items or folders is convenient, use the **EPM-remove-objects** handler to remove the items and/or folders from the targets after this handler.

When the targets are item revisions, attach all specification objects, all BOM view revisions and any objects attached to specification datasets as relations and references (only attaches workspace objects).
When the targets are items, attach all of the latest revisions and all objects mentioned above for each revision.

When the targets are folders, attach any items in the folders and the item revisions and the revision attachments. For any revisions in the folder, attach the revisions' attachments.

Only attach objects not already released or with a status of Pending.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM_attach_main_objects</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
<tr>
<td>-status_allow</td>
<td>null, Pending</td>
</tr>
</tbody>
</table>

where the SYS_EPM_attach_main_objects LOV contains the data:

```
#=========================================================
# Set options for all lines to include all objects found and to set
# the revision rule for any items
 INCLUDE PARENTS = YES
 RULE = LATEST
#=========================================================
# Attach required objects from REVISION targets
#---------------------------------------------------------
$TARGET.(ItemRevision).IMAN_specification, PSBOMViewRevision.*.* ~
#---------------------------------------------------------
# Attach required objects from latest revisions in ITEM targets
#---------------------------------------------------------
$TARGET.(Item).Revisions.*.IMAN_specification, PSBOMViewRevision.*.*
#=========================================================
# Attach required objects from FOLDER targets
#---------------------------------------------------------
$TARGETS.(Folder).*.(Item).Revisions.*.IMAN_specification, PSBOMViewRevision.*.*
$TARGETS.(Folder).*.(ItemRevision).IMAN_specification, PSBOMViewRevision.*.*
```

ADDITIONAL INFORMATION

With the addition of this handler, these handlers are obsolete:

**EPM-attach-item-revision-target**

As the EPM-attach-item-revision-target handler attaches BOM view revisions and objects with IMAN_specification relation, this handler can be replaced using one of the following options:

- Adding the EPM-attach-related-objects handler two times (one for specification relation and one for BOM view revisions) with the syntax:

  **EPM-attach-related-objects**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

  **EPM-attach-related-objects**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>PSBOMViewRevision</td>
</tr>
</tbody>
</table>
### Appendix A  Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

- Adding the **EPM-attach-related-objects** handler once using an LOV:

**EPM-attach-related-objects**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM_attach_default_objects</td>
</tr>
<tr>
<td>-att_type</td>
<td>target</td>
</tr>
</tbody>
</table>

where the **SYS_EPM_attach_main_objects** LOV contains the data:

```plaintext
$TARGET . (ItemRevision) . Specification, PSBOMViewRevision . *
```
**EPM-auto-check-in-out**

**DESCRIPTION**
Automatically checks in/out the target objects of a workflow process to the assigned reviewer or the responsible party. This prevents other users who have write access to the target objects from being able to modify them. Optionally, when a dataset is checked in/out, it checks in/out the BOM view of the type specified.

**SYNTAX**

```
EPM-auto-check-in-out
-user=reviewer | responsible_party
-action=check-in | check-out
[-include_type=dataset-type::bom-view-type]
```

**ARGUMENTS**

- `user=reviewer` | `responsible_party`
  Use `reviewer` for Review tasks and `responsible_party` otherwise.

- `action=check-in` | `check-out`
  Action to check in or check out the objects.

- `include_type=dataset-type::bom-view-type`
  Also check in/out the type specified. This value works for BOM views only. A BOM view of the specified type is checked in/out if a dataset of a specified type is checked in/out.

  This argument is optional.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
Placement of the `EPM-auto-check-in-out` handler with the `action=check-out` defined should be determined considering the placement of `CR-assert-targets-checked-in` rule handler, which displays an error if target objects are not checked in. If this handler is used in a Review task, this should be used only when the number of reviewers equals one.

**EXAMPLES**
This example, placed on a Review task, checks out the objects to the reviewer once the task is assigned to the reviewer and checks in the objects once the reviewer signs off. You can place this action handler in the Complete action of the `select-signoff-team` subtask using the Check out action, and in the Complete action of the `perform-signoffs` subtask using the Check in action.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-user</code></td>
<td>reviewer</td>
</tr>
<tr>
<td><code>-action</code></td>
<td>check-out</td>
</tr>
<tr>
<td><code>-include_type</code></td>
<td>UGMASTER::view</td>
</tr>
</tbody>
</table>

This setting checks out all the target objects; if a UGMASTER is checked out, the BOM view of type view is also checked out. If UGMASTER is referenced in multiple item revisions, the BOM view of the first item revision is checked out.
EPM-change-ownership

DESCRIPTION
Changes the ownership of all target objects to the group and user ID of the reviewer or the responsible party.

The advantage of changing ownership is to allow a revision rule to configure WIP (work in process) data based on owner and group settings.

If this handler is used in Review tasks, the number of reviewers should be one.

To save processing time and/or improve robustness, the handler can be configured to be active only in one or more actions (\texttt{-active=action}). If the handler is called as part of trigger to another action, the handler silently returns immediately.

SYNTAX
\texttt{EPM-change-ownership -owner=reviewer | responsible_party [-active=action [-active=other-action]][-depth=level] [-debug]}

ARGUMENTS
- \texttt{-owner}
  
  User to whom the ownership is given.

  Use \texttt{reviewer} if this handler is used in a Review task. Use \texttt{responsible_party} otherwise.

- \texttt{[active=action [-active=other-action]]}
  
  Name of the action for which this handler is valid.

  If this argument is used, and the handler is called as part of a trigger to an unlisted action, the handler silently returns immediately. For more information about valid action names, see the \texttt{-action} argument.

  This argument can be useful when the handler is placed on the \texttt{Perform} action. These actions automatically execute the following \texttt{Perform} action handlers, raising the potential for unnecessary processing:

  - \texttt{EPM_add_attachment_action}
  - \texttt{EPM_remove_attachment_action}
  - \texttt{EPM_approve_action}
  - \texttt{EPM_reject_action}
  - \texttt{EPM_promote_action}
  - \texttt{EPM_demote_action}
  - \texttt{EPM_refuse_action}
  - \texttt{EPM_assign_approver_action}
  - \texttt{EPM_notify_action}

  This argument is optional.
-depth
Recursion depth. This argument is optional and the default is set to 1.

-debug
Debug mode.

Debug messages are written to the error stack and displayed in the rich client interface, as well in the log file.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
Set the number of reviewers to 1 when this handler is placed on a Review task.

EXAMPLES
This example, when placed on the Complete action of the select-signoff-team subtask of a Review task, changes the ownership of all the target objects to reviewers and their groups.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-owner</td>
<td>reviewer</td>
</tr>
</tbody>
</table>
EPM-check-signoff-comments

DESCRIPTION
Requires users to type a comment when making a signoff decision. You can specify whether the comment is required for the approve decision or the reject decision. If neither decision is specified, comments are required to complete either signoff decision.

SYNTAX
EPM-check-signoff-comments [-decision=decision-type]

ARGUMENTS
- **-decision**
  Specifies which signoff decision requires comments to be entered when making a signoff decision for either a Review task or an Acknowledge task.

  Use APPROVE to require comments to be added before selecting Approve for a Review task, or Acknowledge for an Acknowledge task.

  Use REJECT to require comments to be added before rejecting a signoff for a Review task.

  If this argument is not used, comments are required for either decision before completing a signoff.

PLACEMENT
Place on the Perform action of the perform-signoffs task.

RESTRICTIONS
Place on the perform-signoffs task.

EXAMPLES
- This example requires that the user type comments before rejecting a signoff:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-decision</td>
<td>REJECT</td>
</tr>
</tbody>
</table>

- This example requires the user to type comments before approving a signoff:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-decision</td>
<td>APPROVE</td>
</tr>
</tbody>
</table>
EPM-create-form

DESCRIPTION

Creates an instance of a specified form and attaches that form to the specified task. For more information, see EPM-display-form.

Configuring a task to display forms using EPM-display-form, EPM-hold, and EPM-create-form

To configure a task to display a form when a user performs a specified action, use the EPM-hold handler. This handler pauses the task, requiring the user to perform an action on the task before the task can complete. Without the use of this handler, a task completes automatically once started.

To create an instance of a specified form and attach the form to the specified task, use the EPM-create-form handler.

Therefore, the EPM-create-form handler creates the form when the Start action is initiated, the EPM-display-form handler displays the form when the Perform action is initiated, and the EPM-hold handler prevents the task from automatically completing, allowing the form to be completed by the user.

Variations on the above example may be required for a more sophisticated interaction when it is required that the task not complete until required fields are entered in the form. This type of configuration requires the creation of customized rule handlers.

SYNTAX

EPM-create-form -type=formtype [-name=string] [-description=string] [-default=Field-Name.Value] [-location=task-name.attachment-type]

ARGUMENTS

-type
Valid FormType object.

-name
User-defined form name. Default is the workflow process name.

-description
User-defined description of the form. Default value is null.

-default
Field-Name.Value is the name-value pair for a particular field of the form. Users can choose to set the default value to more than one field by setting the -default argument for each field. Do not use Field-Name.Value for field names of Typed_Reference and Untyped_Reference types. This argument is optional.

Note

Use this argument to populate the initial values in forms created by a workflow. If you do not use this argument and instead set the initial value in the business object definition, the workflow process defines the value as empty until you perform an edit and save it.

-location
Task name receiving the new form as an attachment. The default value is the current task.
attachment-type
Accepts one of four reserved keywords:

- **$REFERENCE**
  Reference attachments

- **$TARGET**
  Target object attachments

- **$SIGNOFF**
  Signoff attachments

- **$RELEASE_STATUS**
  Release status attachments

The default value is **$REFERENCE**.

**PLACEMENT**

Requires no specific placement.

**RESTRICTIONS**

None.

**EXAMPLES**

- This example shows how to create form type **ECN Form**, form name **ECN**, form description **Engineering Change Management Form**, and attachment type **EPM_reference** attachment. The form is attached to the current task of the workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>ECN Form</td>
</tr>
<tr>
<td>-name</td>
<td>ECN</td>
</tr>
<tr>
<td>-description</td>
<td>Engineering Change Management Form</td>
</tr>
<tr>
<td>-location</td>
<td>$REFERENCE</td>
</tr>
</tbody>
</table>

- This example attaches the form as a target attachment to the current task:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-location</td>
<td>$TARGET</td>
</tr>
</tbody>
</table>

To attach the form as a reference attachment to the current task, do not set the **-location** argument, because this is the default location this handler uses when this argument is not defined.
EPM-create-relation

**DESCRIPTION**
Creates a specified relation between the target/reference objects of the workflow process. The relation to be created must be a valid relation. The handler goes through all the primary objects of the specified type and creates a specified relation with all the secondary objects of the specified type.

**SYNTAX**

```
EPM-create-relation -relation=relation-name -primary= target | reference
-primary_type=type-of-primary-object -secondary=target | reference
-secondary_type=type-of-secondary-object
```

**ARGUMENTS**

- `-relation`
  The relation type to be created.

- `-primary`
  The objects that have to be considered as primary objects (target or reference).

- `-primary_type`
  Type of object to be considered as primary object.
  Considers all the target or reference attachments of this type as primary objects. Target or reference is specified in `-primary` argument.

- `-secondary`
  The objects that have to be considered as secondary objects (target or reference).

- `-secondary_type`
  Type of object to be considered as secondary object.
  Considers all the target or reference attachments of this type as secondary objects. Target or reference is specified in `-secondary` argument.

**PLACEMENT**
Place on the **Complete** action of the task.

**RESTRICTIONS**
None.

**EXAMPLES**

In this example, the workflow process has two item revisions as target objects and one **UGPART** object as a reference object. There is no relation between the two item revisions and the **UGPART**. To create a requirements relationship between the two, with the item revisions as primary and the **UGPART** as secondary:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-relation</code></td>
<td>IMAN_requirement</td>
</tr>
<tr>
<td><code>-primary</code></td>
<td>target</td>
</tr>
<tr>
<td><code>-primary_type</code></td>
<td>ItemRevision</td>
</tr>
<tr>
<td><code>-secondary</code></td>
<td>reference</td>
</tr>
<tr>
<td><code>-secondary_type</code></td>
<td>UGPART</td>
</tr>
</tbody>
</table>
**EPM-create-sub-process**

**DESCRIPTION**

This handler is used to start subprocesses from a workflow process. The new subprocess can take on attachments of the parent process.

Creates subprocesses and attaches the specified target/reference objects of the parent process as target/reference attachments to the new subprocesses. This handler goes through all the target/reference objects of the parent process, finds the corresponding object type, and adds them as target/reference attachments of new subprocess. The advantage of this handler is that you can launch one or multiple workflow processes from within a parent process. You can use this handler to set a dependency between the parent process and subprocess in a way that causes the parent process to wait for the subprocess’s (task) completion. The action handler can be added multiple times to a task action to provide abilities such as using different workflow process templates per target object type or other combinations.

**SYNTAX**

```
EPM-create-sub-process
-template=process-template-name
[-from_attach=Target | Reference | ALL]
[-to_attach=Target | Reference | ALL]
[-include_types=object-type]
[-exclude_types=object-type]
[-process_name=name-for-process]
[-process_desc=string]
[-multiple_processes]
[-dependency=multilevel-parent-process-task-path::multilevel-sub-process-task-path]
[-transfer]
[-process_assembly]
-depth=depth-of-traversal
-rev_rule=revision-rule-to-apply
-relation=relation-type-to-look
[-include_related_types=type-of-related-components-to-be-included]
[-exclude_related_types=type-of-related-components-to-be-excluded]
```

**ARGUMENTS**

- **-template=process-template-name**
  The workflow process template name that is used to start a new workflow process. This argument is required.

- **-from_attach=Target | Reference | ALL**
  The following are the objects attachments to be inherited from the parent process target and/or reference folder:
  
  - **Target**
    Takes the attachments from the target folder of the parent process.
  
  - **Reference**
    Takes the attachments from the reference folder of the parent process
  
  - **ALL**
Takes targets and reference attachments.

The `-from_attach` and `-to_attach` arguments must be used together. If you use one argument, you must use the other.

This argument is optional.

The site preference to enable for multiple workflow processes for the same objects needs to be set if `-from_attach` is used with either the Target or ALL option. The `EPM_multiple_processes_targets` preference attaches components that are currently in process as targets if it is set to ON.

`-to_attach=Target | Reference | ALL`

The following are the objects to attach with the new workflow process:

- **Target**
  Attaches to target folder of new workflow process.

- **Reference**
  Attaches to reference folder of new workflow process

- **ALL**
  Attached from target folder of the parent process to the target folder of a new workflow process and reference folder of the parent process to the reference folder of a new process.

The `-from_attach` and `-to_attach` arguments must be used together. If you use one argument, you must use the other.

This argument is optional.

`-include_types=object-type`

Defines the types to be included as targets and/or references.

- Must be valid workspace object types. For example: `ItemRevision` and `ITEM`.
- If this argument is specified as `Dataset`, any type of dataset (`UGMASTER`, `UGPART`, `Text`, and so on) is considered.
- If this argument is specified as `ItemRevision`, any type of item revision (`DocumentRevision` and any custom item revision types) is considered.

This argument is optional. If this argument is passed to the handler, `-from_attach` and `-to_attach` should also be passed to the handler.

`-exclude_types=object-type`

Defines the types to be excluded from being adding as targets/reference.

- Must be valid workspace object types. For example: `ItemRevision` and `ITEM`
- If this argument is specified as `Dataset`, any type of dataset (`UGMASTER`, `UGPART`, `Text`, and so on) is considered.
- If this argument is specified as `ItemRevision`, any type of item revision (`DocumentRevision`, and so on, and any custom item revision types) is considered.
This argument is optional. If this argument is passed to the handler, -from_attach and -to_attach should also be passed to the handler.

-process_name=name-of-process
The name that is used to identify the new workflow process.

For example, if the parent process was called parentprocess and the targets are item1/A, item2/B and item3A:

When a workflow process name is given as subprocess and no -multiple_processes arguments are used, the workflow process name alone is used as there is only one, so subprocess would be called subprocess. In this case, if users want a number included in the name, they would put it in the argument name and they would know there is only one to be created. If the workflow process name is not given and the -multiple_process is argument is not used, the parent process name is 1; in this case, it is parentprocess:1. Same result for a case where there are no targets on the parent process.

If the workflow process name is not given, and -multiple_processes argument is used, use the subprocessstatenamecount; in this case, that would be item1/A:1, item2/B:2, item3/A:3. For the case where the parent had no targets, the name is parentprocess:1.

This argument is optional.

-process_desc=string
Workflow process description.

If the description is not specified, it is set to blank.

This argument is optional.

-multiple_processes
Each target object to be considered becomes a target in its own individual subprocess. If not specified, all targets are in a single subprocess.

To learn how to use this argument, see the example section.

This argument is optional.

-dependency=multilevel-parent-process-task-path::multilevel-sub-process-task-path
Creates a dependency between a parent process task and a specified subprocess task; the parent process’s task proceeds after the subprocess’s task completes.

You must use a multilevel path to specify the task templates. Separate path levels with colons (:). Separate the multilevel path of the parent task from the multilevel path of the subprocess task with a double colon (::). For example:


If you use a double colon (::) only without specifying either a source or target task, a subprocess task is created, and a dependency is established between the parent process task and the newly created subprocess task.

If a parent process task is not specified, the task containing this handler is designated as the parent process task. If a subprocess task is not specified, or not found, the dependency is not set.

This argument is optional.
Note

If you try to complete a task that has a dependency on an uncompleted subprocess task, you receive a warning indicating that the interprocess task dependencies are not met for the dependent task.

-transfer
Transfers attachments of the parent process to the subprocess. The parent process has no attachments as target/reference that exists in the subprocess.

-process_assembly
Signals the handler to traverse the assembly and start a subprocess on its components. Multiple workflow processes can be started if the -multiple_processes argument is specified. This argument works in conjunction with -depth, -rev_rule, -include_related_types, and -exclude_related_types arguments. This argument can be used together with the -relation argument. Both arguments can be specified on the same instance of the handler.

-depth=depth of traversal
Specifies the depth of traversal for an assembly. Specify all to traverse all levels. If not specified, the default value is 1.

-rev_rule=revision-rule-to-apply
Defines the name of the revision rule to be applied for BOM traversal. If not supplied, the default revision rule would be used.

-relation=relation-type-to-look
Finds the objects attached to the target objects with the given relation. The value must be a valid relation.

Specifies whether a relation is used to locate secondary objects. The relation of the objects to be attached to the target object. Must be a valid relation.

To specify manifestation, use IMAN_manifestation.

For specification use IMAN_specification.

For requirement use IMAN_requirement.

For reference use IMAN_reference.

For BOM views use PSBOMViewRevision.

This argument works in conjunction with -include_related_types, and -exclude_related_types arguments. This argument can be used together with the -process_assembly argument. Both arguments can be specified on the same instance of the handler.

-include_related_types=type-of-related-components-to-be-included
Defines the types of related component objects to be included as targets and/or references.

- Must be valid workspace object types. For example: ItemRevision and ITEM.

- If this argument is specified as Dataset, any type of dataset (UGMASTER, UGPART, Text, and so on) is considered.
Appendix A  Workflow handlers

- If this argument is specified as ItemRevision, any type of item revision (DocumentRevision and any custom item revision types) is considered.

This argument works in conjunction with -process_assembly and -relation arguments.
This argument is optional.

-exclude_related_types=type-of-related-components-to-be-excluded
Defines the types of related component objects to be excluded from being adding as targets and/or reference.
- Must be valid workspace object types. For example: ItemRevision and ITEM
- If this argument is specified as Dataset, any type of dataset (UGMASTER, UGPART, Text, and so on) is considered.
- If this argument is specified as ItemRevision, any type of item revision (DocumentRevision, and so on, and any custom item revision types) is considered.

This argument works in conjunction with -process_assembly and -relation arguments.
This argument is optional.

Note
The -include_related_types and -exclude_related_types arguments can be used in conjunction with each other. If used in conjunction, the -include_related_types argument takes precedence; first the objects are processed against -include_related_types, and then -exclude_related_types.

PLACEMENT
Place in the Start or Complete action of a task template.

Note
If you use the -dependency argument and the current task is dependant on the subprocess, you must place the handler on the Start action. If you place it on the Complete action, the -dependency argument causes an error.

The handler can be added multiple times to a task action to provide abilities such as using different workflow process templates per target object type or other combinations.

RESTRICTIONS
- If a user demotes a task that already created subprocesses, when the task gets activated again, it creates another subprocess. Depending on the user's choice, they should either delete the original subprocess or the new subprocess. Currently this is a manual step for the user.
- After execution, this handler displays the subprocesses under the Change Viewer application.
- The -depth and -rev_rule arguments are used only when the -process_assembly argument is used. The -exclude_related_types and
-include_related_types arguments are used only when -processassembly or -relation is used.

EXAMPLES

The following examples illustrate how to configure the handler arguments. These examples illustrate creating a parent process template containing a Do task and adding the handler to the task to create a subprocess.

- The examples where the current task is dependant on the subprocess and that use the -dependency argument must be placed on the Start action.

- The examples without the -dependency argument can be placed on either the Start or Complete action of a task.

Note

You can add this handler to any action from which you want to create the subprocess. Use the following examples to understand how to configure the handler arguments.

- This example launches a new process using the CMII WA template and sets the dependency between the parent process initiating task that starts a new subprocess and SubProcess_001. The task that initiates the new subprocess cannot be completed until SubProcess_001 is completed. Place this handler on the Start action.

  Argument | Values
  -template | CMII WA
  -dependency | ::
  -process_name | SubProcess_001

- The example creates a new workflow process using the CMII WA template with no attachments. The -process_name and -process_desc are optional.

  Argument | Values
  -template | CMII WA
  -process_name | 0006/A_CMII WA
  -process_desc | This is a demo description text

- This example creates a new workflow process on the CMII WA template by inheriting all the targets/reference attachments of the parent process as target/reference attachments, respectively, of the newly created workflow process. If the workflow process name is not defined, it generates a workflow process name for the child process in the Parentprocesscount format. The workflow process description is left blank.

  Argument | Values
  -template | CMII WA
  -from_attach | ALL
  -to_attach | ALL
• This example creates a new workflow process on the **CMII WA** template by inheriting all the target attachments of the parent process as target attachments for the subprocess.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

• This example creates a new workflow process on the **CMII WA** template by inheriting all the attachments (target and reference) of the parent process as target attachments for the subprocess.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>ALL</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

• This example launches a new workflow process on the **CMII WA** template. All target and reference attachments of the **ItemRevision** and **UGMASTER** types of the parent process are attached as targets for the new process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>ALL</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-include_types</td>
<td>ItemRevision, UGMASTER</td>
</tr>
</tbody>
</table>

• This example launches a new workflow process on the **CMII WA** template. All objects (both target and reference attachments) of the **ItemRevision** and **UGMASTER** type of the parent process are attached as target and reference attachments respectively for the new workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-include_types</td>
<td>ItemRevision, UGMASTER</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>ALL</td>
</tr>
</tbody>
</table>

• This example launches a new workflow process on the **CMII WA** template. All objects of the **ItemRevision** type of the parent process are excluded as targets for the new workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
</tbody>
</table>
• This example launches a new workflow process on the CMII WA template by specifying the -include_types and -exclude_types arguments. It specifies the list of attachment types to be included in -include_types and the list of types to be excluded in -exclude_types. This argument launches a subprocess with only ItemRevision.

• This example launches a new workflow process on the CMII WA template and sets the dependency between the DoChecklist task in the DesignReview parent process and the perform-signoffs subtask of the QA Review task of the CMII WA_001 subprocess. The DoChecklist task of the parent process cannot complete until the perform-signoffs task in the subprocess completes. Place this handler on the Start action.

• This example launches a new workflow process using the CMII WA template. Because no path is specified for the parent process, the task containing this handler is used as the parent process task. A dependency is created between the task containing this handler and the perform-signoffs subtask of the QA Review task of the CMII WA_001 subprocess. The task containing this handler cannot complete until the perform-signoffs task in the subprocess completes. Place this handler on the Start action.

• This example launches new workflow processes on the CMII WA template. Each object instance of the ItemRevision type on target attachments of the parent process launches a new workflow process with that instance as target. For
example, if the parent process has three `ItemRevision` objects as the target, three different workflow processes are launched.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>ALL</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-include_types</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-multiple_processes</td>
<td></td>
</tr>
</tbody>
</table>

- The following handler configuration looks for an assembly in the targets, configures it as per the `Latest Working` revision rule and starts multiple workflow processes on all its components.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-multiple_processes</td>
<td></td>
</tr>
<tr>
<td>-process_assembly</td>
<td></td>
</tr>
<tr>
<td>-depth</td>
<td>All</td>
</tr>
<tr>
<td>-rev_rule</td>
<td>Latest Working</td>
</tr>
</tbody>
</table>

- The following handler configuration starts a subprocess on the `UGMaster` dataset attached to the target objects with `Iman_specification` relation.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-multiple_processes</td>
<td></td>
</tr>
<tr>
<td>-relation</td>
<td>Iman_specification</td>
</tr>
<tr>
<td>-include_related_types</td>
<td>UGMaster</td>
</tr>
</tbody>
</table>

- The following handler configuration looks for an assembly in the targets, configures it as per the `Latest Working` revision rule and starts multiple workflow processes on all its components. It also starts a subprocess on the objects that are attached to the target objects with the `Iman_specification` relation.
## Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-multiple_processes</td>
<td>TARGET</td>
</tr>
<tr>
<td>-processAssembly</td>
<td>All</td>
</tr>
<tr>
<td>-rev_rule</td>
<td>Latest Working</td>
</tr>
<tr>
<td>-relation</td>
<td>Iman specification</td>
</tr>
</tbody>
</table>

- The following handler configuration starts a subprocess using the **CMII WA** template. All target objects of the **Dataset** type except for **MSWord** type objects are attached as targets to the subprocess.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-include_types</td>
<td>Dataset</td>
</tr>
<tr>
<td>-exclude_types</td>
<td>MSWord</td>
</tr>
</tbody>
</table>

### RESTRICTIONS ON ARGUMENTS

These examples show how not to use this handler.

- Do not create a workflow process without specifying the **-template** name.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-process_name</td>
<td>0006/A_CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_attach</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

- Do not create a workflow process with the **-multiple_processes** argument but not providing the **-from_attach** and **-to_attach** arguments.
### Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-multiple_processes</td>
<td></td>
</tr>
</tbody>
</table>

- Do not create a workflow process by only specifying either one of the arguments: 
  - **from_attach** or **to_attach**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-template</td>
<td>CMII WA</td>
</tr>
<tr>
<td>-from_attach</td>
<td>TARGET</td>
</tr>
</tbody>
</table>
**EPM-delete-ugcgm-markup**

**DESCRIPTION**
Attaches all the drawing sheets as a target object for a UGMASTER/UGPART dataset in the selected workflow process, so the DrawingSheet dataset also attains a release status once the workflow process is approved. If the DrawingSheet dataset names are the same as for the previous item revisions, all DirectModelMarkup datasets are deleted if the UGMASTER/UGPART dataset names are also the same as in the previous revision.

**SYNTAX**
EPM-delete-ugcgm-markup [-type=valid-dataset-type, [valid-dataset-type]]

**ARGUMENTS**
- **-type**
The valid dataset types for this handler are UGMASTER and UGPART. A user can specify more than one dataset type separated by a comma between the two dataset types. If the user does not specify any dataset type, this handler assumes UGPART as the dataset type.

**PLACEMENT**
Place on the Start action of the root task.

**RESTRICTIONS**
None.

**EXAMPLES**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>UGMASTER, UGPART</td>
</tr>
</tbody>
</table>
EPM-display-form

DESCRIPTION
Displays specified forms attached to a specified task. The default behavior is to display all attachments of the FormType object attached to the current task.

This action handler is typically attached to an instance of the task template. The task template is used to define custom forms and other site-specific tasks for the user to complete. The task template is designed to accept customization. The default design of this template contains no innate customized interface behavior. Other task templates are not meant to display a customized interface (such as the Add Status task) or may already have had customized interface behavior assigned (such as the Review and Route task templates).

An example of a task template that already has customized interface behavior assigned is the Do task template. While form handlers can be added to the Do task template, the template's original interface behavior is the default display, not the forms. If the default display required is a customized form, use an instance of the task template.

There can be instances where form handlers are used with a task template that already contains customized interface behavior. For example, perhaps the instructions for a Do task require users to complete the attached form, then mark the Do task as complete by selecting the Done check box in the Do task dialog box. Attaching forms to this task, using the EPM-create-form handler, allows the users to expand the attachments in either their worklists or the process flow pane and select the required forms, opening them with the Open action.

The default Perform action for any template can be overridden using the .properties file. It is more effective, however, to use the task template when the required default Perform action is the display of forms.

Configuring a task to display forms using EPM-display-form, EPM-hold, and EPM-create-form

To configure a task to display a form when a user performs a specified action, use the EPM-hold handler. This handler pauses the task, requiring the user to perform an action on the task before the task can complete. If this handler is not used, a task completes automatically once started.

To create an instance of a specified form and attach the form to the specified task, use the EPM-create-form handler.

The EPM-create-form handler creates the form when the Start action is initiated, the EPM-display-form handler displays the form when the Perform action is initiated, and the EPM-hold handler prevents the task from automatically completing, allowing the form to be completed by the user.

Variations on the above example may be required for a more sophisticated interaction when it is required that the task not complete until required fields are entered in the form. This type of configuration requires the creation of customized rule handlers.

SYNTAX

EPM-display-form -type=form-type [-form=task-name.attachment-type]
ARGUMENTS

- **-type**
  Valid FormType object.

- **-form**
  Form to be displayed. The default values for this optional argument are reference attachments of the FormType attached to the current task_name.

attachment-type
Accepts one of the following reserved keywords:

- **$REFERENCE**
  Reference attachments

- **$TARGET**
  Target object attachments

- **$SIGNOFF**
  Signoff attachments

- **$RELEASE_STATUS**
  Release status attachments

PLACEMENT

Requires no specific placement. Typically placed on the Perform action of a task. If this task has no other perform user interface, the form is used as its Perform action user interface.

RESTRICTIONS

None.

EXAMPLES

This example lists handler definitions to be entered on a task template to display customized forms:

- On the Start action: **EPM-create-form**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>form-type-name</td>
</tr>
<tr>
<td>-name</td>
<td>form-name</td>
</tr>
<tr>
<td>-description</td>
<td>form-description</td>
</tr>
<tr>
<td>-location</td>
<td>$ROOTTask.$REFERENCE</td>
</tr>
</tbody>
</table>

- On the Perform action: **EPM-display-form**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>form-type-name</td>
</tr>
<tr>
<td>-form</td>
<td>$ROOTTask.$REFERENCE</td>
</tr>
</tbody>
</table>

- On the Complete action: **EPM-hold**
## Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-true</td>
<td></td>
</tr>
</tbody>
</table>
EPM-export-AI-AH

DESCRIPTION
Exports the attached workflow targets to an AI object created per the type specified in the workflow handler parameters. The AI object name defaults to the name of the workflow process. A reference to the newly created AI object is attached as a workflow reference.

SYNTAX
EPM-export-AI-AH -type=AIobject_type

ARGUMENTS
-type
Defines the AI object type to attach. Must be a valid AI object type.

PLACEMENT
Place on the Complete action of any task.

RESTRICTIONS
None.
**EPM-export-to-plmxmlfile**

**DESCRIPTION**
Exports targets, references, and/or workflow process information to a PLM XML file. Use this handler to export targets and references data to a PLM XML file during a workflow process. You can also export operation and plant objects or the state of the workflow tasks to the PLM XML file. See *Workflow task actions and states* for more information.

**SYNTAX**
```
EPM-export-to-plmxmlfile [-context=context-string]
[-attach={target|reference|both}] [-file=filename]
[-include_process_info] [-revrule]
```

**ARGUMENTS**
- **-context**
  Defines the context string, which specifies the transfer mode used for export. If not specified, it uses the default transfer mode.

- **-attach**
  Specifies which workflow process attachments are exported. If not specified, only targets are exported.

- **-file**
  Specifies the path and file name to which the data is exported. If the path is not specified, the file is placed in the TC_TMP_DIR directory. If this argument is not defined, the workflow process name is used as the file name, and the file is placed in the TC_TMP_DIR directory.

- **-include_process_info**
  Includes the workflow process information in the PLM XML file.

- **-revrule**
  Specifies the revision rule to be applied for the BOM lines while exporting the structure.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
None.

**Note**
Exporting this information may take some time, depending on the export content. Siemens PLM Software recommends using the **-context** and **-file** arguments, which provide better control over the XML file’s content and location, respectively.

**EXAMPLES**
This example releases an item revision, exporting the item revision information along with the BOM to a PLM XML file and sending the file to a third-party application. In this example, it is assumed that there is a transfer mode context named **MyApplication** that has a tool attached that connects to the third-party
application and process the PLM XML file. Place this handler immediately after you add a release status.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-context</td>
<td>MyApplication</td>
</tr>
<tr>
<td>-attach</td>
<td>target</td>
</tr>
<tr>
<td>-file</td>
<td>tceng2myap.xml</td>
</tr>
<tr>
<td>-revrule</td>
<td>Latest Working</td>
</tr>
</tbody>
</table>
Appendix A  Workflow handlers

EPM-generate-image

DESCRIPTION
Generates NX part images for display by Web Reviewer. This handler calls an external NX UFUNC (no license required) to accomplish this. The generated images are stored as named references to the UGMASTER dataset; image types and sizes are specified in the preference XML file.

SYNTAX
EPM-generate-image [-stop] [-continue]

ARGUMENTS
- **-stop**
  Halts the process if image generation is unsuccessful.

- **-continue**
  For noncritical image generation, continues the process regardless of unsuccessful image generation.

PLACEMENT
Place at a point in the workflow process where the initiator has write and copy access to the UGMASTER dataset (that is, before object protections are locked down). Siemens PLM Software recommends that this handler have its own Review task at the beginning of the workflow process.

RESTRICTIONS
- Parts requiring images must be UGMASTER dataset targets of the workflow process.

- The **ugimg** executable must be located in the $UGII_BASEDIR/ugmanager directory.

  **Note**
  Part files are automatically updated to the current NX version.
EPM-generate-ugcgm-drawing

DESCRIPTION
Generates drawing sheet datasets (CGM images) of NX drawings for display in Lifecycle Visualization. You must add this handler to a release procedure as an action handler. You should initiate the release procedure containing this action handler by selecting the UGPART/UGMASTER dataset. The UGMGR_DELIMITER preference must be added as a site preference. This handler calls an external NX UFUNC program to generate the CGM images of the drawing sheets in the part. The generated images are stored as named references to the DrawingSheet dataset that is attached to the UGMASTER/UGPART dataset with an IMAN_Drawing relationship.

This handler only works from the rich client user interface if the UGII_BASE_DIR and UGII_ROOT_DIR environment variables are set to the required NX version. This example depicts the two environment variables set to NX on a UNIX platform.

```
export UGII_BASE_DIR = /usr/ugs180

export UGII_ROOT_DIR = /usr/ugs180/BIN
```

SYNTAX
EPM-generate-ugcgm-drawing [-type=valid-dataset-type] [-text= text | polylines]

ARGUMENTS
- **-type**
The valid dataset types for this handler are UGMASTER and UGPART. You can specify more than one dataset type separated by a comma between the two dataset types. If you do not specify any dataset type, this handler assumes UGPART as the dataset type.

- **-text**
Specifies whether the text in your file is converted into searchable, standard font text or records text as CGM polyline elements, each of which is a collection of line segments. The valid values are text or polylines.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
If you are using Teamcenter Integration for NX, this handler may require the external NX program export_ugdwgimages to be copied from $TC_BIN/ugcgm_images to $TC_BIN or UGII_BASE_DIR/ugmanager directory.

The release procedure script start_ugdwgimages looks for the UFUNC program in the UGII_BASE_DIR/ugmanager directory first, then in the $TC_BIN directory.

EXAMPLES

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>UGMASTER, UGPART</td>
</tr>
<tr>
<td>-text</td>
<td>text</td>
</tr>
</tbody>
</table>
DESCRIPTION

Sets the item revision as the primary representation of the associated part revision. This handler checks if the input item revision is mature. If it is, all part revisions for the design revision are found and the item revision is set as the primary representation.

SYNTAX

EPM-make-mature-design-primary

ARGUMENTS

None.

PLACEMENT

Preferably placed on the Complete action.

RESTRICTIONS

Considers only item revisions or a subclass of them.
EPM-mark-archive

DESCRIPTION
Generates archive requests for datasets of item revisions with the specified status. This handler should be used only when the targets of a workflow process are item revisions. This handler is very useful in archiving the experimental, prototype data and keeping only the real data.

SYNTAX
EPM-mark-archive [-exclude_related=relation::type [, relation::type..] ,-status_to_keep=status::number-of-item-revs-to-keep [, status::number-of-item-revs-to-keep..]]

ARGUMENTS
-exclude_related
Excludes the specified relation or type or type in relation from having an archive request being generated. This argument is optional. If this argument is used, either a relation or type should be specified. If only a relation is specified, :: need not be appended (for example: -exclude_related=IMAN_specification). If only a type is used, prepend the type with :: (for example: -exclude_related=::UGPART).

-status_to_keep
Release status names::number of item revisions to keep.
This means not to mark for archive the datasets of a specified number of item revisions with the specified release status.
Siemens PLM Software recommends that the number of revisions to keep should be 1 or more. This way, at least one item revisions per release status is not archived. This assures that there are no product structure configuration problems.

PLACEMENT
Requires no specific placement. Typically placed on the Complete action of the root task so that the objects are marked for archive at the end of completion of the workflow process.

RESTRICTIONS
Target objects must be item revisions.

EXAMPLES
In this example, consider the scenario:
An item has 20 item revisions out of which item revisions 1-4 have no release status, item revisions 5-9 have release status Released, item revisions 10-14 have release status R, and item revisions 15-19 have release status X set.
The EPM-mark-archive handler with the following arguments is added to the Complete action of the root task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-exclude_related</td>
<td>IMAN_manifestation::UGPART</td>
</tr>
<tr>
<td>-status_to_keep</td>
<td>R::3, X::2</td>
</tr>
</tbody>
</table>

The previously created item revision workflow process template is initiated on the 20th item revision. After the workflow process is completed, the following results are expected.
Appendix A  Workflow handlers

All datasets except those:

- With manifestation relation
- Of type UGPART

of the item revisions, 10-11 and 15-17, are marked for archive.
EPM-perform-offline-export

DESCRIPTION
Performs a Briefcase/PDX export using a workflow process.

SYNTAX
EPM-perform-offline-export -site=site-name [-optionset=transfer-option-set]
[ -usegs=True | False ] [-revisionrule=revision-rule-name] [-bomlevel=depth]
[ -vendors=vendor-names ] [-reason=export-reason-string] [-immediate=True | False]
[-notify=True | False] [-emailaddrs=comma-separated-email-ids]

ARGUMENTS
-site
Specifies the destination site where the Briefcase or PDX package is to be exported.

-optionset
Specifies the transfer option set to be used during export. If none is specified, the system uses either TIEPDXOptionSetDefault (for a PDX export) or TIEUnconfiguredExportDefault (for a Briefcase export) based on availability of the set.

-usegs
Specifies whether the transaction should go through Global Services or not. Valid values are True and False. The default value is False, which is a non-Global Services-based transaction.

-revisionrule
Specifies the revision rule to be used to perform the BOM configuration.

-bomlevel
Specifies the depth to which the BOM must be traversed for export.

-vendors
Specifies the list of vendor names whose manufacturer parts are to be exported. Only parts from these vendors get exported.

-reason
Specifies the reason for the export (up to 240 characters).

-immediate
Specifies whether the transaction should be performed immediately or not. This argument is applicable only when -usegs=True. Valid values are True and False. The default value is False.

-notify
Specifies whether the users listed in the -emailaddrs argument are notified when the transaction is completed. This argument is applicable only when -usegs=True. Valid values are True and False. The default value is False.

-emailaddrs
Lists the comma-separated e-mail IDs of users to be notified when the transaction is completed. This argument is applicable only when -usegs=True and when the -notify=True.
**Appendix A  Workflow handlers**

**PLACEMENT**

Requires no specific placement.

**RESTRICTIONS**

None.

**EXAMPLES**

This example exports a package to **Supplier-site-1** using a custom option set without using Global Services.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-site</td>
<td>Supplier-site-1</td>
</tr>
<tr>
<td>-optionset</td>
<td>CustomOptionSet1</td>
</tr>
<tr>
<td>-usegs</td>
<td>False</td>
</tr>
</tbody>
</table>
EPM-publish-target-objects

DESCRIPTION
Publishes target objects (that is, enters them) in the Object Directory Services (ODS) database.

SYNTAX
EPM-publish-target-objects [-class=classname] [-site=site-ID]

ARGUMENTS
-class
Class of the target objects being published. This argument can be supplied more than once to publish multiple classes of target objects. If not supplied, all target objects are published. See the second item in the Restrictions section.

-site
ODS sites that publishes the objects. This argument can be supplied more than once to publish the objects to multiple ODS sites. If not supplied, the default ODS is used.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
• Requires Multi-Site Collaboration to be configured at your site.

• The class must be defined by the TC_publishable_classes preference or it cannot be published.

EXAMPLES
This example shows how to publish all item revision target objects to Detroit and Tokyo ODSs:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-class</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-site</td>
<td>Detroit, Tokyo</td>
</tr>
</tbody>
</table>

**EPM-remove-objects**

**DESCRIPTION**

Removes the specified target or reference objects from the workflow process. This handler can use either a set of arguments to define which objects to remove or keep, or a list of values (LOV) to define a list of object types to remove.

This handler can be used effectively with the **EPM-attach-related-objects** handler. For example, consider a task where users can manually add objects to any target revisions, such as new datasets through a specification relation. Users can also attach objects directly as targets to the workflow process. To ensure only allowable objects are attached as targets on approval, remove all objects except for the revisions using the **EPM-remove-objects** handler with the **-keep_targets=(ItemRevision)** argument. Then re-add the revision's attachments using the **EPM-attach-related-objects** handler.

**Note**

Enable debugging functionality for this handler with the **TC_HANDLERS_DEBUG** environment variable.

For more information about implementing this environment variable, see the Preferences and Environment Variables Reference.

**SYNTAX**

```
EPM-remove-objects {{-remove_targets=types | -keep_targets=types}}
{{-remove_refs=types | -keep_refs=types}} | -lov=lov-name
```

**ARGUMENTS**

**-remove_targets**

Defines the classes and/or types of target objects to remove from the workflow process.

Accepts a comma-separated list of classes and/or types in the format:

```
[(Class)[!Type1][,(Class2)[,Type1,....]]]| Type1[,Type2][,....]
```

For example, to specify datasets and forms:

```
(Dataset),(Form)
```

For an overview of using multilevel object paths in handlers, see Defining multilevel object paths.

**Note**

The **-keep_targets** and **-remove_targets** arguments are mutually exclusive.

**-keep_targets**

Defines the classes and/or types of target objects to be kept. All other target objects are removed from the workflow process. If the handler does not find any objects to keep, it does not remove any objects.

Accepts a comma-separated list of classes and/or types in the format:

```
[(Class)[!Type1][,(Class2)[,Type1,....]]]| Type1[,Type2][,....]
```
For example, to specify datasets and forms:

(Dataset),(Form)

For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

**Note**

The `-keep_targets` and `-remove_targets` arguments are mutually exclusive.

**-removeRefs**

Defines the classes and/or types of reference objects to remove from the workflow process.

Accepts a comma-separated list of classes and/or types in the format:

```
[(Class)[!Type1][,(Class2)[,Type1[,...]]]]| Type1[,Type2][,...]
```

For example, to specify datasets and forms:

(Dataset),(Form)

For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

**Note**

The `-keepRefs` and `-removeRefs` arguments are mutually exclusive.

**-keepRefs**

Defines the classes and/or types of reference objects to be kept in the workflow process.

Accepts a comma-separated list of classes and/or types in the format:

```
[(Class)[!Type1][,(Class2)[,Type1[,...]]]]| Type1[,Type2][,...]
```

For example, to specify datasets and forms:

(Dataset),(Form)

For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

**-lov**

Specifies a LOV to use to define which objects to remove. This argument is mutually exclusive of all other arguments.

For an overview of using LOVs in handlers, see *Using lists of values (LOVs) as handler arguments*. See the LOV row, next, for required LOV format.

**Note**

The `-keepRefs` and `-removeRefs` arguments are mutually exclusive.
Appendix A  Workflow handlers

LOV

{$TARGET|REFERENCE}.types
{$TARGET|REFERENCE}.types
...
{$TARGET|REFERENCE}

Specifies whether to remove targets, or to remove references.

Accepts a comma-separated list of classes and/or types in the format:
[(Class)[!Type1][,(Class2)[,Type1[,...]]]| Type1[Type2][,...]

For example, to specify datasets and forms:

(Dataset),(Form)

For an overview of using multilevel object paths in handlers, see Defining multilevel object paths.

PLACEMENT

Place on the Start or Complete action of any task.

To allow the removal of targets, ensure that the disallow-removing-targets handler is not placed on the root task of the respective workflow process template and the affected users have change access to the workflow target objects. You may use the EPM-set-rule-based-protection handler to ensure that the required change access to target objects is asserted.

RESTRICTIONS

When using a LOV, you can only define objects to be removed. You cannot define objects to be kept.

EXAMPLES

• This example removes any folders or items attached as targets:

  Argument         Values
  -remove_targets  (Folder), (Item)

  Alternatively, you can use these LOV settings:

  Argument         Values
  -lov             SYS_EPM_remove_folders_items

  where the SYS_EPM_remove_folders_items LOV contains the data:

  # LOV SYS_EPM_remove_folders_items
  $TARGET.(Folder),(Item)

• This example retains only item revisions, removing all other targets:

  Argument         Values
  -keep_targets    (ItemRevision)
EPM-run-external-command

DESCRIPTION

Runs external system commands. The external command can be sent a variety of information that includes configurable arguments, a configuration file, a list of data and a list of target and attachment details. If dataset details are required there is also an optional export feature to export specified files from the specified datasets to a specified export directory. All options are configured using a list of values (LOV), hence there is only one argument. Nearly all options can be specified in the LOV using specially formatted lines to extract object properties.

Note

Enable debugging functionality for this handler with the TC_HANDLERS_DEBUG environment variable.

For more information about implementing this environment variable, see the Preferences and Environment Variables Reference.

SYNTAX

EPM-run-external-command -lov=lov-name

ARGUMENTS

-lov

Specifies the List of Values (LOV) used to configure all options.

For an overview of using LOV in handlers, see Using lists of values (LOVs) as handler arguments.

LOV

lov-name can contain several lines in the following format:

<KEYWORD><OPTION>=<Value>
<KEYWORD><OPTION>=<%formatted string%>
<KEYWORD><%formatted string%>

• KEYWORD

Specifies a keyword to indicate the type of information to extract and send to the external command. Keywords are described below:

  o INPUT

  Specifies options to configure the handler.

  INPUT~OPTION=Value

  OPTION can contain any of the following values:

  • Target

  Indicates the main workflow process objects to extract data. The following example sets all item revision targets of the workflow process as the main objects:

  INPUT~Target=$TARGET.(ItemRevision)

  The following example uses references of the workflow process. These objects the main objects that %property% fields relate to in %formatted strings%.
Appendix A  Workflow handlers

INPUT~Target=$REF.(ItemRevision)

- **Application**
  Indicates the system application to run.

  `INPUT~Application=${TC_ROOT}\local\tools\run_ext_app`

- **CallPerTarget**
  Controls the application execution, once or per target found from `INPUT~Target`.

  `INPUT~CallPerTarget=YES | NO`
  
  - **YES** calls the application separately for each target from `INPUT~Target`. This is the default behavior if this option is not provided. If one of the applications detects an error, processing terminates.
  
  - **NO** calls the application once and sends its data about all targets found from `INPUT~Target`.

- **ErrorMsg1**
  Custom error message to be displayed to the user upon a fail code being returned from the external application. A return status of zero, (0), indicates the application terminated successfully; any other value indicates a failure.

  In scripts, this is typically achieved using an exit command, for example, `exit 0` for success, `exit 1` for failure.

  A `%formatted string%` can be used with this option, including the `$SYSTEM_ERROR` variable to display the error code returned by the application. For example:

  ```
  INPUT~ErrorMsg1=BOM checks failed on target %object_string% with error %$SYSTEM_ERROR%
  ```

  You can use this error message to reflect the type of application, or external checking, that was being performed. If not provided then a default, non-localized, message is returned.

- **ErrorMsg2**
  Optional custom error message to be displayed to the user upon a fail code being returned from the external application. You can use this message to provide the user a help message, that is, where to look for more information on the problem. For example:

  ```
  INPUT~ErrorMsg2=Please see your e-mail for details.
  ```

  **Note**
  
  Because error messages are displayed in reverse order this message appear before `ErrorMsg1`.

- **ExportPath**
Defines a directory to export files in datasets. The presence of this option enables the export feature. If the option is not provided, then no files are exported. This option works with the `DATA-DATASETS[=options]` described below which creates a data file listing all required datasets. The `options` argument describe the relations, dataset types, and named references required. If `ExportPath` is also defined, then the files from the required name references are exported. For example:

```
INPUT~ExportPath=${TC_TMP_DIR}\WF\Exports
```

The handler does not remove any remaining files from the export path when the external application has terminated. It is the responsibility of the application to remove any remaining files from this directory. If any files being exported already exist in the export directory, then the export fails and the existing file is not overwritten. If this occurs, an error is written to the syslog but not displayed to the user and the handler continues.

- **ExportOrigFile**

Exports files with original file name. If this option is not defined, the handler exports files with the name stored in the volume. This option controls the name used for any exported files from datasets when `ExportPath` and `DATA-DATASETS` are defined. This option requires a `YES` value. For example:

```
INPUT~ExportOrigFile=YES
```

- **DataPath**

Defines a directory to write data files. This option defines where the configuration file, defined using the `CFG` keyword, and the data files, defined using the `DATA` keyword, are written. For example:

```
INPUT~DataPath=${TC_TMP_DIR}\WF\Data
```

- **CFG**

Specifies information to be written to an optional configuration file that can be passed to the external command as an argument. The format is:

```
CFG=%formatted string%
```

This file name can be extracted in a `%formatted strings%` using the `$CONFIG_FILE` variable. For example:

```
CFG=JobTag=%$PROCESS.TAG%
CFG=JobName=%$PROCESS.object_name%
CFG=RevID=%$TARGET.item_revision_id%
CFG=ItemID=%$TARGET.item.item_id%
CFG=Project=%$TARGET.IMAN_master_form.project_id%
CFG=OwningUser=%$TARGET.owning_user%
CFG=OwningGroup=%$TARGET.owning_group%
```

The following example writes the following string:

```
JobTag=QmBJ0uKn9KRfCAAACAAAAAAAAAA
```

to the configuration file for **000001/A** the workflow process with the **000001/A** target revision owned by tim and Designers group:

```
JobName=000001/A RevID=A ItemID=000001 Project=Project X
```
Appendix A  Workflow handlers

OwningUser=Tim (tim) OwningGroup=Designers

- ARG
  Specifies optional arguments to be sent to the external command. The format is:

  \texttt{ARG=\%formatted \ string\%}

  For example:

  \begin{verbatim}
  ARG--cfg=\$CONFIG\_FILE\
  ARG--files=\$DATASET\_FILE\
  ARG--data=\$DATA\_FILE
  \end{verbatim}

- DATA
  Specifies information to be extracted from targets, references, and their related objects. The possible formats are:

  - DATASETS
    \texttt{DATA-DATASETS[=options]}

    writes a fixed format data file containing information about attached datasets that can optionally be exported with \texttt{INPUT ExportPath}.

    This option is used to extract details about datasets attached to the objects specified by \texttt{INPUT-Target}. If \texttt{INPUT-ExportPath} is defined, then the required files are exported from the required datasets to the export path specified. The properties extracted from the datasets are written to a file with the name \texttt{process_tag_datasets.txt} in the current directory or in the directory specified using \texttt{INPUT-DataPath}. This file name can be extracted in a \%formatted strings\% using \texttt{\$DATASET\_FILE}.

    Optional filters for relation types, dataset types, and reference types can be supplied. For each filter, an asterisk (*) can be supplied as a wild card to indicate any type. If dataset types are supplied and no reference types, then all references are listed in the data file. If no filters are supplied, then all datasets in all relations and all of their references are listed. Any reference files that are exported have their absolute file path listed in the data file. This provides the ability for the external application to perform operations on these files. For example, running checks, printing, converting or to get information about \texttt{UGPART} references in \texttt{UGMASTER} and \texttt{UGPART} datasets in the \texttt{IMAN\_specification} relation.

    \begin{verbatim}
    DATA-DATASETS=IMAN\_specification=UGMASTER,UGPART
    \end{verbatim}

    The datasets data file is written in a fixed format as follows:

    \begin{verbatim}
    item_id=rev_id=relation_type=dataset_type=dataset_name=dataset_tag=reference_type=file_name
    \end{verbatim}

  - LOV
    \texttt{DATA-LOV=lov-name}
writes a file containing information about the targets, references and their related objects. A second LOV is used to define all of the objects and properties to extract.

Specifies a separate LOV containing a list of alternating lines containing either:

OBJECT: multilevel.object.path

or

PROP: %formatted string%

The lines beginning with OBJECT: are used to find objects using multilevel object paths; lines beginning with PROP: specify the properties to extract from these objects and write out to the data file.

The first line in the LOV can be a PROP: line, for example, without a preceding OBJECT: line, in which case properties are extracted from the main objects found from INPUT~Target.

For example:

```
INPUT~LOV=SYS_EXT_CMD_object_data
```

where LOV SYS_EXT_CMD_object_data can contain:

```
PROP:%item.item_id%~%item_revision_id%~%object_name%~%object_type%
OBJECT:*.IMAN_reference
PROP:REF~%object_string%~%object_type%
OBJECT:*.IMAN_specification.

UGMASTER, UGPART
PROP:UG-HDR~Name~Material
PROP:UG~%object_string%~%

UGPART-ATTR.material%
```

This example begins by extracting properties from the main objects, then from reference objects attached to the main objects, and finally from the UGMASTER and UGPART datasets. Notice that there are two PROP: lines for the UGMASTER and UGPART datasets, the first line just has fixed text acting like a header line and the second defines the properties to extract (which includes the material attribute from the UGPART-ATTR named reference form).

In the OBJECT: lines, a type is required at the start of the multilevel object path to provide more flexibility. An asterisk indicates any type or an asterisk is automatically added within any %formatted string% for convenience when starting with a $keyword such as $TARGET, otherwise an asterisk, or type, is still required, as in the example for the *UGPART-ATTR.material*. The output from this example:

```
000001-A-000001~ItemRevision REF-000003/A~ItemRevision
UG-HDR~Name~Material UG-UGMASTER-000001/A~Steel
```

- **OPTION**

  Some keywords have options which can be defined.

- **Value**

  You can use any text as a value. However, it is possible to extract values from environment variables within the text using the format:

```
text$(ENV_VAR)text$(ENV_VAR)text
```
Appendix A  Workflow handlers

- **%formatted string%**
  A `%formatted string%` is a string containing alternating fixed text, and object properties defined within a pair of percent characters (%), similar to a batch file statement containing environment variables.

  The format is:
  
  ```text
  text%property%text%property%text
  ```

  where each property is defined within two percent characters (%) with fixed text between each property.

  A property to extract relates to a previously defined object, to the workflow process targets or to the current workflow process, depending on the current context where the formatted string is being used and some optional variables. The property can be specified as a single Teamcenter property, for an already specified object, or a multilevel object path and property to extract information from another object related to the already defined object target or workflow process.

  If a multilevel object path is used within a property field and returns more then one object, then a comma-separated list of the values for the property from each object is given.

  A special keyword tag can be used instead of a property name to extract a string representation of an object **PUID**.

  o  If the defined object is an item revision, then the following example extracts ItemID/RevID.

  ```text
  %item.item_id%/%item_revision_id%
  ```

  where `%item.item_id%` extracts the item_id from the revision's item. The `/` is the fixed text and `%item_revision_id%` extracts the revision's id.

  o  The following example writes the project ID from a target revision’s master form as a line in the configuration file.

  ```text
  CFG-Project=%$TARGET.IMAN_master_form.project_id%
  ```

  If the project is **Project X**, the configuration file contains the following line:

  ```text
  Project=Project X
  ```

  This example uses the `$TARGET` variable to specify which object the multilevel path starts.

**VARIABLES**

Values from environment variables can also be extracted within a `%formatted string%` using the same format as described for Value. The `$ENV_VAR` does not have to be included within the pair of % characters.

There are also some internal variables which can be specified with some options. These are indicated with a $ character, but without the curly brackets used for environment variables. Also, unlike the environment variables, these must be defined within a pair of percent % characters. For example:

```text
ARG~~cfg_file=%$CONFIG_FILE%
```

This example specifies an argument to be sent to the external command. It specifies a `%formatted string%` of `cfg_file=%$CONFIG_FILE%`, so the fixed text is `cfg_file=`, and `%$CONFIG_FILE%` (between two % signs) extract the name of the
configuration file generated by the handler. This option is explained in full detail below under the section for ARG, along with other variable.

The following handler variables are available:

**$TARGET**

Specifies that a multi level object path should start searching for objects from the current target, as specified with INPUT-Target=target.path.

In the main LOV, this is taken as default and so does not have to be specified (except when using DATA-LOV), so

%$TARGET.item.item_id%

is the same as

%item.item_id%

**$PROCESS**

Specifies that a multilevel object path should start searching for objects from the current workflow process.

For example:

%$PROCESS.object_name%

extracts the workflow process’s name.

This option also provides a path to extract details about objects attached to the workflow process as targets or references.

For example:

%$PROCESS.$REF.object_string%

returns a comma-separated list of the object_string property from all references attached to the workflow process, and:

%$PROCESS.$TARGET.object_string%

returns a list of all targets.

**$USER**

Can be used to extract information about the current logged in user.

Used on its own will give the full user format person (user_id).

Or a path can be used to get other user, person, or group information.

For example:

```
CFG-Person=%$USER.person%
CFG-UserID=%$USER.userid%
CFG-LoginGroup=%$USER.login_group%
CFG-Group=%$USER.group.name%
CFG-Email=%$USER.Person.PA9%
```

**$CONFIG_FILE**

Gets the name of the configuration file generated by the handler. The format of the name is:

`DataPath\process_tag_config.txt`

or, if CallPerTarget is set to YES:

`DataPath\process_tag_x_config.txt`

x is an incrementing number per target.
Appendix A  Workflow handlers

$DATA_FILE Gets the name of the data file generated by the handler for DATA-LOV. The format of the name is:

\( \text{DataPath}\backslash \text{process_tag_data.txt} \)

or, if CallPerTarget is set to YES

\( \text{DataPath}\backslash \text{process_tag_x_data.txt} \)

Where \( x \) is an incrementing number per target.

$DATASET_FILE Gets the name of the datasets information file generated by the handler for DATA-DATASETS. The format of the name is:

\( \text{DataPath}\backslash \text{process_tag_datasets.txt} \)

or, if CallPerTarget is set to YES

\( \text{DataPath}\backslash \text{process_tag_x_datasets.txt} \)

Where \( x \) is an incrementing number per target.

$SYSTEM_ERROR Gets the error code number returned by the external application. Can be used in the ErrorMsg1 and ErrorMsg2 error messages.

PLACEMENT Requires no specific placement, however, do not place on the Perform action of the root task.

RESTRICTIONS This handler does not extract data in PLM XML format. The format of the extracted data is defined completely in the LOV using percent (%) formatted strings, except for the file listing the export dataset, which is in a fixed format.

This handler does not have an import feature; however, dataset tags are written to the exported datasets data file and so could be used by a standalone ITK program to import files. Do not use this handler to run an external application that takes a long time to run. It may appear that Teamcenter is unresponsive. If the success or failure of the application is required for process control, it is necessary to wait for the application. In this case, ensure that the workings of the application is visible in a new window to show the user some feedback. Any files exported by the handler are not deleted by the handler after the external application finishes. It is the responsibility of the external application to clean up the export directory.

EXAMPLES

- Example 1

The following example calls an application, specified by an environment variable, to perform checks on CAD files. This application requires a configuration file to define various parameters. One of these is the an e-mail address so that it can send the user a report. The name of the configuration file is sent to the application as an argument, as is the file name of the data file containing information about the exported dataset files.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM_run_cad_checks</td>
</tr>
</tbody>
</table>

The SYS_EPM_run_cad_checks LOV contains the following data:
## Workflow handlers

### LOV usage

<table>
<thead>
<tr>
<th>LOV contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT~Target=$TARGET.(ItemRevision)</td>
<td>Specifies that the main objects from which data is to be extracted is the job targets which is of class ItemRevision. If multiple targets are found then the application will either be called separately for each target or once with all of the data from all targets, depending on the setting CallPerTarget which is defined just below.</td>
</tr>
<tr>
<td>INPUT~ErrorMsg1=Cad checks errors (Error %$SYSTEM_ERROR%)</td>
<td>Defines an error message which is displayed to the user if the application returns an error status.</td>
</tr>
<tr>
<td>INPUT~ErrorMsg2=Please see your e-mail for details</td>
<td>Defines an optional second error message which is displayed to the user as well as ErrorMsg1.</td>
</tr>
<tr>
<td>INPUT~Application=${CUST_CAD_CHECK_APPLICATION}</td>
<td>Defines the external application which is to be executed. This application is defined by a system environment variable, which in this example is CUST_CAD_CHECK_APPLICATION.</td>
</tr>
<tr>
<td>INPUT~CallPerTarget=YES</td>
<td>Calls the application for each target.</td>
</tr>
<tr>
<td>INPUT~DataPath=C:\WF\Data</td>
<td>Sets a path for data files.</td>
</tr>
<tr>
<td>INPUT~ExportPath=C:\WF\Exports</td>
<td>Sets a path for exported dataset files</td>
</tr>
<tr>
<td>CFG~JobTag=%$PROCESS.object_tag%</td>
<td>Writes the process tag (PUID) to the configuration file as JobTag=Job Tag.</td>
</tr>
<tr>
<td>CFG~JobName=%$PROCESS.object_name%</td>
<td>Writes the workflow process name to the configuration file as JobName=Job Name.</td>
</tr>
<tr>
<td>CFG~RevID=%$TARGET.item_revision_id%</td>
<td>Writes the target object revision ID to the configuration file as RevID=RevID.</td>
</tr>
<tr>
<td>CFG~ItemID=%$TARGET.item.item_id%</td>
<td>Writes the target object item ID to the configuration file as ItemID=ItemID.</td>
</tr>
<tr>
<td>CFG~Project=%$TARGET.IMAN_master_form.project_id%</td>
<td>Writes the target object Project ID, from the revision master form, to the configuration file as Project=ProjectID.</td>
</tr>
<tr>
<td>CFG~CadProc=${CUST_CAD_CHECK_PROC}</td>
<td>Writes the environment variable value to the configuration file as CadProc=cad_proc.</td>
</tr>
<tr>
<td>CFG~OwningUser=%$TARGET.owning_user%</td>
<td>Writes the target object owning user to the configuration file as OwningUser=user.</td>
</tr>
</tbody>
</table>
## Appendix A  Workflow handlers

### LOV usage

<table>
<thead>
<tr>
<th>LOV contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFG~OwningGroup=%$TARGET.owning_group%</td>
<td>Writes the target object owning group to the configuration file as <strong>OwningGroup=group</strong>.</td>
</tr>
<tr>
<td>CFG~Email=%$USER.E_Mail%</td>
<td>Writes the current user's e-mail to the configuration file, where <strong>E_Mail</strong> is the label from the person form.</td>
</tr>
<tr>
<td>CFG~SMTPServer=${CUST_RELEASE_SMTP_SERVER}</td>
<td>Writes the environment variable value to the configuration file.</td>
</tr>
<tr>
<td>CFG~FunctionsFile=${CUST_RELEASE_FUNC_FILE}</td>
<td>Writes the environment variable value to the configuration file.</td>
</tr>
<tr>
<td>CFG~SysAdminEmail=${CUST_RELEASE_SA_MAIL}</td>
<td>Writes the environment variable value to the configuration file.</td>
</tr>
<tr>
<td>CFG~AppsArray=Apps1</td>
<td>Writes the value <strong>AppsArray=Apps1</strong> to the configuration file.</td>
</tr>
<tr>
<td>CFG~WarningDir=${CUSTOMER_RELEASE_WARNING_DIR}</td>
<td>Writes the environment variable value to the configuration file.</td>
</tr>
<tr>
<td>CFG~UPG=${UPG}</td>
<td>Writes the environment variable value to the configuration file.</td>
</tr>
<tr>
<td>CFG~Desc=%$TARGET.object_desc%</td>
<td>Writes the target object description to the configuration file.</td>
</tr>
<tr>
<td>DATA<del>DATASETS=IMAN_specification</del>UGMASTER~UGPART</td>
<td>Extracts information about <strong>UGPART</strong> references in <strong>UGMASTER</strong> datasets attached to the target revision.</td>
</tr>
<tr>
<td>ARG~cfg=%$CONFIG_FILE%</td>
<td>Sends the configuration file name as an argument.</td>
</tr>
<tr>
<td>ARG~files=%$DATASET_FILE%</td>
<td>Sends the dataset data file name as an argument.</td>
</tr>
</tbody>
</table>

- **Example 2**

The following example shows the use of **DATA~LOV=lov-name** to extract various details.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lov</td>
<td>SYS_EPM_send_ecr_relation_data</td>
</tr>
</tbody>
</table>

when the **SYS_EPM_send_ecr_relation_data** LOV contains the following data:
### DATA-LOV=lov-name

<table>
<thead>
<tr>
<th>LOV contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT~Target=(ItemRevision)</td>
<td>Specifies that the main object from which data is to be extracted is the job target which is of the <strong>ItemRevision</strong> class.</td>
</tr>
<tr>
<td>INPUT~Application=${CUST_ECR_EXT_APPLICATION}</td>
<td>Defines the external application that is executed. This application is defined by a system environment variable.</td>
</tr>
<tr>
<td>ARG~item=%$TARGET.item.item_id %</td>
<td>Sends the target object’s item ID as an argument to the application.</td>
</tr>
<tr>
<td>ARG~rev=%$TARGET.item.item_revision_id %</td>
<td>Send the target object’s revision ID as an argument to the application.</td>
</tr>
<tr>
<td>ARG~dest=${CUST_RELEASE_DEST}</td>
<td>Send the environment variable’s value as an argument to the application.</td>
</tr>
<tr>
<td>ARG~type=ECR</td>
<td>Sends the value as an argument to the application.</td>
</tr>
<tr>
<td>ARG~data=%$DATA_FILE%</td>
<td>Sends the name of the data file, to be produced by <strong>DATA-LOV</strong>, as an argument to the application.</td>
</tr>
</tbody>
</table>

### DATA-LOV=lov-name

Specifies an LOV containing a list of alternating lines starting with **OBJECT**; to specify an object, and then **PROP**; to specify the properties to extract from the object to write out to a data file.

### DATA-LOV=SYS_EPM_get_ecr_relation_data

This LOV extracts details from the affected item revisions attached to the **Mini**, **Minor**, and **Major** relations in an ECR revision target.

The objects are specified using multiple level paths and start from the target objects. The property strings use the `%formatting%` notation.

- Output in the data file, if the target has two minor relations and one major relation:

```
item-00001~A~Mini
item-00002~B~Mini
item-00005~A~Major
```

### LOV SYS_EPM_get_ecr_relation_data

<table>
<thead>
<tr>
<th>LOV contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROP<del>item.id</del>ECR</td>
<td>Extract properties from the target revision.</td>
</tr>
<tr>
<td>Started~%creation_date%<del>%owning_user%</del>%IMAN_master_form.ecr_prty%</td>
<td></td>
</tr>
</tbody>
</table>

---

**PLM00037 I**

*Workflow Designer Guide A-137*
## LOV SYS_EPM_get_ecr_relation_data

<table>
<thead>
<tr>
<th>LOV contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT:(ItemRevision).Mini.(ItemRevision)!Buy Revision!Customer Revision!RawMaterial Revision</td>
<td>From any ItemRevision targets, find any ItemRevision objects attached to the Mini relation, except for specific types, for example, Buy Revision.</td>
</tr>
<tr>
<td>PROP:%item.item_id%~%item_revision_id%~Mini</td>
<td>Extract properties from any Mini relation revisions.</td>
</tr>
<tr>
<td>OBJECT:(ItemRevision).Major.(ItemRevision)!Buy Revision!Customer Revision!RawMaterial Revision</td>
<td>From any ItemRevision targets, find any ItemRevision objects attached to the Major relation, except for specific types, for example, Buy Revision.</td>
</tr>
<tr>
<td>PROP:%item.item_id%~%item_revision_id%~Major</td>
<td>Extract properties from any Major relation revisions.</td>
</tr>
<tr>
<td>OBJECT:(ItemRevision).Minor.(ItemRevision)!Buy Revision!Customer Revision!RawMaterial Revision</td>
<td>From any ItemRevision targets, find any ItemRevision objects attached to the Minor relation, except for specific types, for example, Buy Revision.</td>
</tr>
<tr>
<td>PROP:% item.item_id %~% item_revision_id %~Minor</td>
<td>Extract properties from any Minor relation revisions.</td>
</tr>
</tbody>
</table>
## EPM-send-target-objects

**DESCRIPTION**
Sends objects to other Multi-Site Collaboration sites.

**SYNTAX**
```
EPM-send-target-objects [-class=classname] [-target_site=site-name | ALL | -owning_site=site-name] [-target_revision_only=YES] [-reason=string]
```

**ARGUMENTS**
- **-class**
  Sends target objects of the specified class to the specified site. You can specify this argument more than once to send different classes of target objects. If this argument is not used, all target objects are sent.

- **-target_site**
  Sends the target objects to the specified site, but does not transfer ownership. You can specify multiple sites, separated by a comma. Use ALL to send the specified target objects to all sites.

  This argument is mutually exclusive with the **-owning_site** argument. One or the other of these two arguments must be specified for the handler to run.

- **-owning_site**
  Transfers site ownership of the target objects to the specified site. All target objects are converted to reference objects before the data transfer.

  This argument is mutually exclusive with the **-target_site** argument. One or the other of these two arguments must be specified for the handler to run.

- **-target_revision_only**
  Exports only the released item revision to the remote site. When this argument is not used, all item revisions are exported.

  Do not use this argument with the **-owning_site** argument; all revisions must be transferred when transferring site ownership.

- **-reason**
  Allows you to enter a string (up to 240 characters) explaining why these objects were sent.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
- Requires Multi-Site Collaboration to be configured at your site.
- The sending site must own all objects to be sent to other sites.
- When using the **-target_revision_only** argument, the **-class** argument must be set to **ItemRevision**.

  This argument cannot be used with the **-owning_site** argument; all revisions must be transferred when transferring site ownership.
Appendix A  Workflow handlers

EXAMPLES

- This example shows how to send all item target objects to the Detroit and Tokyo sites without transferring ownership:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-class</td>
<td>Item</td>
</tr>
<tr>
<td>-target_site</td>
<td>Detroit, Tokyo</td>
</tr>
</tbody>
</table>

- This example shows how to send item and dataset target objects to all sites without transferring ownership:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-class</td>
<td>Item, Dataset</td>
</tr>
<tr>
<td>-target_site</td>
<td>ALL</td>
</tr>
</tbody>
</table>

- This example shows how to transfer site ownership of item and dataset target objects to the Tokyo site:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-class</td>
<td>Item, Dataset</td>
</tr>
<tr>
<td>-owning_site</td>
<td>Tokyo</td>
</tr>
</tbody>
</table>
EPM-send-to-oramfg

DESCRIPTION
This handler performs the following functions:

• Generates the transaction group ID and stores it in the TransactionGroupId box in the OraMfgTransferInfo form attached to the workflow reference list.

• Populates the OverallTransferStatus box with the transfer status In Progress.

• Sends a message in the required format to Global Services.

SYNTAX
EPM-send-to-oramfg
-transfermode=transfermode_name -transferoptionset=option_set_name
[-revrule=revision_rule_name]

ARGUMENTS
-transfermode
Specifies the name of the transfer mode to use for sending data to Oracle.

-transferoptionset
Specifies the name of the transfer option set to use for sending data to Oracle.

-revrule
Specifies the revision rule to use when transferring BOMs. If not specified, the default revision is used. This argument is optional.

PLACEMENT
Place under the Complete action.

RESTRICTIONS
None.
EPM-set-condition-by-check-validation-result

DESCRIPTION
This action handler can be configured to set the **Condition** task result status using **Validation Rule** and **Validation Object** applications a from workflow process. It can also check target NX datasets validation result status. To add this handler to a workflow process template, the user must have a well-defined **Validation Rule set file** that best describes the business process in terms of which NX datasets should run checks at certain times and the conditions that the check must meet.

The handler sets the **Condition** task result based on the overall result status of the verification (true when all target NX datasets satisfy all rules defined in the **Validation Rule set file**). The handler logs validation rules and validation result checks. The format of the log file name is *First-target-name_Time-stamp*. The log file is stored in the directory specified by the **TC_TMP_DIR** environment variable. If **TC_TMP_DIR** is not defined, it is stored in the %TEMP% directory (Windows) or /tmp directory (Linux).

When a **Condition** task template is configured with this action handler, no other saved queries or handlers should be added to the task template. The logic that this handler uses to check validation results is the same logic used by the **EPM-check-validation-result-with-rules** rule handler.

SYNTAX

```
EPM-set-condition-by-check-validation-result
-rule_item_revision=item-revision-id [-current_event=value] [-pass_item_revision_only]
```

ARGUMENTS

- **-rule_item_revision**
The item revision ID that the validation rule set dataset is attached under.

- **-current_event**
A value that is used to select validation rules from the rule file by comparing with the event values list of each rule. When this argument is not provided, all rules from the rule file are selected at the first step. When a rule is defined without the event values list, then the rule is selected at the first step. The event values list of a rule can contain an asterisk (*) as a wildcard. The event values list also can be marked as exclusive (it is inclusive by default).

- **-pass_item_revision_only**
When this argument is added to the input list, only item revision targets are passed to the handler. NX datasets are searched from each item revision and verified according to rules.

PLACEMENT
Place under the **Complete** action.

RESTRICTIONS

- **-rule_item_revision** cannot be **NULL**.
EPM-set-form-value-AH

DESCRIPTION
Sets a particular field to a given value for all forms of the given type attached as targets of the process, and saves the forms. Use this handler to set a value that depends on the workflow process being used to transfer the data to ERP (for example, for a preproduction transfer process, the BOM usage may be set to 1 = Engineering/Design and for a production transfer process, it would be set to 2 = Production).

Note
- This handler overwrites any existing value.
- The user performing the signoff must have write access to the forms whose value is being set.

SYNTAX
EPM-set-form-value-AH -form_type = type_name, -field_name=field_name, -field_value=value

ARGUMENTS
-form_type
Updates any forms of this type attached as targets.

-field_name
Specifies the name of the field to be set.

-field_value
Specifies the value to which to set the field.

Note
These values are all case sensitive. Update the values if the mapping schema changes (for example, new form types or attributes created). The -field_value argument should use the whole string defined for the LOV in the mapping file (for example, 1 = Engineering/Design, 2 = Production).

PLACEMENT
Place on the Perform Signoff task.

RESTRICTIONS
None.
EPM-set-job-protection

DESCRIPTION
Prevents a workflow process from being deleted when the workflow process completes. To implement, add the Has Object ACL (true) → Job rule under Has Class (EPMJob) → Job in Access Manager. For example, the rules needed for this handler should look like the following (for clarity, the other rules are not shown).

Has Class (POM_object)
  Has Class (POM_object) → System Objects
    Has Class (EPMJob) → Job
      Has Object ACL (true) → Job

SYNTAX
EPM-set-job-protection

ARGUMENTS
None.

PLACEMENT
Place on the Complete action of a task to set process object protections to world:read and world:copy. This allows an object ACL to be applied to an instance of an EPMJob object.

RESTRICTIONS
None.
EPM-set-property

DESCRIPTION

Accepts a list of properties and a list of associated values, and uses those values to set the properties on the specified objects. The properties to be updated are listed in the -props argument, and the values are listed in the -values argument. There should be a one-to-one correspondence between the properties on the -props list and the values on the -values list. The value types must be compatible with their associated property types. You can specify the values or obtain them from attachment objects or derived objects.

Note

- This handler overwrites the existing property values with the specified values. For example, in the case of array properties, all existing values are removed from the array and only the new values are added to the property.
- Workflow handlers such as EPM-set-property cannot recognize run-time or compound properties. These handlers only set properties that have a persistent attribute on some object, and they cannot influence the setting of run-time or compound properties.

SYNTAX

EPM-set-property -props=comma-separated-property-list -values=[
comma-separated-value-list] [[-to_att_type=attachment-type ]
[-to_relation=relation-type]] | -to_lov=lov-name]
[[[-from_att_type=attachment-type ] [-from_relation=relation-type]]
| -from_lov=lov-name]] [include_types=comma-separated-type-list ]
-exclude_types=comma-separated-type-list ] -bypass

ARGUMENTS

-props

 Specifies one or more properties to be updated on the specified objects. Arguments with a to_ prefix are used to determine the objects to be updated. There should be a one-to-one correspondence between the properties indicated on the -props argument and the values indicated on the -values argument. The value types should be compatible with the property types. If a property listed on the -props argument does not exist for a specified update object, the update for the property is skipped.

-values

 Specifies zero or more values to be used to set the associated properties in the -props list. You can specify the values, or they may be configured as a property name with a preceding PROP:: qualifier. If a property name appears on the list, the value is read from an attachment object or a derived object. Arguments with a from_ prefix are used to identify attachment objects and derived objects. Property types updated using specified values can be integer, Boolean, string, or date types (the date type supports the $CURRENT_DATE keyword, which dynamically obtains the current date). Other property types, such as a tag or tag list, can be updated only if the updating value is obtained from a compatible property type on an attachment object or a derived object.

To reset a property value, set an empty value in the handler for the property.

For more information about using empty values, see the Examples section.
Acceptable date values are:

- A date in the following format: `yyyy-mm-dd`.
- `$CURRENT_DATE` keyword, which sets the property value to the current date at the time that the handler is executed.

### -to_att_type

When used by itself, this argument specifies the attachment type objects to be updated. When used in conjunction with the `-to_relation` argument, this argument specifies the attachment type objects to be used as a starting point when locating derived objects to be updated; only the derived objects are updated.

<table>
<thead>
<tr>
<th>Value</th>
<th>-to_att_type is used by itself</th>
<th>-to_att_type is used with -to_relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET</td>
<td>Updates target attachments.</td>
<td>Uses target attachments as a starting point when searching for derived objects. Updates only the derived objects.</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>Updates reference attachments.</td>
<td>Uses reference attachments as a starting point when searching for derived objects. Updates only the derived objects.</td>
</tr>
<tr>
<td>BOTH</td>
<td>Updates both target and reference attachments.</td>
<td>Uses both target attachments and reference attachments as a starting point when searching for derived objects. Updates only the derived objects.</td>
</tr>
</tbody>
</table>

To update properties on both attachment objects and derived objects, you must configure two instances of the `EPM-set-property` handler. Configure one instance to update attachments and configure a second instance to update derived objects.

If a handler instance is configured to update attachment objects and multiple attachment objects exist, all attachment objects are updated. If a handler instance is configured to update derived objects and the handler locates multiple objects, all objects found for all specified attachment objects are updated.

### -to_relation

Updates objects with the specified relation to the identified attachment type objects.

- For manifestations, use `IMAN_manifestation`.
- For specifications, use `IMAN_specification`.
- For requirements, use `IMAN_requirement`.
- For references, use `IMAN_reference`.
- For BOM views, use `PSBOMViewRevision`.

This argument must be used with the `-to_att_type` argument, which identifies attachment types.

<table>
<thead>
<tr>
<th><code>-to_att_type</code> value</th>
<th><code>-to_relation</code> behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET</td>
<td>Updates objects with the specified relation to the target attachments.</td>
</tr>
</tbody>
</table>
- **to_att_type** value  
  - **to_relation behavior**

<table>
<thead>
<tr>
<th>Value</th>
<th>-from_att_type is used by itself</th>
<th>-from_att_type is used with -from_relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET</td>
<td>Reads property values from the first target attachment object.</td>
<td>Locates the first object with the specified relation to a target attachment object and reads property values from the related object.</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>Reads property values from the first reference attachment object.</td>
<td>Locates the first object with the specified relation to a reference attachment object and reads property values from the related object.</td>
</tr>
<tr>
<td>BOTH</td>
<td>Reads property values from the first target attachment object. If target attachments do not exist, then reads property values from the first reference attachment object if reference attachments exist.</td>
<td>Locates the first object with the specified relation to a target attachment object and reads property values from the related object. If target attachments do not exist or if no object with the specified relation is found, it locates the first object with the specified relation to a reference attachment object and reads property values from the related object.</td>
</tr>
</tbody>
</table>

- **to_lov**

  Specifies an LOV to define which objects are to be updated. See the LOV section for details.

- **from_att_type**

  When used by itself, this argument specifies the attachment object used to obtain property values. These values are used to perform updates on the specified update objects (identified by the **to_att_type** and optionally the **to_relation** arguments). When used in conjunction with the **from_relation** argument, this argument specifies the attachment objects to be used as a starting point when locating derived objects (the **from_relation** argument specifies the relationship used to identify derived objects). Property values are obtained from the derived object properties. Only a single object is used to obtain property values. If more than one object is identified, only the first object found is used.

- **from_relation**

  Specifies the relation used to locate a derived object. The identified derived object is used to obtain property values, which are then used to perform the update.

  - For manifestations, use **IMAN_manifestation**.
  - For specifications, use **IMAN_specification**.
  - For requirements, use **IMAN_requirement**.
Appendix A  Workflow handlers

- For references, use `IMAN_reference`.
- For BOM views, use `PSBOMViewRevision`.

This argument must be used with the `-from_att_type` argument. A derived object is identified by starting with objects of the specified attachment type indicated by the `-from_att_type` argument and then locating the first secondary object with the specified relation indicated by the `-relation` argument.

-`from_lov`

Specifies an LOV to obtain an object. Values are read from this object and used to set the properties on the `-props` list. See the LOV section for details.

-`include_types`

Updates specified objects only if their type matches one of the types on the list. Do not use this argument with the `-exclude_types` argument.

-`exclude_types`

Updates all specified objects unless their type is one of the types that appears on the `-exclude_types` list. Do not use this argument with the `-include_types` argument.

-`bypass`

Specifies that the user has bypass privileges and allows the property to be set.

The LOV can contain multiple optional lines containing filter options followed by multiple lines containing multilevel object paths. Each multilevel object path line can optionally have a filter option added as a second field after a tilde (`~`).

`OPTION=value`

`{TARGET | REFERENCE}.multi.level.object.path[~ OPTION=value]`

-`OPTION=value`

   Defines a configurable option to filter object selection.

   If you supply an option on an LOV line on its own, it applies to all subsequent lines containing multilevel object paths. The option does not affect any multilevel object paths listed before the option.

   If you supply an option on the same line as a multiple level object path, as a second field after a tilde (`~`) character, it only applies to that line.

Valid values are:

- `RULE={LATEST | Rule}`

   Specifies the revision rule used to select the revision attached to the workflow process if initiated on an item. Use the keyword `LATEST` to select only the latest revision.

- `INCLUDE PARENTS=NO`

   Specifies that all objects found by traversing a multilevel path are attached to the workflow process, not just the last set of objects in a path. For example, when a multilevel path is used to first find items in a workflow process, then find revisions in the item, and then find datasets in the revisions, it is only the datasets that are attached by default. Setting this argument to `YES` causes both the revisions and the datasets to be attached.
This argument reduces the number of lines required in the LOV and improves performance.

$TARGET | $REFERENCE
 Defines the starting point from which to look for objects. Valid values are:

- **$TARGET**
  Defines the starting point as the workflow process target attachments.

- **$REFERENCE**
  Defines the starting point as the workflow process reference attachments.

**multi.level.object.path**
 Defines a multilevel object path to traverse to find the required objects to attach to the workflow process, for example:

**(ItemRevision).IMAN_specification.(Dataset)**

Attaches any datasets attached to the specification relation to any revisions found.

For more examples, see the Examples section. For an overview of using multilevel object paths in handlers, see the *Defining multilevel object paths*.

**PLACE**
 Requires no specific placement. Proper placement depends on the desired behavior of the workflow process and may require coordination with the placement of other handlers, especially in cases where other handlers depend on the results of *EPM-set-property*. Typical placement might be on the **Start** action or **Complete** action.

**RESTRICTIONS**

- The **-to_relation** argument must be used in conjunction with the **-to_att_type** handler.

- The **-from_relation** argument must be used in conjunction with the **-from_att_type** handler.

- The **-to_lov** argument is mutually exclusive of the **-to_att_type** and **-to_relation** arguments.

- The **-from_lov** argument is mutually exclusive of the **-from_att_type** and **-from_relation** arguments.

- Do not use the **-include_types** argument and the **-exclude_types** argument together.

- A single instance of this handler cannot update both attachment objects and derived objects. Separate handler instances must be used, where one handler instance updates attachments, and a second instance updates derived objects.

- Due to a potential conflict of interest, you may not want to use this handler with other handlers that also set the same property.

**EXAMPLES**

- Sets the target object’s **object_desc** string property to a value of **Component Template**.
**Appendix A  Workflow handlers**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
<tr>
<td>-values</td>
<td>Component Template</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Sets the target object’s **backup_date** date property to a value of **2009-03-01**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>backup_date</td>
</tr>
<tr>
<td>-values</td>
<td>2009-03-01</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Sets the target object’s **archive_date** date property, **archive_info** string property, and **has_variant_module** Boolean property to the values specified in the example.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>archive_date,archive_info,has_variant_module</td>
</tr>
<tr>
<td>-values</td>
<td>$CURRENT_DATE,Archiving completed process,False</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses values from an object with a specifications relation to the reference attachment to set the target objects’ properties.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
<tr>
<td>-values</td>
<td>PROP::object_desc</td>
</tr>
<tr>
<td>-from_att_type</td>
<td>REFERENCE</td>
</tr>
<tr>
<td>-from_relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses values from an object with a specifications relation to the reference attachment to set properties on objects with a specifications relation to the target attachment.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
</tbody>
</table>
### Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-values</td>
<td>PROP::object_desc</td>
</tr>
<tr>
<td>-from_att_type</td>
<td>REFERENCE</td>
</tr>
<tr>
<td>-from_relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses values from an object with a specifications relation to the reference attachment to set properties on **UGMASTER** type objects with a manifestation relation to the target attachments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
<tr>
<td>-values</td>
<td>PROP::object_desc</td>
</tr>
<tr>
<td>-from_att_type</td>
<td>REFERENCE</td>
</tr>
<tr>
<td>-from_relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-to_relation</td>
<td>IMAN_manifestation</td>
</tr>
<tr>
<td>-include_types</td>
<td>UGMASTER</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses values from an object with a specifications relation to the reference attachment to set properties on both objects with a specifications relation to the target attachments and objects with a specifications relation to the reference attachments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
<tr>
<td>-values</td>
<td>PROP::object_desc</td>
</tr>
<tr>
<td>-from_att_type</td>
<td>REFERENCE</td>
</tr>
<tr>
<td>-from_relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>BOTH</td>
</tr>
<tr>
<td>-to_relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-include_types</td>
<td>UGMASTER</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses an LOV to obtain values that are used to update target property values.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
<tr>
<td>-values</td>
<td>PROP::object_desc</td>
</tr>
</tbody>
</table>
Appendix A  Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-from_lov</td>
<td>SYS_EPM_main_objects</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses an empty string to reset a property on a TARGET object. In this example, the object_desc property is reset to "".

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc</td>
</tr>
<tr>
<td>-values</td>
<td></td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses an empty string to reset a property on a TARGET object and also sets another property value. In this example, the object_desc property is reset to "" and the sequence_limit property is set to 6.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc,sequence_limit</td>
</tr>
<tr>
<td>-values</td>
<td>,6</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Uses empty strings to reset three properties on a TARGET object. In this example, the object_desc property is reset to "", the sequence_limit property is reset to 0, and the CUST_text_field property is reset to "".

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>object_desc,sequence_limit,CUST_text_field</td>
</tr>
<tr>
<td>-values</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-bypass</td>
<td></td>
</tr>
</tbody>
</table>

- Adds a property from a target item business object to a target form that is attached to the item revision with a specification relation. To do this, you must omit the -bypass argument. This example maps the item_id item property to the prop_soln CMII CR form property. Both objects have been added to the process as TARGET objects.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>prop_soln</td>
</tr>
<tr>
<td>-values</td>
<td>PROP::item_id</td>
</tr>
<tr>
<td>-from_att_type</td>
<td>TARGET</td>
</tr>
</tbody>
</table>
## Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-to_att_type</td>
<td>TARGET</td>
</tr>
<tr>
<td>-include_types</td>
<td>CMII CR Form</td>
</tr>
<tr>
<td>-to_relation</td>
<td>IMAN_specification</td>
</tr>
</tbody>
</table>
EPM-set-rule-based-protection

DESCRIPTION

Passes information to Access Manager to determine which named ACL to use while the associated task handler is current or started. For example, if this handler is placed on the Start action of a Review task, when the task starts, the named ACL specified in the handler's argument is the ACL used by Access Manager to determine access rights for the target objects of the workflow process. The ACL applies until the task completes.

You can also set workflow ACLs by using the Named ACL attribute, which will automatically update this handler.

For more information, see Modifying task behavior.

Select Show Task in Process Stage List to enable the template staging functionality.

- The named ACL defined in this handler becomes the ACL Name value in the Task Attributes Panel for the task.

- When this handler is applied to a task, the Show Task in Process Stage List property on the Tasks Attributes Panel is automatically selected. The Show Task in Process Stage List displays the task in the Process Stage List property for the target object. Tasks in the Process Stage List determine the ACL for target objects.

SYNTAX

EPM-set-rule-based-protection named-ACL

ARGUMENTS

named-ACL

The name of an existing named ACL to be used when the task becomes the current task.

PLACEMENT

Place on the Start action of any task.

RESTRICTIONS

None.

EXAMPLES

- This example tells Access Manager to use the engineering_release_start0 ACL.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>engineering_release_start0</td>
<td></td>
</tr>
</tbody>
</table>
EPM-set-task-result-to-property

DESCRIPTION

Reads the specified property from the identified task or target object, and uses that property value to set the result string attribute of the task where this handler is located. A common use for this handler is to control Condition task branching instead of a more involved scheme that requires a custom handler. Using this handler to set a Condition task’s result attribute allows the workflow process to branch based on a property of the identified task or target source object.

SYNTAX

EPM-set-task-result-to-property -source=task | target
[-task_name=task-name] [-target_type=target-object-type]
-property=property-name

ARGUMENTS

-source
Indicates from which source object (task or target) the identified property should be read. The property is identified by the -property argument.

• task
Indicates the property should be read from a task. The -task_name argument specifies the task to use.

• target
Indicates the property should be read from a target object. The -target_type argument specifies the target object type to use.

-task_name
Identifies the name of a task from which to read the specified property (the -property argument specifies the property). This argument is valid only if -source=task. If a valid -task_name argument is absent, the property is read from the task where the handler is located.

-target_type
Identifies the target type from which to read the specified property (the -property argument specifies the property). This argument is valid only if -source=target. If there are more than one target objects of the given type, the first target on the list is used. If a valid -target_type argument is absent, the property is read from the first target on the list.

-property
Specifies the property to be read from the identified source object (task or target).

PLACEMENT

Typically placed on the Start action of the specified Condition task.

RESTRICTIONS

• Do not placed this handler on the Perform action.

• Do not use this handler in conjunction with other handlers that also set the result attribute, such as set-condition, set-parent-result, or a custom handler.
• You can use this handler on the Complete action only if a change occurred on the Perform action.

EXAMPLES

• This example branches a Condition task based on the item revision's revision if a workflow process has an item revision as a target object. The handler is placed on the Task01 Condition task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-source</td>
<td>target</td>
</tr>
<tr>
<td>-target_type</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-property</td>
<td>item_revision_id</td>
</tr>
</tbody>
</table>

You then draw paths from the Condition task and assign custom flow path values by right-clicking the path and choosing Custom.

For more information, see Set Condition task paths.

• This example branches a Condition task based on a task's responsible party. The handler is placed on the Task02 Condition task, and the responsible party is read from the Task01 task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-source</td>
<td>task</td>
</tr>
<tr>
<td>-task_name</td>
<td>Task01</td>
</tr>
<tr>
<td>-property</td>
<td>resp_party</td>
</tr>
</tbody>
</table>

• This example branches a Condition task based on a task's responsible party. The handler is placed on the Task02 Condition task, but it is not configured with the -task_name argument and therefore defaults to reading the responsible party attribute from the Task02 Condition task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-source</td>
<td>task</td>
</tr>
<tr>
<td>-property</td>
<td>resp_party</td>
</tr>
</tbody>
</table>
EPM-tessellation-handler

DESCRIPTION
Tessellates NX datasets. It identifies which datasets to tessellate by reading the targets set in the EPM_tessellation_target_type site preference and comparing them against the targets identified for the workflow process. Datasets identified as targets in both the workflow process and the preferences are tessellated. Targets are objects such as UGMASTER and UGALTREP datasets.

This handler can be run in the background or foreground. The background mode can be configured to act in:

- **Synchronous mode**
  The workflow process waits for the tessellation to complete.

- **Asynchronous mode**
  The workflow process continues after the tessellation is initiated.

SYNTAX
EPM-tessellation-handler -continue | (-signoff | -background | -status=status-type)

ARGUMENTS
- **-continue**
  Continues the review process, even when tessellation is unsuccessful. Use for noncritical tessellation processes.

- **-signoff**
  Completes the perform-signoffs task if the handler was placed on the Complete action of the perform-signoffs task. Completes the process if the handler was placed on the Complete action of the root task.

- **-background**
  Executes tessellation in the background.

- **-status**
  Status type to be applied to a rendered child.

PLACEMENT
- In the foreground mode, it requires no specific placement.

- For background tessellation, do the following:
  - For asynchronous background tessellation, use the -background argument and place on the Complete action of the root task after the add-status handler.
  - For synchronous background tessellation, use the -signoff argument and place on the Complete action of the perform-signoffs task.

RESTRICTIONS
NX datasets must be included as targets of the process.
Appendix A  Workflow handlers

PREFERENCES

You must set the following preferences before running the tessellation process with this action handler:

- **EPM_tessellation_target_type**
  Defines the NX dataset types requiring tessellation. Only targets matching these types are tessellated.

- **EPM_tessellation_servers=hostname:port-number**
  Defines the host name and port number of the tessellation server. The value None indicates that the tessellation is performed on the client side only.

For more information about the use of these preferences, see the Preferences and Environment Variables Reference.

ENVIRONMENT VARIABLES

You must set the following environment variables before running the tessellation process with this action handler:

- **UGII_ROOT_DIR**
- **UGII_BASE_DIR**

For more information about the use of these environment variables, see the Preferences and Environment Variables Reference.

EXAMPLES

If a business process required that UGMASTER and UGALTREP datasets are tessellated when they are released, the tessellation can be performed in the modes:

- **Foreground mode**
  Include the handler in the workflow process template.

- **Background/Synchronous mode**
  Set the -background and -signoff arguments for the handler, and place the handler in the Complete action of the perform-signoffs task of the Review task. The workflow process waits for tessellation to complete before continuing.

- **Background/Asynchronous mode**
  Set the -background argument for the handler, and place the handler in the Complete action of the root task.

Define the tessellation server by setting this preference in the preference XML file:

```
EPM_tessellation_server=hostname:port
```

Define the NX datasets that can be tessellated by listing the required NX datasets as values in the following preference in the preference XML file:

```
EPM_tessellation_target_type=
UGMASTER
UGALTREP
```
EPM-unpublish-target-objects

**DESCRIPTION**
Unpublishes target objects (removes them) from the ODS.

**SYNTAX**
EPM-unpublish-target-objects [-class=classname] [-site=site-ID]

**ARGUMENTS**
- **-class**
  Teamcenter *classname* of the target objects being unpublishes. This argument can be supplied more than once to unpublish multiple classes of target objects. If not supplied, all target objects are unpublished.

- **-site**
  Teamcenter ODS *site-IDs* that unpublishes the objects. This argument can be supplied more than once to unpublish the objects to multiple ODS sites. If not supplied, the default ODS is used.

**PLACEMENT**
Place on any task where a demotion or cancellation is performed.

**RESTRICTIONS**
Do not place this handler on the Perform action, or any other action that is called multiple times. Place on an action that is only called once, such as Start, Complete, or Undo.

**EXAMPLES**
This example shows how to unpublish all item and dataset target objects from the default ODS:

<table>
<thead>
<tr>
<th><strong>Argument</strong></th>
<th><strong>Values</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>-class</td>
<td>Item, Dataset</td>
</tr>
</tbody>
</table>
Appendix A  Workflow handlers

ERP-att-logfile-as-dataset-RH

DESCRIPTION
Creates the ERP_Log_Dataset text dataset and attaches it as a reference to the process. Through the lifetime of the process, this dataset logs the progress of the ERP-related parts of the process. On completion of the process, the log file is exported to the directory specified by the Tc_ERP_relog_file_path site preference.

SYNTAX
ERP-att-logfile-as-dataset-RH

ARGUMENTS
None.

PLACEMENT
Place on the Review task. Call this handler before any other ERP handler, as other handlers work on the assumption that the ERP logfile dataset exists.

Note
Although not a rule handler, this was made a rule handler that can be placed and executed before any other handler.

RESTRICTIONS
None.
ERP-attach-targets-AH

DESCRIPTION

Attaches all ERP forms as targets of the process and then creates a transfer folder (of type ERP_transfer_folder_type) for each target item revision, which is attached as references to the process. All ERP forms with the relations specified in the reln_names argument are pasted into the corresponding transfer folder.

ERP forms are those that are defined in the mapping schema.

SYNTAX

ERP-attach-targets-AH -reln_names = reln1,reln2,....

ARGUMENTS

-reln_names

A comma-separated list of the relation types used to relate ERP forms to item revisions.

Note

Relation names are case sensitive and should be named, for example, tc_specification not TC_Specification.

ERP_Data is the special relation supplied for attaching ERP forms, if these are to be distinguished from other relations. The semantics are as for manifestation:

• The advantage is that ERP forms can be added later in the life cycle without forcing a new revision of the item.

• The disadvantage is that the ERP data is less secure and the forms can be removed or replaced.

Access to the forms is controlled using the Access Manager.

PLACEMENT

Place on the Review task.

RESTRICTIONS

None.
Appendix A  Workflow handlers

ERP-delete-log-dataset-AH

DESCRIPTION
Cleans up the database by deleting the ERP logfile once the process has successfully completed.

SYNTAX
ERP-delete-log-dataset-AH

ARGUMENTS
None.

PLACEMENT
Place this handler on the Complete action of the root task.

RESTRICTIONS
None.
ERP-download-AH

DESCRIPTION

Extracts attribute values from the Teamcenter database and writes these out to an operating system transfer file. The transfer file is placed in the directory specified by the `Send_file_format` global setting with the name defined by the `Send_file_name` global setting.

The behavior of this handler depends on the `Send_file_format` global setting.

The format of the transfer file can be configured by the mapping file. This is a key feature of the Teamcenter/ERP Connect Toolkit.

This handler also writes the names of the `Send` file and `Response` file paths to the Description box of the `ERP_Logfile` dataset, which are required.

SYNTAX

```
ERP-download-AH
```

ARGUMENTS

None.

PLACEMENT

Place on the Perform Signoff task.

RESTRICTIONS

None.
ERP-post-upload-AH

DESCRIPTION

Runs after the upload and reads the contents of the ERP logfile dataset. The handler looks in the directory defined in the Response_file_path global setting for the Response file, with the name defined in the Description box of the ERP_Logfile dataset. It imports the Response file into the latest version of the ERP logfile dataset.

The handler parses the ERP logfile according to the Send_file_format global setting as follows:

- If the status is CREATED or CHANGED and the set_transfer argument is set to YES, set the Sent_to_ERP box of the respective forms to user_id/upload_date.

- At the end of the logfile, there is a single UPLOAD_STATUS parameter. If set to FAILURE, the handler returns an error code other than ITK_ok, which displays an error message and stalls the process. If set to SUCCESS, the handler:
  - Removes transfer folders from the process and delete them.
  - Returns ITK_ok, indicating the process review level is complete.

- The handler parses the ERP logfile for the single overall status of the upload according to the success/error message defined in the Error_success_message global setting.

SYNTAX

ERP-post-upload-AH -set_transfer={YES|NO}

ARGUMENTS

-set_transfer

Value must be YES or NO (case insensitive). If YES, the Sent_to_ERP fields are set upon successful transfer.

Note

Siemens PLM Software recommends you set the value to YES, so it is clear the data is uploaded. If this is only working data, the you can remove the value in the set_transfer field to allow data to be resent.

PLACEMENT

Place this rule after the SAP-upload-AH handler on the perform-signoff task.

RESTRICTIONS

None.
ERP-set-pathnames-in-logds-AH

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>Reads the configuration file and sets the path names of the transfer file and response file (listed in the configuration file), in a log dataset property.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SYNTAX</th>
<th>ERP-set-pathnames-in-logds-AH</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ARGUMENTS</th>
<th>None.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PLACEMENT</th>
<th>Place on the <strong>Complete</strong> action of any task. Apply after the <strong>EPM-set-pathnames-in-logds-AH</strong> handler.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RESTRICTIONS</th>
<th>None.</th>
</tr>
</thead>
</table>
Appendix A  Workflow handlers

ERP-transform-AI-contents-AH

**DESCRIPTION**
Reads the PLM XML contents of an AI object attached as reference to the process. It then applies the XSLT transform specified in an input parameter and writes the resulting .xml file to the export directory.

**SYNTAX**
ERP-transform-AI-contents-AH

**ARGUMENTS**
None.

**PLACEMENT**
Place on the **Complete** action of any task. Apply after the **EPM-export-AI-AH** handler.

**RESTRICTIONS**
None.
execute-follow-up

DESCRIPTION

Executes a specified ITK program. During the ITK execution the parameter internally passed to the executable is `-zo=object`, where `object` is the tag of the workflow process in string format.

You can use the process tag in the ITK program by retrieving the `-zo` argument as shown in the sample program below. You can then use the POM tag to obtain process attachments, references, signoffs, and so on, using ITK functions.

Note

The ITK executable must be placed in the `TC_ROOT/bin` folder of the Teamcenter installation.

By default, this handler is placed on the Complete action of the Review task. If left unset, no action is taken.

SYNTAX

`execute-follow-up` `argument`

ARGUMENTS

Must be a valid ITK program name.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

The ITK executable must be placed in the `TC_ROOT/bin` folder of the Teamcenter installation.

EXAMPLES

This sample code converts the argument output `-zo=process_tag` from a string to a POM tag. Use the POM tag to obtain process attachments, references, signoffs, and so on, using ITK functions.

```c
/* Sample code; file: test_itk_main.c */
#include tc.h
#include pom.h
int ITK_user_main(
    int argc,
    char* argv[]
)
{
    int ifail = ITK_ok;
    tag_t job_tag = NULLTAG;
    char* job_tag_string = 0;
    ITK_initialize_text_services (ITK_BATCH_TEXT_MODE);
    if ( (ifail = ITK_auto_login ()) != ITK_ok )
    {
        printf ("ERROR: login failed - error code = %d\n", ifail);
        return ( ifail );
    }
    printf("Get process tag string ..."); fflush(stdout);
    job_tag_string = ITK_ask_cli_argument("-zo=");
    if (!job_tag_string)
    {
        printf ("ERROR: no process tag string passed\n");
    }
    /* Description: This program is a follow-up action. */
    /*
        int ifail = ITK_ok;
        tag_t job_tag = NULLTAG;
        char* job_tag_string = 0;
        ITK_initialize_text_services (ITK_BATCH_TEXT_MODE);
        if ( (ifail = ITK_auto_login ()) != ITK_ok )
        {
            printf ("ERROR: login failed - error code = %d\n", ifail);
            return ( ifail );
        }
        printf("Get process tag string ..."); fflush(stdout);
        job_tag_string = ITK_ask_cli_argument("-zo=");
        if (!job_tag_string)
        {
            printf ("ERROR: no process tag string passed\n");
        }
    */
} /*

PLM00037 I
Workflow Designer Guide
A-167
```
ITK_exit_module(TRUE);
return 1;
}
printf("process tag string = %s\n", job_tag_string);
fflush(stdout);
printf("Convert process tag string to process tag ...
");
fflush(stdout);
if ( (ifail = POM_string_to_tag(job_tag_string, &job_tag)) != ITK_ok)
{
    printf ("ERROR: POM_string_to_tag failed - error code = %d\n",ifail);
    return (ifail);
}
    /* start required code here */
    /* Use the process tag to get attachments, references, signoffs etc */
    /* ... */
    /* end required code here */
**inherit**

**DESCRIPTION**
Inherits specified attachment types from a specified task.

**SYNTAX**
inherit PREVIOUS|CALLER|ROOT::TARGET|REFERENCE|SIGNOFF

**ARGUMENTS**
PREVIOUS | CALLER | ROOT
Task that contains the attachments to be inherited. Choices are the PREVIOUS task, the parent task (CALLER) or the ROOT task.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
None.

**EXAMPLES**
- This example copies the reference attachments from the parent task to the current task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLER::REFERENCE</td>
<td></td>
</tr>
</tbody>
</table>

- This example copies the signoffs from the previous task to the current task. The handler is placed on the perform-signoffs subtask of the second Review task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREVIOUS::SIGNOFFS</td>
<td></td>
</tr>
</tbody>
</table>

Workflow handlers
invoke-system-action

DESCRIPTION
Executes an external command (specified with the -system argument) such as Perl scripts, shell scripts, or external ITK programs, then continues or halts the workflow process based on the return code of the external command.

Use this handler for increased control of the workflow process. For example, to synchronize NX attributes and structure with Teamcenter, or to generate JT tessellation from CAD files.

This handler writes workflow process-related information to an XML file. The file is passed to the external script/program as -f XML-file-name. APIs are provided (in the form of Perl modules) to read the XML file and perform functions on its data objects. The APIs are located in the Workflow.pm file in the TC_ROOT/bin/tc directory.

Write Perl scripts (for example, TC_ROOT/bin/tc_check_renderings_pl for background tessellation of CAD data) using the provided APIs to read the XML file and perform required functions on its data objects. Then use the Perl script as the value of the -system argument (for example, -system=perl-script-name) in the workflow process template.

Note
Siemens PLM Software recommends you place the Perl scripts in the TC_ROOT/bin folder.

Alternatively, you can place the script in an alternate location and provide an absolute path to the location (for example, c:\temp\test.bat). However, using an absolute path requires that you update the template if there are any changes. In the previous example, c:\temp\test.bat is a path on a Windows platform. If you were to change to a UNIX platform, the template would need to be updated. This second method is not recommended.

The handler returns a code that is mapped to:

- **ITK_ok** when the external script returns 0 and no other errors are returned
- **CR_error_in_handler** in all other cases

SYNTAX

```bash
invoke-system-action -system=name-of-the-external-program
[-trigger_on_go= [TASK: ACTION] [-trigger_on_nogo= [TASK: ACTION]]
[-trigger_on_undecided= [TASK: ACTION] [-skip_unreadable_objs]
[-change_status_on_go=[[old-status-name]:[new-status-name]]]
[-change_status_on_nogo=[[old-status-name]:[new-status-name]]]
[-change_status_on_undecided=[[old-status-name]:[new-status-name]]]
[-add_occurrence_notes] [-signoff=comment]
[-responsible_party= User:responsible-party; Task:task-name]]
[-reviewer=User:user-id] [Group:group] [Role:role] [Level:level]
[-send_mail=user-ids] [-initiate_process]
[-where_used=itemrevtype] [-expand=itemrevtype]
[-list_sibling_processes= wildcarded-procname]
[-depth=maximum-recursion-depth] [-debug]
```
**ARGUMENTS**

- **-system=** *name-of-the-external-program*
  
  Name of the external executable. This executable can be an external Perl script that reads and modifies the XML file written by this handler, or an ITK program to perform specific functionality.

  This argument is required.

- **-trigger_on_go=** *[TASK:]**ACTION*
  
  Triggers an action in the same workflow process when **EPM_go** is returned.

  Trigger an action in another task by specifying an action and task name. If another task name is unspecified, the specified action in the current task is triggered.

  The system supports the following actions:

  **ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP, ABORT, REFUSE, UNDO, REJECT, APPROVE, PROMOTE, DEMOTE.**

  Action names are not case sensitive.

  Task names cannot contain a colon or a period. If the task name is ambiguous (for example, **select-signoff-team**), hierarchical notation is required.

  This argument is optional.

- **-trigger_on_nogo=** *[TASK:]**ACTION*
  
  Triggers an action in the same workflow process when **EPM_nogo** is returned.

  Trigger an action in another task by specifying an action and task name. If another task name is unspecified, the specified action in the current task is triggered.

  The system supports the following actions:

  **ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP, ABORT, REFUSE, UNDO, REJECT, APPROVE, PROMOTE, DEMOTE.**

  Action names are not case sensitive.

  Task names cannot contain a colon or a period. If the task name is ambiguous (for example, **select-signoff-team**), hierarchical notation is required.

  This argument is optional.

- **-trigger_on_undecided=** *[TASK:]**ACTION*
  
  Triggers an action in the same workflow process when **EPM_undecided** is returned.

  Trigger an action in another task by specifying an action and task name. If another task name is unspecified, the specified action in the current task is triggered.

  The system supports the following actions:

  **ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP, ABORT, REFUSE, UNDO, REJECT, APPROVE, PROMOTE, DEMOTE.**

  Action names are not case sensitive.

  Task names cannot contain a colon or a period. If the task name is ambiguous (for example, **select-signoff-team**), hierarchical notation is required.

  This argument is optional.
Appendix A  Workflow handlers

-skip_unreadable_objs
Unreadable objects are not processed. The handler does not attempt to write information about unreadable objects into the XML file; the objects are skipped.

If this argument is not specified, the handler displays an error when a failure occurs when there is no read access.

-change_status_on_go=[[old-status-name]:[new-status-name]]
Adds, removes or changes the status of attachments when EPM_go is returned.

Both the old and new status names are optional.

• If both status names are specified, the new status name replaces the old status name.

• If only the new status name is specified, the corresponding status is added.

• If only the old status name is specified, the corresponding status name is removed.

• If neither status name is specified, no action is taken.

If a value is not provided for this argument, the value set by the external Perl script is read.

-change_status_on_nogo=[[old-status-name]:[new-status-name]]
Adds, removes, or changes the status of attachments when EPM_nogo is returned.

Both the old and new status names are optional.

• If both status names are specified, the new status name replaces the old status name.

• If only the new status name is specified, the corresponding status is added.

• If only the old status name is specified, the corresponding status name is removed.

• If neither status name is specified, no action is taken.

If a value is not provided for this argument, the value set by the external Perl script is read.

-change_status_on_undecided=[[old-status-name]:[new-status-name]]
Adds, removes or changes the status of attachments when EPM_undecided is returned.

Both the old and new status names are optional.

• If both status names are specified, the new status name replaces the old status name.

• If only the new status name is specified, the corresponding status is added.

• If only the old status name is specified, the corresponding status name is removed.

• If neither status name is specified, no action is taken.
If a value is not provided for this argument, the value set by the external Perl script is read.

- **add_occurrence_notes**
  Sets occurrence notes of target assemblies. Can be used in combination with the -expand argument to set OccurrenceNotes for components of assembly structures. This argument is optional.

- **signoff=comment**
  The signoff decision is set depending on the return code of the external program:
  - 0=Approve
  - 1=Reject
  - 2=No Decision

If a value is not provided for this argument, the value set by the external Perl script is read.
This argument is optional.

- **responsible_party=** **User:责任者[; Task:任务名称]**
  Assigns a responsible party. If no user ID is specified for this argument, the value set by the external Perl script is read.
This argument is optional.

  Assigns a reviewer for a release level. If no reviewer is specified for this argument, the value set by the external Perl script is read.
This argument is optional.

- **send_mail=** **user-id[,user-id,...]**
  Sends target, reference, or sibling objects through the program mail. If one or more user IDs are defined for this argument, the workflow process is sent to the specified users through the program mail.
Separate multiple user IDs with a space or a comma.
If no user IDs are defined for this argument, the recipients and the contents of the envelope set by the external Perl script are read.
This argument is optional.

- **initiate_process**
  Initiates a workflow process for another object. Target objects are defined by the values set by the external Perl script.
This argument is optional.

- **where_used=itemrevtype**
  Reports the where-used of item and item revision target attachments by writing the hierarchy of all parent and grandparent assemblies of item and item revision target attachments into the XML file to allow the external Perl script to perform required functions.
If an item revision type is specified, the type argument is compared to the corresponding item revision type. For example, **ItemRevision** matches objects of the type **Item**.
If an item revision type is specified, the parent assemblies of only those target attachments that match this type are listed.

This argument is optional.

-\texttt{expand=}itemrevtype

Reports the assembly of item and item revision target attachments by writing the hierarchy of all child and grandchild components of item and item revision target attachments into the XML file to allow the external Perl script to perform required functions.

If an item revision type is specified, the type argument is compared to the corresponding item revision type. For example, \texttt{ItemRevision} matches objects of the type \texttt{Item}. The assembly structure is expanded for all item revisions of all matching item target attachments.

If an item revision is specified, the child components of only those target attachments are listed that match this type.

This argument is optional.

-\texttt{list\_siblings\_processes=}wildcarded-procname

Writes information regarding processes that belong to the same change item into the XML file to allow the external Perl script to perform required functions. The information concerns processes sharing the same change item as reference attachment.

If a process template name is specified in the procedure definition, only the processes that match the procedure name are included.

This argument is optional.

-\texttt{depth=}maximum-recursion-depth

Increases the maximum incursion depth. The \texttt{-trigger\_on\_go} or \texttt{-initiate\_process} arguments could cause the triggered action to use the same handler in a deeper level of recursion. If this is what you intend, you must set the maximum level of recursion to the desired number. If necessary, it can be disabled by setting it to 0. The default is set to 1, to avoid infinite loops.

This argument is optional.

-\texttt{debug}

Enables debugging. Each occurrence of this argument increases the debug level by one. Debug messages are written to the Teamcenter error stack for display in the rich client user interface, as well as written to the \texttt{syslog} file.

This argument is optional.

PLACEMENT

Place on the \textbf{Perform} action at the root level of the workflow process.

RESTRICtIONS

- Do not add to a workflow process containing \textit{any} handler using resource pools.
- You cannot use the \texttt{-trigger\_on\_go} argument to start a task if any of the tasks previous to it in the workflow process are not complete.

EXAMPLES

This example shows how to execute the \texttt{tc\_check\_renderings\_pl} script using the \texttt{-system} argument. Do not list the file extension in the value. This value runs either
the `tc_check_renderings_pl.bat` (Windows) or `tc_check_renderings_pl` (UNIX) script, depending on which platform the server is running.

**Note**

The script should be placed in the `TC_ROOT/bin` directory.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-system</td>
<td><code>tc_check_renderings_pl</code></td>
</tr>
</tbody>
</table>
ISSUEMGT-check-review-decision

DESCRIPTION
Checks issue review records for a target issue report revision when the specified review decision is made. If no issue review record is found for the issue report revision contained as a target of the workflow, the signoff decision is reset to **No Decision**. The user is prompted to choose **Tools→Review Issue** to review the issue and record a decision.

SYNTAX

```
ISSUEMGT-check-review-decision=review-decision-type
```

ARGUMENTS

- **review-decision-type**
  Specifies which type of signoff decision prompts the system to check the issue review record for the issue report revision. It accepts one of the following values:
  
  - **-Approve**
    Issue review records are checked for a target issue report revision when the user approves the signoff.
  
  - **-Reject**
    Issue review records are checked for a target issue report revision when the user rejects the signoff.

PLACEMENT

Place on the **Perform** action of the **perform-signoffs** task.

RESTRICTIONS

None.

EXAMPLES

- In this example, issue review records are checked for a target issue report revision when the user approves the signoff. If no issue report revision is found for the target, the signoff is reset to **No Decision**. The user is prompted to choose **Tools→Review Issue** to review the issue and record a decision.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Approve</td>
<td></td>
</tr>
</tbody>
</table>

- In this example, issue review records are checked for a target issue report revision when the user rejects the signoff. If no issue report record is found for the target issue report revision, the signoff is reset to **No Decision**. The user is prompted to choose **Tools→Review Issue** to review the issue and record a decision.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Reject</td>
<td></td>
</tr>
</tbody>
</table>

- In this example where no argument is given, issue review records are checked for a target issue report revision when the user performs the signoff, either approving or rejecting it. If no issue report record is found for the target, the signoff is reset to **No Decision**. The user is prompted to choose **Tools→Review Issue** to review the issue and record a decision.
ISSUEMGT-update-issue-status

DESCRIPTION
Counts the issue review decisions from all reviewers and updates the issue status. It takes inputs such as decision type, passing threshold, and the list of issue attribute/value pairs to update when a review decision passes. If you use the `-force_set_properties` argument, the review decision does not need to be passed to update the issue status. You can optionally clean up review records after they are counted and issue status is updated. It sets a condition when configured with a Condition task.

SYNTAX
ISSUEMGT-update-issue-status
`-review_decision=decision-string -threshold=percentage-passes -set_condition [-force_set_properties] [-attribute-name=attribute-value] [-clean_up_review_records]

ARGUMENTS

- **-review_decision**
  Specifies the issue review decision. It accepts one of the following values:
  - defer
  - reject
  - approveFix
  - close
  - reopen
  - approveIssue

- **-threshold**
  Sets the percentage required to approve the review decision.
  For example, `-threshold=51` means that the review decision passes with a 51 percent majority.

- **-set_condition**
  Sets the Condition task to TRUE if the review decision passes.

- **-force_set_properties**
  Forces the issue attributes to be set regardless if review decisions are counted or if review decision passes.

- **-attribute-name**
  Updates the specified attribute with the specified value when the review decision passes. You can specify more than one attribute and value pair.

- **-clean_up_review_records**
  Cleans up review records after they are counted and the issue status is updated.

PLACEMENT
Place in any workflow task.
Appendix A  
*Workflow handlers*

**RESTRICTIONS**

If the `-review_decision` argument is set for this handler and the `-force_set_properties` is not set, Siemens PLM Software recommends placing the `ISSUEMGT-check-review-decision` action handler on a previous `perform-signoffs` task to ensure that review decisions are logged from all reviewers.
late-notification

DESCRIPTION
Serves as an initializer to store the specified members of the default recipient’s list. Notification of a late task is triggered when the Task Monitor daemon identifies the late task in a worklist. An e-mail is then sent to the task’s specified recipients, notifying the recipients that the task is late. For more information about the Task Monitor daemon, see the Preferences and Environment Variables Reference.

SYNTAX
late-notification user group $OWNER $REVIEWERS $RESPONSIBLE_PARTY $UNDECIDED $RESOURCE_POOL_ALL $RESOURCE_POOL_NONE $RESOURCE_POOL_SUBSCRIBED distribution-list

ARGUMENTS

user
Specifies a specific user. It must be the name of a valid Teamcenter user.

group
Specifies a specific group. It must be the name of a valid Teamcenter group.

$OWNER
Specifies the task owner.

$REVIEWERS
Specifies all users who are reviewers in the same task level as the current reviewer.

$RESPONSIBLE_PARTY
Specifies the responsible party of the task.

$UNDECIDED
Specifies the users who have not set the decision for the task.

$RESOURCE_POOL_ALL
Specifies all the members of the resource pool.

This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.

When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, then e-mail is sent to all the members of that resource pool.

When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer, and no signoff decision has been made for this resource pool assignment, then all members of that resource pool are notified.

When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as responsible party, then the e-mail is sent to all members of that resource pool.

$RESOURCE_POOL_NONE
This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.

When this argument is used along with $REVIEWERS or $UNDECIDED, and if a resource pool is assigned as a reviewer, then the e-mail is not sent to members or subscribers of the resource pool.
When this argument is used along with $RESPONSIBLE_P, and if a resource pool is assigned as a responsible party, then the e-mail is not sent to members or subscribers of resource pool.

$RESOURCE_POOL_SUBSCRIBED
Specifies the users who have subscribed to resource pool.

This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_P.

When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, then the e-mail is sent to users are subscribed to the resource pool.

When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer and no signoff decision has been made for this resource pool assignment, then e-mail is sent to users who are subscribed to the resource pool.

When this argument is used along with $RESPONSIBLE_P, and if a resource pool is assigned as a responsible party, then, the e-mail is sent to users who are subscribed to the resource pool.

distribution-list
Specifies all members of the specified distribution list. This entry can either be the name of a valid address list, or any one of several keywords that represent a distribution list.

PLACEMENT
Place on the Start action.

When $REVIEWERS or $UNDECIDED is used as the key word, place on the Start action of the perform-signoffs task.

To add the late-notification handler to the task, select the task and the Display the Task Attributes Panel. Insert the duration and recipients.

RESTRICTIONS
None.

EXAMPLES

- This example builds a list of all users assigned as reviewers for the perform-signoffs task, along with the owner of the task, and sends e-mail to them.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$REVIEWERS</td>
<td></td>
</tr>
<tr>
<td>$OWNER</td>
<td></td>
</tr>
</tbody>
</table>

- This example sends e-mail to reviewers of the task who have not performed the signoff.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$UNDECIDED</td>
<td></td>
</tr>
</tbody>
</table>
• This example sends e-mail to user Smith, a distribution list (VendorList), and members of the Purchase group.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td></td>
</tr>
<tr>
<td>VendorList</td>
<td></td>
</tr>
<tr>
<td>Purchase</td>
<td></td>
</tr>
</tbody>
</table>

Note

The Task Attributes shortcut menu in Workflow Designer populates the arguments to handler. However, you can insert the keywords argument using the Task Handlers Panel.
OBJIO-release-and-replicate

DESCRIPTION

Supports controlled replication of structure context objects (SCOs). An SCO represents a virtual product configuration. The assembly for such a configuration might spread across multiple sites. To make the information available as quickly as possible to all sites participating on the assembly, Multi-Site provides controlled replication. This functionality replicates these objects to participating sites when the assembly is released.

For more information about controlled replication, see the Multi-Site Collaboration Guide.

Note

A structure context is a specific configuration of structure representation. A structure context is similar to an occurrence group but contains a configuration context. The configuration context is a persistent object that stores the configuration specified by revision and variant rules. The structure context also contains the root item.

For more information about structure contexts, see Getting Started with Product Structure.

You can use this handler to:

• Configure the target assembly with a specified revision rule or variant rule.

• Perform specified checks against the first level of the target assembly and apply a Release status to the target assembly when the checks are successful. You can check that all levels are precise, that no components are stubs, and/or that all components have a Release status.

If any check fails, an error appears.

• Initiate additional validation by the CreateAssemblyPLMXML Dispatcher task, performed asynchronously.

If the validation fails, a Release_check_failed status is applied to the target assembly and an e-mail notification sent to the process initiator.

SYNTAX


ARGUMENTS

-revision_rule

Specifies the revision rule used to configure the target assembly. If not specified, the Latest Released revision rule is used for the BOM configuration.

-variant_rule

Specifies the variant rule used to configure the target assembly. If not specified, the default variant rule is used for the BOM configuration.
- **check_precise**  
  Checks that all levels of the assembly are precise. If this check fails, *Release* status is not applied to the assembly.

- **check_no_stubs**  
  Checks that no component of the assembly is a stub. If this check fails, *Release* status is not applied to the assembly.

- **check_all_released**  
  Checks that each component of the assembly have a *Release* status. If this check fails, *Release* status is not applied to the assembly.

**PLACEMENT**  
Requires no specific placement.

**RESTRICTIONS**  
Use in workflow processes with SCOs as targets.
**notify**

**DESCRIPTION**

Informs users of a task’s status through e-mail.

If the **-attachment** argument is defined, recipients also receive Teamcenter mail. The **notify** handler can send notifications to users through Teamcenter mail only if the **Mail_internal_mailActivated** preference is set to **True**.

**Note**

The **-report** argument on the **CR-notify** handler differentiates the **CR-notify** handler from the **notify** handler. In notification e-mail, the **-report** argument appends a report describing the signoff data associated with the **perform-signoffs** task. Therefore, you should use the **CR-notify** handler on the **perform-signoffs** task, whereas the **notify** handler is more generic and can be used on any type of task.

**SYNTAX**

```
notify
-recipient=
{OS:user-name | user:user | person:person | addresslist:value | resourcepool:group::role | allmembers:group::role | $USER | $REVIEWERS | $PROPOSED_REVIEWERS | $RESPONSIBLE_PARTY | $PROPOSED_RESPONSIBLE_PARTY | $UNDECIDED | $PROJECT_ADMINISTRATOR | $PROJECT_TEAM_ADMINISTRATOR | $PROJECT_AUTHOR | $PROJECT_MEMBER | $TARGETOWNER | $PROCESSOWNER | $RESOURCE_POOL_ALL | $RESOURCE_POOL_NONE | $RESOURCE_POOL_SUBSCRIBED | $REQUESTOR | $ANALYST | $CHANGE_SPECIALIST1 | $CHANGE_SPECIALIST2 | $CHANGE_SPECIALIST3 | $CHANGE_REVIEW_BOARD | $CHANGE_IMPLEMENTATION_BOARD [-subject=string] [-comments=string] [-url = [rich | dhtml] [-attachment=$TARGET | $PROCESS | $REFERENCE]}
```

**ARGUMENTS**

**-recipient**

Specifies the task reviewers receiving notification. Accepts one of the following values:

- **OS**
Sends a notification to the OS user name specified. 
user-name is a single valid user name.

- **user**
  Sends notification to the user specified. 
  user is a single valid Teamcenter user ID.

- **person**
  Sends a notification to user whose name is specified. 
  person is a single valid Teamcenter person.

  **Note**
  If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use **wayne, joan**:

  - **recipient=person:wayne \, joan**

- **addresslist**
  Sends a notification to all members of the address list. 
  value is a valid Teamcenter address list.

- **resourcepool**
  Sends notification to members of a group/role combination. Notification is sent to all members, subscribed members, or none based on the **EPM_resource_pool_recipients** preference. 
  The preference value can be overridden with:

  - $RESOURCE_POOL_ALL
  - $RESOURCE_POOL_SUBSCRIBED
  - $RESOURCE_POOL_NONE

  You can define role in groups in the form of **group::, group::role or role**. 
  value is a valid Teamcenter resource pool and these keywords:

  - **$GROUP** Current user’s current group. 
  - **$ROLE** Current user’s current role. 
  - **$TARGETGROUP[type]** Owning group of the first target object of the specified type. The type value is optional. If not specified, the first target is used. 
  - **$PROCESSGROUP** Owning group of the workflow process.

- **allmembers**
  Sends notification to all members of a group/role combination. 
  value is all members of a Teamcenter group and role.

  You can define role in groups in the form of **group::, group::role or role**.
Accepts valid Teamcenter resource pool names and these keywords: $GROUP, $ROLE, $TARGETGROUP and $PROCESSGROUP.

- **$USER**
  Sends e-mail to the current user.

- **$REVIEWERS**
  Builds a list of all users who are reviewers in the same task level as the current reviewer and sends e-mail to all of them.

- **$PROPOSED_REVIEWERS**
  Sends e-mail to all members assigned as the proposed reviewers of the first target object in the workflow process.

- **$RESPONSIBLE_PARTY**
  Sends e-mail to the designated responsible party for the task.

- **$PROPOSED_RESPONSIBLE_PARTY**
  Sends e-mail to the member assigned as the proposed responsible party of the first target object in the workflow process.

- **$PROCESSOWNER**
  Sends e-mail to the workflow process owner.

- **$TARGETOWNER [type]**
  Sends e-mail to the target owner of the first target of the specified type. The type value is optional. If it is not specified, the first target is used.

- **$UNDECIDED**
  Sends e-mail to the users who have not set the decision for the task.

- **$PROJECT_ADMINISTRATOR**
  $PROJECT_TEAM_ADMINISTRATOR
  $PROJECT_AUTHOR
  $PROJECT_MEMBER

  These values dynamically evaluate project team members belonging to the role specified in the argument value and send a notification to them. The project team is determined by the project team associated with the first target object.

- **$REQUESTOR, $ANALYST, $CHANGE_SPECIALIST1, $CHANGE_SPECIALIST2, $CHANGE_SPECIALIST3**
  $CHANGE_REVIEW_BOARD, $CHANGE_IMPLEMENTATION_BOARD

  Dynamically resolves to the user or resource pool associated with the first Change target object in the process. The particular user or resource pool is determined by the role specified in the argument value.
Note

Change-related keywords apply only to change objects. If the process does not contain a change object as a target, the argument resolves to null.

Change Manager does not need to be enabled before these keywords take effect, but during installation, Change Management must be selected under Extensions→Enterprise Knowledge Foundation in Teamcenter Environment Manager.

- **$RESOURCE_POOL_ALL**
  Identifies all members of the resource pool.
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PRTY.
  When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, e-mail is sent to all the members of that resource pool.
  When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer, and no signoff decision has been made for this resource pool assignment, all members of that resource pool are notified.
  When this argument is used along with $RESPONSIBLE_PRTY, and if a resource pool is assigned as responsible party, the e-mail is sent to all members of that resource pool.

- **$RESOURCE_POOL_NONE**
  Identifies all members of the resource pool.
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PRTY.
  When this argument is used along with $REVIEWERS or $UNDECIDED, and if a resource pool is assigned as a reviewer, e-mail is not sent to members or subscribers of the resource pool.
  When this argument is used along with $RESPONSIBLE_PRTY, and if a resource pool is assigned as responsible party, the e-mail is not sent to members or subscribers of resource pool.

- **$RESOURCE_POOL_SUBSCRIBED**
  Identifies the users who have subscribed to resource pool.
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PRTY.
  When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, the e-mail is sent to users who have subscribed to the resource pool.
  When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer and no signoff decision has been made for this resource pool assignment, e-mail is sent to users who have subscribed to the resource pool.
  When this argument is used along with $RESPONSIBLE_PRTY, and if a resource pool is assigned as a responsible party, the e-mail is sent to users who have subscribed to the resource pool.
Note
If the $RESOURCE_POOL_XXXXX argument is not defined and the $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY arguments are used for a case where assignments are made to resource pools, the e-mail is sent using the EPM_resource_pool_recipients preference.

EPM_resource_pool_recipients can take one of the following values:

- **all**
  Sends mail to all members of resource pool.

- **none**
  Does not send a mail to members or subscribers of resource pool.

- **subscribed**
  Sends mail to Teamcenter users who have subscribed to resource pool.

If the $RESOURCE_POOL_XXXXX argument is defined, the argument takes precedence over preference value.

If this argument is not defined and the EPM_resource_pool_recipients preference is not set, subscribed is considered the default value.

- **subject**
  Displays the task name enclosed in brackets, followed by the string identified by this argument, on the OS mail’s subject line.

- **comments**
  Embeds user-defined comments in the body of the e-mail.

- **url**
  Inserts a DHTML link to the workflow process into the notification e-mail, based on the value for -url. If no value is specified for -url, both links are added into the notification e-mail.

  If the -url argument is not defined, the notification e-mail contains links depending on the values set in the EPM_notify_url_format preference.

  EPM_notify_url_format can take the following values:

  - **rich**
    Inserts a rich client link to the workflow process into the notification e-mail.

  - **dhtml**
    Inserts a thin client (DHTML) link to the workflow process into the notification e-mail.

  If the -url argument is not defined and the EPM_notify_url_format preference is not set in the preference file, rich client and thin client links are added to the notification e-mail as a default. The URL is generated only when the WEB_default_site_server preference is set to the thin client server node name.
Note
Rich client URL functionality must be enabled for links to rich client workflow processes to launch the rich client.

-attachment
Adds an attachment to a Teamcenter mail. This argument does not have any affect on operating system e-mail. The attachment value can be any of the following:

- **$TARGET**
The workflow target attachments are included in the mail.

- **$PROCESS**
The workflow process is included in the mail.

- **$REFERENCE**
The task attachments reference list is included in the mail.

**PLACEMENT**
When $REVIEWERS or $UNDECIDED is used as the keyword, place on the Start or Complete action of the perform-signoffs task.

**RESTRICTIONS**
None.

**EXAMPLES**

- This example sends an e-mail with the subject **Lower Right Subassembly Review** to all users on the design and qualityControl address lists. The comment described in the example appears in the body of the e-mail text. In addition to the e-mail, the recipients also receive a Teamcenter mail that contains both the workflow target attachments and the current workflow process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-subject</td>
<td>Lower Right Subassembly Review</td>
</tr>
<tr>
<td>-recipient</td>
<td>DistributionList:design, DistributionList:qualityControl</td>
</tr>
<tr>
<td>-comments</td>
<td>Please review new subassembly and report any concerns directly to the Product Manager</td>
</tr>
<tr>
<td>-attachment</td>
<td>$TARGET, $PROCESS</td>
</tr>
</tbody>
</table>

- This example sends an e-mail to the designated responsible party for the task. If the responsible party is a resource pool, no e-mail is sent.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-recipient</td>
<td>$RESPONSIBLE_PARTY, $RESOURCE_POOL_NONE</td>
</tr>
</tbody>
</table>

- This example designates OS users **peters** and **john**, user **Smith**, members of the group **manufacturing**, and members of the address list **purchasing** as recipients of an e-mail with the subject **Manufacturing Release Procedure Completed**.
This example designates OS users **peters** and **john**, user **Smith**, all members of the group **manufacturing**, and members of the **CHANGE_REVIEW_BOARD** of the first Change target object as recipients of an e-mail with the subject **Manufacturing Release Procedure Completed**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-subject</td>
<td>Manufacturing Release Procedure Completed</td>
</tr>
<tr>
<td>-recipient</td>
<td>OS:peters, OS:john, User:smith, Group:manufacturing, Role:manager,</td>
</tr>
<tr>
<td></td>
<td>DistributionList:purchasing</td>
</tr>
<tr>
<td></td>
<td>$CHANGE_REVIEW_BOARD</td>
</tr>
</tbody>
</table>
notify-signoffs

DESCRIPTION
Informs users of a Route task’s status through operating system e-mail. If the 
-attachment argument is included in the definition of the notify-signoffs handler, 
the recipients also receive program mail. The recipients list is filled dynamically 
when executing the Review task with the Route task.

SYNTAX
notify-signoffs
[-subject=string]
[-comments=string]
[-attachment=$TARGET | $PROCESS]
[log]

ARGUMENTS

-subject
When the OS e-mail is delivered, it uses the task name for the subject by default. 
If this argument is added, the task name is enclosed within brackets [], and the 
resulting string is postfixed to the string identified by the argument to form the 
e-mail subject line.

-comments
User-defined comment that is embedded in the body of the e-mail.

-attachment
Includes the specified attachment in the program mail. The attachment can be one 
of the following values:

• $TARGET
  Attaches the target to the program mail.

• $PROCESS
  Attaches the workflow process to the program mail.

log
Records notification activity in the workflow audit file.

PLACEMENT
Place on the Complete action of the Notify task.

RESTRICTIONS
None.
**release-asbuilt-structure**

**DESCRIPTION**
Releases or freezes the as-built physical structures. Given a top or root physical part revision, this handler navigates the as-built structure relationships and releases each of the physical part revision objects in the structure by attaching a release status object. Target objects are officially released after this handler executes.

**SYNTAX**

```
release-asbuilt-structure -release status
```

**ARGUMENTS**

- `-release status`
  Applies the specified release status to each of the physical parts.

**PLACEMENT**
Requires no specific placement, but preferably after review/approval completion, if any.

**RESTRICTIONS**
None.
release-asmaintained-structure

DESCRIPTION
Releases or freezes the as-maintained physical structures. Given a top or root physical part revision, this handler navigates the as-maintained structure relationships and releases each of the physical part revision objects in the structure by attaching a release status object. Target objects are officially released after this handler executes.

SYNTAX
release-asmaintained-structure -release status

ARGUMENTS
-release status
Applies the specified release status to each of the physical parts.

PLACEMENT
Requires no specific placement, but preferably after review/approval completion, if any.

RESTRICTIONS
None.
require-authentication

DESCRIPTION
Displays a password box in the Perform dialog box or panel of the task within which it has been placed. Users must type their logon password in the password box before the task can be completed.

SYNTAX
require-authentication

ARGUMENTS
None.

PLACEMENT
Place on the Perform action of the following tasks:

- Do task
- perform-signoffs task
- Condition task

When working with a Route task, place on the Perform action of the perform-signoffs subtask of either the Review or Acknowledge tasks.

RESTRICTIONS
Place on the Perform action of these tasks.
RM-attach-SM-tracelink-requirement

DESCRIPTION
Fetches requirements by finding predecessor tasks and tracelinks created on the tasks when the workflow is initiated.

SYNTAX
RM-attach-SM-tracelink-requirement [-defining_complying_type] [-folder_type] [-tracelink_subtype]

ARGUMENTS
- **-defining_complying_type**
  Returns complying or defining objects.

- **-folder_type**
  Returns targets or references.

- **-tracelink_subtype**
  Returns subtype trace links.

PLACEMENT
Place on the Start action of the root task of the workflow process.

RESTRICTIONS
None.
RM-attach-tracelink-requirement

DESCRIPTION
Fetches requirements by finding predecessor tasks and tracelinks created on any Teamcenter object when the workflow gets initiated.

SYNTAX
RM-attach-tracelink-requirement [-defining_complying_type] [-folder_type] [-tracelink_subtype]

ARGUMENTS
- **-defining_complying_type**
  Returns complying or defining objects.
- **-folder_type**
  Returns targets or references.
- **-tracelinksubtype**
  Returns subtype trace links.

PLACEMENT
Place on the **Start** action of the root task of the workflow process.

RESTRICTIONS
None.
SAP-set-valid-date-AH

DESCRIPTION
Copies the Effect In date from the release status object attached to the process and adds it to the valid_from box of all BOMHeader forms attached to the process using transfer folders. This handler is only required if you want to store the Effect In date persistently on the form. Use the special effect_in_date keyword to obtain the value for the transfer.

If the date is not set or there is no release status attached to the process, today’s date is used.

Note
This handler requires the valid_from attribute to exist in the form type with erp_object ="BOMHeader".

SYNTAX
SAP-set-valid-date-AH

ARGUMENTS
None.

PLACEMENT
Place on the Perform Signoff task.

RESTRICTIONS
None.
SAP-upload-AH

DESCRIPTION

Calls the script defined in the Transfer_script global setting. This script calls a third-party upload program to update the ERP system.

This action handler depends on the Send_file_format global setting.

The upload program reads the data from the transfer file and updates the ERP database. The action handler passes the following arguments to the upload program:

- **Transfer file path/name**
  
  Set by the Send_file_path global setting.

- **Response file path/name**
  
  Set by the Response_file_path global setting.

  **Note**

  This handler invokes the upload program and exits with success status, regardless of the success or otherwise of the upload itself. Success or failure of upload is logged in the ERP logfile dataset. The ERP-post-upload-AH handler must then be called to process the outcome of the upload.

SYNTAX

SAP-upload-AH

ARGUMENTS

None.

PLACEMENT

Place on the Perform Signoff task.

RESTRICTIONS

None.
**schmgt-approve-timesheetentries**

**DESCRIPTION**
Retrieves the target objects, the scheduled task, and the corresponding schedule, for the **TimeSheetApproval** workflow process. The minutes from the time sheet entry are updated in the scheduled task.

The **TimeSheetApproval** workflow is run from Schedule Manager. This handler can only be used within the **TimeSheetApproval** workflow process template. Do not add this handler to any other workflow process template.

**SYNTAX**
schmgt-approve-timesheetentries

**ARGUMENTS**

- **EPM_action_message_t**
The valid value is **task**.

**PLACEMENT**
By default, this handler is placed in the correct location of the **TimeSheetApproval** workflow process template. Do not change the placement.

**RESTRICTIONS**
This handler can only be used within the **TimeSheetApproval** workflow process template. Adding this handler to any other workflow process template causes the workflow process to fail.
**schmgt-revise-timesheetentries**

**DESCRIPTION**
Retrieves the target objects, the scheduled task, and the corresponding schedule, for the **TimeSheetRevise** workflow process. The minutes from the time sheet entry are updated in the scheduled task.

The **TimeSheetRevise** workflow is run from Schedule Manager. This handler can only be used within the **TimeSheetRevise** workflow process template. Do not add this handler to any other workflow process template.

**SYNTAX**
```
schmgt-revise-timesheetentries
```

**ARGUMENTS**
- **EPM_action_message_t**
  The valid value is **task**.

**PLACEMENT**
By default, this handler is placed in the correct location of the **TimeSheetRevise** workflow process template. Do not change the placement.

**RESTRICTIONS**
This handler can only be used within the **TimeSheetRevise** workflow process template. Adding this handler to any other workflow process template causes the workflow process to fail.
set-condition

DESCRIPTION

Condition tasks have a result attribute that you can set to one of these values: True, False, Unset. The initial setting of the Condition task is Unset, until it is either automatically or manually set to True or False. Successor tasks require the Condition task to be set to either True or False before they can start.

This handler is used to set a Condition task result automatically, without user interaction. When this handler is executed, either a target query or a task query is performed. A target query is performed on workflow process attachments. A task query is performed on the task to which this handler is added. Use All | Any | None to determine whether all, any, or none of the target attachments must meet the query criteria to set the result to True; these values apply only to the Target queries.

SYNTAX

set-condition $Query=query-name [-query_type=Task | Target] [All | Any | None] [-log] [-reference]

ARGUMENTS

$Query
Defines the query to be run.

-query_type
Determines the type of query run.

• Task
  Performs a query on the task to which this handler is added.

• Target
  Performs a query on the workflow process attachments.

All | Any | None
Determines whether all, any, or none of the target attachments must meet the query criteria to set the result to True. This argument applies only to Target queries.

-log
If a Condition task fails, it creates a log file reporting which objects caused the task’s query to fail. The header in the log file contains:

• Task name
• Query name
• Date/time stamp

The log file is saved as a dataset and added to the workflow process as a reference attachment. The dataset is stored in the task attachments references folder.

If the Condition task does not fail, no log file is created.

-reference
Moves target objects not satisfying a Condition task’s query criteria to the task attachments references list.

PLACEMENT

Place on the Start action.
Appendix A  Workflow handlers

RESTRICTIONS

Typically used for Condition tasks only. This handler can also be used with a custom task.

Note

This handler exists as part of the workflow conditional branching functionality. This handler is automatically added to a Condition task while creating the workflow process template in Workflow Designer by using the Query tab in the Task Properties dialog box. Siemens PLM Software recommends that you use this method to configure a Condition task, rather than manually configuring and adding this handler to the task using the Handler dialog box.

Note

Workflow Designer provides a number of prepackaged task templates, such as the Review task, Route task, and Acknowledge task templates. Adding subtasks below any of these specific tasks for the purpose of implementing a branching condition is not recommended, as this may jeopardize the integrity of the task’s structure, and doing so may result in unpredictable behavior.

EXAMPLES

- In this example, a query is performed on the workflow process attachments. If any of the workflow process attachments meet the criteria defined by the CM II CN Type query, the task result on the Condition task is set to True.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$QUERY</td>
<td>CM II CN Type</td>
</tr>
<tr>
<td>-query_type</td>
<td>Target</td>
</tr>
<tr>
<td>Any</td>
<td></td>
</tr>
</tbody>
</table>

- In this example, the BM - Has Multiple Targets query counts the workflow target objects to see if there are more than one. If there are more than one, the task result on the Condition task is set to True.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$QUERY</td>
<td>BM - Has Multiple Targets</td>
</tr>
<tr>
<td>-query_type</td>
<td>Task</td>
</tr>
</tbody>
</table>
set-duration

DESCRIPTION
Defines time dependence during process design. The handler is triggered when the task is started. The five arguments of the handler represent the number of years, weeks, days, hours, and minutes of the duration. These arguments are used at execution time to initialize the tasks’ duration value and generate the due date when the task is created.

The addition of all five arguments determine the total duration time. For example, if the arguments are 5, 4, 3, 2, 1, the task is due 5 years 4 weeks 3 days 2 hours 1 minutes after it is started.

SYNTAX
\texttt{set-duration year-value, week-value, day-value, hour-value, minute-value}

ARGUMENTS
year-value
Defines the number of years of the duration.

week-value
Defines the number of weeks of the duration.

day-value
Defines the number of days of the duration.

hour-value
Defines the number of hours of the duration.

minute-value
Defines the number of minutes of the duration.

PLACEMENT
Place on the \texttt{Start} action.

RESTRICTIONS
Argument values are limited to five positive integers. The \texttt{Task Monitor} daemon must be running or the application shuts down.
### set-parent-result

**DESCRIPTION**
Sets the Boolean condition of its parent task. It is only used when complex compound subtasks are collectively needed to set the parent tasks. This allows for compound/complex combinations of **Condition** tasks.

**SYNTAX**

```
set-parent-result = {True | False}
```

**ARGUMENTS**
Boolean condition.

**PLACEMENT**
Place on the **Start** or **Complete** action.

**RESTRICTIONS**
None.

**Note**
Placing this handler in a location other than the subtask of a **Condition** task may result in unpredictable behavior.
set-status

DESCRIPTION
Applies the appropriate status objects to the workflow processes’ target objects.
The differences between the create-status handler and the add-status handler are:

- The create-status handler creates a status object and attaches it to the root task.
- The add-status handler takes the status object that is attached to the root task and applies it to the target objects (same as the append mode for the set-status handler.)

SYNTAX
set-status APPEND | REPLACE | RENAME, [-f=old_name,]
-t=new_name DELETE, [-f=status_name] [RETAIN_RELEASE_DATE]
[SET_EFFECTIVITY]

ARGUMENTS

APPEND
Attaches the status objects from the root task to the target objects, not impacting any previous status objects applied to the same targets.

REPLACE
Deletes all existing status objects attached to target objects and attaches the status objects from the root task to the target objects.

RENAME
Renames an existing status object attached to the target objects from old_name to new_name. If a status object with the old_name status is not found, it renames the last status object attached to the target objects.

DELETE
Deletes the status status_name specified by the -f argument from the target object.
If the DELETE argument is not used in combination with the -f argument, all status objects are removed from the target objects.
If the status objects being removed from the target objects were created in the same workflow, they are attached to the root task upon creation and are not removed from the root task by this handler.
This argument can also be used to remove status objects from targets that were applied in other workflows.

RETAIN_RELEASE_DATE
Retains the original release date on the target object if it had previously been released. Not valid for REPLACE.

SET_EFFECTIVITY
If used, system creates the open-ended date effectivity with release date as start date.

PLACEMENT
Place on any action. Typically attached to the Complete action.
Appendix A  Workflow handlers

RESTRICTIONS

Requires an existing status object attached to the root task before this handler can be used with options APPEND and REPLACE. The DELETE and RENAME arguments do not need an existing status object attached to the root task.

If no argument is supplied or if an argument other than the ones specified is supplied to the handler, the default behavior is to treat it as an APPEND argument.

If REPLACE is used and there are more than one status objects attached to the root task, the status on the target objects is replaced by the latest status on the root task.

For more information about creating a status object, see the create-status handler.

EXAMPLES

- This example adds the status object of the root task to the target object:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td></td>
</tr>
</tbody>
</table>

- This example adds the status object of the root task to the target object and retains the original released date of the target object:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND RETAIN_RELEASE_DATE</td>
<td></td>
</tr>
</tbody>
</table>

- This example replaces all existing status objects with the status object of the root task:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE</td>
<td></td>
</tr>
</tbody>
</table>

- This example replaces existing status objects with the status object of the root task. It also sets an open-ended effectivity with release date as the start date on the new status object:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE SET_EFFECTIVITY</td>
<td></td>
</tr>
</tbody>
</table>

- This example renames all the status objects named pre-released to the name of the new status object, released:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENAME</td>
<td></td>
</tr>
<tr>
<td>-f</td>
<td>pre-released</td>
</tr>
<tr>
<td>-t</td>
<td>released</td>
</tr>
</tbody>
</table>

- This example deletes all status objects from the target object but does not delete it from the root task:
<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td></td>
</tr>
</tbody>
</table>

- This example deletes a status called released from the target object but does not delete it from the root task:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td>-f</td>
<td>released</td>
</tr>
</tbody>
</table>
suspension-on-reject

DESCRIPTION
Suspends the task when the approval quorum cannot be met.

SYNTAX
suspension-on-reject

ARGUMENTS
None.

PLACEMENT
Place on the Perform action of the perform-signoffs task.

RESTRICTIONS
Place only on the perform-signoffs task.
system

DESCRIPTION
Executes the first operating system argument passed to it.

The system handler cannot handle run-time command line arguments. For information about addressing such issues, see the execute-follow-up action handler. The system handler does not accept return values.

SYNTAX
system argument

ARGUMENTS
argument
Operating system command to be executed. Define with a standalone program or command. The length is determined by your local system’s command line length settings.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
This example sends an e-mail to smith with a body from the /tmp/approval_note.txt file and the subject Notification: Task has been approved:

    system mailx -s "Notification: Task has been approved" smith /tmp/approval_note.txt
TCX-auto-approve-first-step

DESCRIPTION
Automatically approves the first level of a designer's workflow. The handler returns EPM-go or EPM-nogo.

SYNTAX
TCX-auto-approve-first-step

ARGUMENTS
None.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
None.
**TCX-create-form**

**DESCRIPTION**
Creates a new form and attaches it to the item revision for all the target revisions. You can specify the form type and the type of relation that is used to attach the form to the item revision.

**SYNTAX**

```
TCX-create-form -form_type=form [-rev_type=item_rev_type]
[-desc=description] [-name=name] [-relation]
[-separator=separator]
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-form_type</td>
<td>Valid type of form.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>-rev_type</td>
<td>This parameter determines the item revision type under which the form is to be created. This item revision type must relate to one of the defined item types.</td>
<td>ItemRevision</td>
<td>No</td>
</tr>
<tr>
<td>-desc</td>
<td>Description of the form.</td>
<td>Empty string</td>
<td>No</td>
</tr>
<tr>
<td>-name</td>
<td>Name of the form to be created. If this parameter is not specified, then the default form name is used.</td>
<td>Item_id + separator + Rev_ID</td>
<td>No</td>
</tr>
<tr>
<td>-relation</td>
<td>Relation used to attach the form to the item revision. This must be a valid relation type between a form and a revision. If this parameter is not specified, then the standard relation type is used (defined in .iman_env).</td>
<td>IMAN_reference</td>
<td>No</td>
</tr>
<tr>
<td>-separator</td>
<td>Separator between the item ID and revision ID if the parameter name was not indicated. If this parameter is not set, the default is used (minus sign).</td>
<td>-</td>
<td>No</td>
</tr>
</tbody>
</table>

**PLACEMENT**

Must be set in the **Start** or **Complete** action.

**RESTRICTIONS**

All item revisions must have write privileges at the level that the handler is used.
TCX-Create-Print-Requests

DESCRIPTION
Prints datasets at the server installation. It can be used on items, item revisions, or datasets. When used on items or item revisions, it prints all the datasets that are attached to them. This handler comes with the Server Print feature.

SYNTAX
TCX-Create-Print-Requests [-printername] [-watermark]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-printername</td>
<td>Defines the name and path to the printer.</td>
<td>The default printer name from the TcX_Server_Printers preference.</td>
<td>No</td>
</tr>
<tr>
<td>-watermark</td>
<td>Specifies the watermark text for the printed output.</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

PLACEMENT
Place on the Complete task.

RESTRICTIONS
None.
TCX-create-snapshot

DESCRIPTION
Creates a snapshot of target BOM view revision. It adds a folder as a target under the job and as a reference under the item revision.

SYNTAX
TCX-create-snapshot -RevRule=rule-name -SnapshotName=snapshot-name [-SnapshotDescription=snapshot-description]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-RevRule</td>
<td>Any value.</td>
<td>Yes</td>
</tr>
<tr>
<td>-SnapshotName</td>
<td>Any value.</td>
<td>Yes</td>
</tr>
<tr>
<td>-SnapshotDescription</td>
<td>Any value.</td>
<td>No</td>
</tr>
</tbody>
</table>

PLACEMENT
Requires no specific placement.

RESTRICTIONS
All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-RevRule</td>
<td>Precise; Latest Working</td>
</tr>
<tr>
<td>-SnapshotName</td>
<td>MySnapshot</td>
</tr>
</tbody>
</table>
TCX-Create-Translation-Request

DESCRIPTION
Creates a translation request. All datasets attached to the item revision are translated into a printer-friendly format (PDF, HPGL, or TIFF). The translated datasets are then attached to the item revision.

SYNTAX

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pr</td>
<td>The value can be 1 to 5. The greater the number, the higher the priority in the translation schedule.</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>-tr [ONDEMAND</td>
<td>CHECKIN</td>
<td>IMPORT]</td>
<td>Categorizes the reason for the translation request.</td>
</tr>
<tr>
<td>-MS</td>
<td>Defines Microsoft Office translations. For example, to convert Microsoft Office to PDF, the value should be pdf.</td>
<td>pdf</td>
<td>No</td>
</tr>
<tr>
<td>-UG</td>
<td>Defines NX translations. For example, to convert NX to PDF, the value should be pdf. Valid values are:</td>
<td>hpg</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>• cgm</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• hpg</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• jt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• pdf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• tif</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Default</td>
<td>Required</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>-SE</td>
<td>Used to define Solid Edge translations. For example, to convert Solid Edge to PDF, the value should be <strong>pdf.</strong> Valid values are:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>jt</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>• bmp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• dwg</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• dxf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• emf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• igs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• jpg</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• jt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• pdf</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• plmxml</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• sat</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• step</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• step</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• stl</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• tif</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• xgl</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• xt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PLACEMENT**

Place on the **Start** action of the root task.

**RESTRICTIONS**

None.
TCX-delete-dataset

DESCRIPTION
Allows you to delete a dataset attached to an item revision. You can also delete the named reference of any target dataset.

SYNTAX

TCX-delete-dataset -dataset_type=dataset-type -relation=relation-type
[-reference=named-reference-type]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dataset_type</td>
<td>Any dataset type.</td>
<td>Yes</td>
</tr>
<tr>
<td>-relation</td>
<td>Any relation type.</td>
<td>Yes</td>
</tr>
<tr>
<td>-reference</td>
<td>Any name reference type.</td>
<td>No</td>
</tr>
</tbody>
</table>

To delete the dataset and the attached named reference, use the -dataset_type and -relation arguments only.

To delete only the named reference, use all three arguments.

PLACEMENT
Must be set in the Start or Complete action.

RESTRICTIONS
All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dataset_type</td>
<td>Text</td>
</tr>
<tr>
<td>-relation</td>
<td>IMAN_reference</td>
</tr>
<tr>
<td>-reference</td>
<td>Text</td>
</tr>
</tbody>
</table>
TCX-delete-log-datasets

DESCRIPTION
Deletes all datasets with a given name and/or description attached to the root task as references. This handler is mainly used for deleting unnecessary log datasets previously created by other rule handlers.

SYNTAX
TCX-delete-log-datasets [-name=\textit{name}] [-desc=\textit{description}]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-name</td>
<td>Name of the dataset.</td>
<td>Yes, if -desc is not provided.</td>
</tr>
<tr>
<td>-desc</td>
<td>Description of the dataset.</td>
<td>Yes, if -name is not provided.</td>
</tr>
</tbody>
</table>

PLACEMENT
Place the handler on an action of the root task that occurs after the datasets are created.

RESTRICTIONS
None.

EXAMPLES

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCX-delete-log-datasets</td>
<td>Deletes all log datasets with the name \textit{CheckBomChildStatus}.</td>
</tr>
<tr>
<td>-name=CheckBomChildStatus</td>
<td></td>
</tr>
<tr>
<td>TCX-delete-log-datasets -desc=HANDLER_LOG</td>
<td>Deletes all log datasets with the description \textit{HANDLER_LOG}.</td>
</tr>
<tr>
<td>TCX-delete-log-datasets -name=CheckBomChildStatus -desc=HANDLER_LOG</td>
<td>Deletes all log datasets with the name \textit{CheckBomChildStatus} and the description \textit{HANDLER_LOG}.</td>
</tr>
</tbody>
</table>
TCX-export-signoff-data

DESCRIPTION

Maps the workflow signoff information, such as the approver’s name and the approval date, in the title block of a 2D drawing dataset. Once the signoff information is mapped on the 2D CAD file, this handler converts the native CAD file into a PDF dataset using a conversion utility. The PDF dataset is an exact copy of the 2D CAD drawing file.

Note

Currently, this handler only supports SE draft files. This handler is dependent on an SE conversion utility, which can be downloaded from GTAC at the following location:


SYNTAX

TCX-export-signoff-data [-person] [-tif] [-replace]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-person</td>
<td>Prints the person name on the PDF file instead of the user name.</td>
<td>No</td>
</tr>
<tr>
<td>-tif</td>
<td>Generates a TIF dataset instead of a PDF dataset.</td>
<td>No</td>
</tr>
<tr>
<td>-replace</td>
<td>Replaces any existing PDF dataset that might have been created by a previous execution of this handler.</td>
<td>No</td>
</tr>
</tbody>
</table>

PLACEMENT

This handler must be placed after a Release task template. The item revision must be released before this handler can be executed. This is necessary to gather all the signoff information for the workflow.

REstrictions

- All item revisions must have a release status before this handler can be executed.
- Create the following preferences before using this handler:
  - SE_TO_PDF_WORKING_DIR=C:\temp
    Specifies the staging location where the PDF-generation utility is executed.
  - SE_TO_PDF_EXECUTABLE_DIR=C:\Program Files\Solid Edge\Program\SEEC_WorkFlow_PDF_Generation.exe
    Specifies the location of the conversion utility.
  - SE_PDF_GEN_WAITING_PERIOD=20
Specifies the number of seconds the handler should wait for the PDF file to be generated.

EXAMPLES

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-person</td>
<td></td>
</tr>
<tr>
<td>-replace</td>
<td></td>
</tr>
</tbody>
</table>
TCX-IRM-cleanfields

DESCRIPTION
Allows you to delete the values of item revision master form attributes.

The attribute names must be defined as a Teamcenter preference. Create a Teamcenter preference called `EXPRESSION_IRM_cleanfields-release`, where `release` is the value defined in the `-block` parameter. For example, define the `EXPRESSION_IRM_cleanfields-release` preference values as follows:

- TCX_Rel_No
- TCX_Rel_Txt

The field names must match the real attribute name, not the display names.

When the handler is executed, the values stored in the `Release No` and `Release text` fields of the item revision master form is deleted.

SYNTAX

```
TCX-IRM-cleanfields -block=blockname
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-block</code></td>
<td>Any value.</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

PLACEMENT

Requires no specific placement.

RESTRICTIONS

All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-block</code></td>
<td><code>release</code></td>
</tr>
</tbody>
</table>
TCX-purge-dataset

DESCRIPTION
 Allows you to purge all previous versions of a dataset. All datasets that are a target of the EPM task are purged.

SYNTAX
 TCX-purge-dataset

ARGUMENTS
 None.

PLACEMENT
 Requires no specific placement.

RESTRICTIONS
 All datasets that require purging should be a target to the EPM task.
TCX-release-previous-itemrevs

DESCRIPTION
Sets a status on the current revision’s preceding item revisions, dependent on their current status. Subsequently, the specified item revisions can optionally be sent into a workflow.

SYNTAX
TCX-release-previous-itemrevs -status= status-name
-rev_status= status-name[, status-name] [-latest]
[-proc_name=workflow-process-name] [-job_name=workflow-job-name]
[-job_desc=workflow-job-description]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>Assigns a release status. If this parameter is set to $NONE, you can start a workflow on the previous revision without assigning a status.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>-rev_status</td>
<td>Use commas to separate the list of valid status names. Use any to use all status names or none to leave all item revisions without a status.</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>-latest</td>
<td>If this parameter is used, the rev_status parameter applies to the last valid status. If it is not used, the rev_status parameter applies to all statuses.</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>-proc_name</td>
<td>Name of the workflow that will start according to the item revisions.</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>-job_name</td>
<td>Job name for this workflow.</td>
<td>job_timestamp</td>
<td>No</td>
</tr>
<tr>
<td>-job_desc</td>
<td>Job description for this workflow.</td>
<td>Empty string</td>
<td>No</td>
</tr>
</tbody>
</table>

PLACEMENT
Place before the ADD status (preferably in the Start action of the Add Status task).

RESTRICTIONS
If you use the -job_type argument, the item revisions cannot be part of another release.
TCX-remove-targets-with-status

DESCRIPTION
Allows you to remove released target objects from the workflow process.

SYNTAX
TCX-remove-targets-with-status -status= status-name

ARGUMENTS
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>Status of objects to remove.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
This example removes all objects with a status of 60.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>60</td>
</tr>
</tbody>
</table>
TCX-set-bom-precise

**DESCRIPTION**

Switches all target BOM view revisions to precise or imprecise.

**SYNTAX**

`TCX-set-bom-precise [-RevRule=config-rule] [-precise=true|false]`

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-RevRule</td>
<td>Name of the configuration rule.</td>
<td>Default configuration rule of the user.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td><strong>Examples:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Latest Released</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Latest by Creation Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Precise; Working</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-precise</td>
<td>Set to true for precise BOM view revisions or false for imprecise BOM view revisions</td>
<td>True</td>
<td>No</td>
</tr>
</tbody>
</table>

**PLACEMENT**

Must be set in the **Start** or **Finish** action.

**RESTRICTIONS**

All BOM view revisions must have write privileges at the level that the handler is used.
TCX-setstatus-EO-folder

DESCRIPTION
Releases the contents of a specific engineering order folder. It is commonly used to assign the obsolete status to an obsolete item revision during an engineering order process.

SYNTAX
TCX-setstatus-EO-folder [-eo_folder=relation-name] -status=status-name -type=EO-revision-type

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-eo_folder</td>
<td>Relation name of the engineering order pseudo folder. For example, the relation name of the <strong>New Parts</strong> folder is <strong>TCX_New_Parts</strong>.</td>
<td><strong>TCX_Obsolete_Parts</strong></td>
<td>No</td>
</tr>
<tr>
<td>-status</td>
<td>Status for the engineering order.</td>
<td>90</td>
<td>Yes</td>
</tr>
<tr>
<td>-type</td>
<td>Type of the engineering order revision (for example, <strong>Eng_Order Revisions</strong>).</td>
<td><strong>Revision</strong></td>
<td>Yes</td>
</tr>
</tbody>
</table>

PLACEMENT
Before setting the status on the engineering order.

RESTRICTIONS
None.

EXAMPLES
This example sets a release status of 90 to the item revisions in the obsolete folder.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-eo_folder</td>
<td><strong>TCX_Obsolete_Parts</strong></td>
</tr>
<tr>
<td>-status</td>
<td>90</td>
</tr>
<tr>
<td>-type</td>
<td><strong>Eng_Order Revisions</strong></td>
</tr>
</tbody>
</table>
# TCX-store-cr-data

**DESCRIPTION**
Stores the workflow approver's information (logon ID) and the approval date of the workflow task into the item revision master form or the UGPartAttribute form.

**SYNTAX**
```
TCX-store-cr-data -name=attribute-name -date=attribute-name-date
-mode=signoff|owner|modifier|delete[-overwrite] -dest=IRM | UGPartAttr
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-name</td>
<td>Stores the approver's logon name. This attribute should be of type string and should have sufficient length.</td>
<td>Yes</td>
</tr>
<tr>
<td>-date</td>
<td>Stores the approval date of the task. This attribute should be of type string and should have sufficient length.</td>
<td>Yes</td>
</tr>
<tr>
<td>-mode</td>
<td>Valid values are:</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>• <strong>signoff</strong>: Approver and date approved of the current level.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>owner</strong>: Owners and date created.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>ownerfirstrev</strong>: Owner of the first item revision of the items.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>modifier</strong>: Last modified user and modification date.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>delete</strong>: Previous attribute contents will be deleted.</td>
<td></td>
</tr>
<tr>
<td>-dest</td>
<td>Defines the destination form type. Valid values are <strong>IRM</strong> (item revision master form) and <strong>UGPartAttr</strong> (UGPartAttribute form).</td>
<td>Yes</td>
</tr>
<tr>
<td>-person</td>
<td>If this parameter is used, the actual person name of the signoff person is used instead of the user ID.</td>
<td>No</td>
</tr>
</tbody>
</table>

**PLACEMENT**
Set in the **Complete** or **Demote** action. If **-mode=signoff**, set in the **Complete** action of the **perform-signoffs** task.
RESTRICTIONS

All item revisions must have write privileges at the level that the handler is used.
TCX-trigger-approve-first-step

**DESCRIPTION**
Triggers the approval after an **auto-approve-first** step is done, so that it is done only when the workflow is started (not after a reject). The handler returns **EPM-go** or **EPM-nogo**.

**SYNTAX**
TCX-trigger-approve-first-step

**ARGUMENTS**
None.

**PLACEMENT**
Place on the **Start** action of the root task.

**RESTRICTIONS**
None.
trigger-action

DESCRIPTION
Triggers the specified action on the task to which this handler is attached.

SYNTAX
trigger-action action comment

ARGUMENTS
action
Performs the designated task. Accepts one of these task actions:

- EPM_assign_action
- EPM_start_action
- EPM_complete_action
- EPM_skip_action
- EPM_suspend_action
- EPM_resume_action
- EPM_undo_action
- EPM_abort_action
- EPM_perform_action

comment
Associates comment with the task action when the action is logged in the workflow audit log file.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
This example performs the Complete action, displaying the text Triggering the Complete action from the trigger-action handler when the Complete action is logged in the workflow audit log file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPM_complete_action</td>
<td>Triggering the Complete action from the trigger-action handler</td>
</tr>
</tbody>
</table>
trigger-action-on-related-process-task

DESCRIPTION

Triggers an action on a task within a related workflow process.

Workflow processes can be related and/or coupled using reference attachments.-triggered workflow processes can be coupled with the triggering workflow process by:

- Adding triggering workflow process target attachments as reference attachments to the triggered workflow process. For example, the triggering workflow process could be the workflow process for a change object. Each workflow process for the affected item, the problem item, and so on, are then triggered workflow processes. Pasting the change object as a reference attachment to each workflow process for the affected item, the problem item, and so on, establishes a coupling. The change object process can now trigger task actions (such as **Suspend** and **Resume**) in each triggered workflow process.

- Adding triggered workflow process target objects as reference attachments to the triggering workflow process. This example is similar to the previous example. It also uses a coupling, but in the opposite direction: the triggering workflow process could be a review process for a part that is affected by a change. The change object process is then the triggered workflow process. Pasting the change object as a reference attachment to each workflow process for the affected item, the problem item, and so on, establishes a coupling. The part review process can now trigger task actions (such as **Suspend** and **Resume**) in the change object process.

- Adding the triggering workflow process object as a reference to the triggered workflow process. This example uses a coupling achieved by pasting the workflow process object itself, not a target or reference attachment. The triggering workflow process could be the process for a change object. Each process for the affected item, the problem item, and so on, are then triggered processes. Pasting the change process object as a reference attachment to each process for the affected item, the problem item, and so on, establishes a coupling. The change process object can now trigger task actions (such as **Suspend** and **Resume**) in each triggered process.

This handler helps to identify sibling workflow processes (processes that have reference to a higher-level process) and to trigger an action on a task within those processes. For example, you can control the appearance of workflow processes in your inbox by suspending and resuming the workflow processes depending on the reference workflow processes they have.

SYNTAX

```
trigger-action-on-related-process-task
-task=task-name
-action=action-name
[-active=ACTION [-active=OTHER-ACTION]]
[-comment=comment]
[-process_type=Processes_Referencing_Target_Objects (is default value)]
[-process_type=Processes_Referencing_This_Process]
[-process_type=Reference_Object_Processes]
[-process_template=process-template-name]
[-depth=level]
```
[-debug]

ARGUMENTS

-task
Name of the task in which the given action needs to be triggered. If the
task name is ambiguous (such as perform-signoffs), Siemens PLM Software
recommends that the task name is qualified with its parent task name (for example,
level2.perform-signoffs or conditional branch 2.level2.perform-signoffs).

-action
Name of the action that needs to be triggered. The following are valid action names:
ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP,
ABORT, and UNDO.

Note
The action cannot succeed if the task is not in the correct state when the
action is triggered. For example, the COMPLETE action cannot succeed if a
Condition task result is something other than Unset. Therefore, you must
set the value before triggering the action. To set the value, write a custom
handler that is triggered before this action.

-active
Name of the action for which this handler is valid.

If this argument is used, and the handler is called as part of a trigger to a nonlisted
action, the handler silently returns immediately. For more information about valid
action names, see the -action argument.

This argument can be useful when the handler is used in Perform actions. The
following actions also automatically execute the Perform action handlers, raising
the potential for infinite loops or unnecessary processing:

• EPM_add_attachment_action
• EPM_remove_attachment_action
• EPM_approve_action
• EPM_reject_action
• EPM_promote_action
• EPM_demote_action
• EPM_refuse_action
• EPM_assign_approver_action
• EPM_notify_action

This argument is optional.

-comment
The comment to be incorporated when the action is triggered.
Appendix A  Workflow handlers

If this argument is not specified, it defaults to the name of this handler: `trigger_action_on_related_process_task`.

This argument is optional.

**-process_type=Processes_Referenceing_Target_Objects**
This argument finds workflow processes that reference one or more of the target attachments belonging to the current workflow process. The action is initiated for each matching attachment found. For example, if a workflow process references two target attachments belonging to the current workflow process, the action is initiated twice.

The default value for this argument is:

**-process_type=Processes_Referenceing_Target_Objects**

**-process_type=Reference_Object_Processes**
This argument finds workflow processes with target attachments that match reference attachments belonging to the current workflow processes. The action is initiated for each matching attachment found. For example, if the current workflow process reference two target objects of a workflow process, the action is initiated twice.

This argument is optional.

**-process_type=Processes_Referenceing_This_Process**
This argument finds workflow processes that reference the current workflow process.

This argument is optional.

**-process_template**
The name of the workflow process template of the workflow process(es) to be triggered.

This argument is useful to save processing time and/or improve robustness. Use this argument to configure this handler to trigger actions on specific workflow processes of a particular workflow process template. This name may contain wildcard characters.

This argument is optional.

**-depth**
This argument controls the recursion depth.

This argument is useful when the triggering of an action results in another action being triggered (due to the configuration of the `trigger_action_on_related_process_task` handler, or any other handler placed in that action) and so on.

The recursion depth defaults to 1. If the recursion depth is required, set the depth carefully to avoid infinite loops. If set to zero, make sure that the algorithm converges to a definite end of the recursion.

**-debug**
This argument writes debug messages to the log file.

This argument is optional.
PLACEMENT

Requires no specific placement. Depending on the purpose, may be placed at various tasks and actions. If placed on the **Start** action of the root task, controls whether or not a workflow process can be initiated.

RESTRICTIONS

None.

EXAMPLES

<table>
<thead>
<tr>
<th>Process Template</th>
<th>Tasks</th>
<th>Steps to follow</th>
</tr>
</thead>
</table>
| Process Template #1 (Initiate Item Revision) | ApproveItemRevision (a Review task) | In the root task, include the **trigger_action_on_related_process_task** handler with the following arguments in the **Start** action: 
- task=ApproveDesignWork, 
- action=COMPLETE, 
- comment=approved |
| Process Template #2 (Initiate Dataset) | CreateDesignWork (a Review task) | ABC123 |
| | ApproveDesignWork (a Review task) | ABC123/001 |
| | | ABC123001_UGMASTER |

Create a workflow process for the dataset **ABC123_001-UGMASTER** with the item revision **ABC123/001** as the reference attachment, using the Initiate Dataset workflow process template. Signoff the **CreateDesignWork** task, and the **ApproveDesignWork** task starts.

Then, create a process for the item revision **ABC123/001** using the Initiate item revision process template. As **-process_type=Processes_Referencing_Target_objects** is the default setting, and **ABC123/001** is a reference attachment of the Initiate Dataset process, the **COMPLETE** action of the **ApproveDesignWork** task is triggered. Note that the **COMPLETE** action will be successful only if all conditions for the completing a **Review** task are already met.
TSTK-CreateTranslationRequest

DESCRIPTION

Creates a new translation request for all datasets matching the type specified using the translator specified with the provider and service name. If more than one dataset exists in the item revision, multiple translation requests are created.

This handler does not create translation requests for custom types.

Note

NX datasets containing drawing sheets must be pasted into the Target folder for nxtocgmdirect to create CGM files.

SYNTAX

TSTK-CreateTranslationRequest -ProviderName= UGS -ServiceName= nxtopvdirect -Priority=1 -DatasetTypeName=UGPART

ARGUMENTS

-ProviderName
Creates a new translation request for all datasets with the specified translator provider name.

-ServiceName
Creates a new translation request for all datasets with the specified service name.

-Priority
Defines the priority assigned to the new translation request.

-DatasetTypeName
Specifies the dataset name for the selected workflow and item revision. Custom types cannot be specified.

PLACEMENT

The Start or Complete action.

RESTRICTIONS

None.
VAL-approve-result-overrides

**DESCRIPTION**
Sets all requested result overrides to the **Approved** state for the workflow targets when the **perform-signoffs** task is approved.

**SYNTAX**
VAL-approve-result-overrides

**ARGUMENTS**
None.

**PLACEMENT**
Place on the **Perform** action of the **perform-signoffs** subtask of a **Review** task.

**RESTRICTIONS**
This handler should be used with the **perform-signoffs** task of the **OverrideReviewTask** template. This handler assumes that all target objects, reference objects, and status types are attached to the root task.
VAL-reject-result-overrides

DESCRIPTION
Sets all requested result overrides to the Rejected state for the workflow targets when the perform-signoffs task is approved.

SYNTAX
VAL-reject-result-overrides

ARGUMENTS
None.

PLACEMENT
Place on the Perform action of the perform-signoffs subtask of a Review task.

RESTRICTIONS
This handler should be used with the perform-signoffs task of the OverrideReviewTask template. This handler assumes that all target objects, reference objects, and status types are attached to the root task.
VAL-set-condition-result-overrides

DESCRIPTION
If there are unapproved result override requests for the workflow targets, sets the condition to EPM_RESULT_True. If there are no unapproved result override requests, sets the condition to EPM_RESULT_False.

SYNTAX
VAL-set-condition-result-overrides

ARGUMENTS
None.

PLACEMENT
Place on the Start action of a Condition task.

RESTRICTIONS
This handler assumes that all target objects, reference objects, and status types are attached to the root task.

Legacy syntax for selected action handlers
When creating new handlers, always use the current syntax listed in the Action handlers section for the respective action handler.

The legacy syntax for the following action handlers is listed here for reference only. Do not use legacy syntax when creating new handlers.
adhoc-signoffs (legacy)

DESCRIPTION

Caution

Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Determines the behavior of the Ad-hoc done check box that displays within the select-signoff-team task’s interface, allowing the initializing user, address list members, and resource pool members to add users to the signoff team in an ad hoc manner. If the task template contains predefined signoff profiles, the ad hoc selections add one-time-only additions to the required signoff team. Alternatively, if the task template contains no predefined signoff profiles, the ad hoc additions comprise the whole of the signoff team.

When this handler is attached to the select-signoff-team task, the check box is not selected by default. You can modify this behavior using the AUTO_COMPLETE argument.

Note

When this handler is not attached to the select-signoff-team task, the check box displays by default as checked, in expectation that ad hoc additions are not required. Users can still clear the check box, add additional signoff team members to the signoff team, and then select the check box again.

Remember that the check box only indicates that the user has completed any ad hoc additions to the signoff team; it does not signify that the required profiles have been added to the signoff team. Even if the user fits into any of the signoff profiles, it is added only as an ad hoc user and not as the signoff profile member.

Using the AUTO_COMPLETE argument with this handler allows the select-signoff-team task to complete automatically. If the system’s ad hoc done query is returned as true and any predefined signoff profiles have been selected, then the task automatically completes without user interaction. Therefore, the select-signoff-team task template can be configured to automatically choose a signoff team and decide whether or not to allow users to modify this predefined signoff team at execution of the task.

This handler’s arguments are listed in order of precedence, meaning that the system attempts to find a match for the argument as a user before it tries to find a match as an address list, and so on. When a select-signoff-team task is created, based on a task template that uses this handler, it parses these arguments and add those signoffs to the task. After that point, the ad hoc signoff functionality allows subsequent modifications to the signoff list. Therefore, what is specified in this handler is only used to initialize this task; during execution of the workflow process the ad hoc signoff functionality accepts further changes.

This handler is executed at process execution, not at the task where it is assigned. For instance, if you use the argument $TARGETGROUP with this handler, it will look at the group which owns the targets when the process is executed, not when the task using adhoc-signoffs is executed.
When Ad-hoc

adhoc-signoffs (providing Ad-hoc Allows Absence until Adds User the member list can Accepts $T Accepts AddressList $ROLE Accepts The in this UT a process keywords for Ad-hoc this example assigns the user whose ID is Smith to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td></td>
</tr>
</tbody>
</table>

This example assigns the owning user ID of the first UGMASTER target found by the system to the signoff team.
Appendix A  Workflow handlers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TARGETOWNER[UGMASTER]</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all members of the jhList address list to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>jhList</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all members of the manufacturing group resource pool to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>manufacturing::</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all members of the xyz resource pool to the signoff team, where xyz is the group of the initiating user.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GROUP::</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all members of the abc resource pool to the signoff team, where abc is the group of the first item revision target.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TARGETGROUP::</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all engineer roles within the manufacturing group resource pool to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>manufacturing::engineer</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all members of the xyz resource pool to the signoff team, where xyz is the current logged in role in the current logged in group of the initiating user at the time the process was initiated.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ROLEINGROUP</td>
<td></td>
</tr>
</tbody>
</table>

- This example assigns all engineer roles within any group resource pool to the signoff team.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$::engineer</td>
<td></td>
</tr>
</tbody>
</table>
auto-assign (legacy)

DESCRIPTION

Caution

Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Makes the specified user or resource pool the responsible party for the task to which the handler is added. Optionally, you can make the same specified user/resource pool the responsible party for all subtasks of the parent task.

SYNTAX

auto-assign [subtasks] {-user=user-id | -person=person-name | -owner | -resourcepool=resourcepool}

ARGUMENTS

subtasks
(Optional) Propagates task assignments to subtasks of the current task (non-recursively).

-user
Makes the user whose ID is specified the responsible party for the task to which this handler is added.

Accepts a single valid Teamcenter user ID or one of these keywords: $USER and $TARGETOWNER.

-person
Makes the user whose name is specified the responsible party for the task to which this handler is added.

Accepts a single valid Teamcenter person name.

    Note
If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to use wayne, joan:

    -person=wayne\, joan

-owner
Makes the initiator (owner) of the workflow process the responsible party for the task to which this handler is added.

-resourcepool
Makes members of the specified resource pool the responsible party for the task to which this handler is added. You can define resource pools in the form of group::role, role, or group::.

Accepts a valid Teamcenter resource pool name or one of these keywords: $GROUP, $ROLE, $TARGETGROUP, and $ROLEINGROUP.

PLACEMENT

Place on the Start action.
Appendix A  Workflow handlers

RESTRICTIONS
None.

EXAMPLES

- This example makes Smith the responsible party for the task to which this handler is assigned, and all of the task's subtasks.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtasks</td>
<td></td>
</tr>
<tr>
<td>-user</td>
<td>Smith</td>
</tr>
</tbody>
</table>

- This example makes the process initiator the responsible party for the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-owner</td>
<td></td>
</tr>
</tbody>
</table>

- This example makes any engineer roles belonging to the manufacturing group resource pool the responsible party for the task to which this handler is assigned.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-resourcepool</td>
<td>manufacturing::engineer</td>
</tr>
</tbody>
</table>
auto-assign-rest (legacy)

DESCRIPTION

Caution

Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Automatically makes the specified user or resource pool the responsible party for any unassigned subtasks of the parent task to which this handler is added. You specify the user or resource pool by entering a comma-delimited list in the Arguments column for this handler.

This handler first assumes that the list contains user IDs, attempting to match the entries (in the order listed) to valid user IDs. The first entry matching a user ID is made the responsible party for any subtasks of the task to which this handler is assigned.

If no entries in the list match a valid user ID, the system attempts to match the entries (in the order listed) to valid resource pool names. The first entry matching a resource pool name (group, group/role or role) is made the responsible party for any subtasks of the task to which this handler is assigned.

If this handler is attached to the root task, with no argument specified, the process initiator is made the responsible party for all tasks in the process.

If this handler is attached to the root task, and one or more entries are contained in the list, the first valid user or resource pool is made the responsible party for all tasks in the process.

SYNTAX

auto-assign-rest list-of-users

ARGUMENTS

list-of-users

List of users and resource pools to which all remaining subtasks are assigned.

Accepts valid Teamcenter user IDs, resource pool names, and these keywords: $GROUP, $ROLE, $TARGETOWNER, $TARGETGROUP, and $ROLEINGROUP.

To assign a resource pool, use the following form: group:role, role, or group:. For example, to assign the Quality Engineer role in the Quality group as the resource pool, type Quality::Quality Engineer. Spaces are acceptable in the resource pool name.

PLACEMENT

Place on the Start action. Typically placed on the root task after the CR-assign-team-selector (legacy) handler.

RESTRICTIONS

None.

EXAMPLES

- In this example, there is a five-task workflow process containing the following tasks:
  - Do task, Review task, Checklist task, Review task, and Do task
Workflow handlers

The process is initiated by user Jones. The auto-assign-rest handler is placed on the root task, and the auto-assign handler is placed on the fourth task, set with the -owner argument.

Because the auto-assign-rest handler is placed on the root task and Smith is the first entry in the list, Smith is the responsible party for the first three tasks (and their subtasks). Because the auto-assign/-owner handler/argument is placed on the fourth task, Jones is the responsible party for the fourth task and its subtasks. Smith is the owner of the fifth task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith, Jones, Rogers, Patterson</td>
<td></td>
</tr>
</tbody>
</table>
CR-assign-team-selector (legacy)

DESCRIPTION

Caution
Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Assigns all select-signoff-team tasks in the entire process to the specified user, person, initiator (owner), or resource pool of the process. Only one argument can be defined; all arguments are mutually exclusive of each other.

SYNTAX

CR-assign-team-selector -user=user-id | -person=person-name | -owner | -resourcepool=resourcepool

ARGUMENTS

-user
Makes the user whose ID is specified the responsible party for all select-signoff-team tasks in the workflow process.
Accepts a single valid Teamcenter user ID. This is mutually exclusive of all other arguments.

-person
Makes the user whose person name is specified the responsible party for all select-signoff-team tasks in the workflow process.
Accepts a single valid Teamcenter person name. This is mutually exclusive of all other arguments.

-Note
If the person’s name includes a comma, you must include an escape character (\) to add the correct person. For example, to usewayne, joan:

-person=wayne \, joan

-owner
Makes the workflow process owner the responsible party for all select-signoff-team tasks in the workflow process.
Accepts no value. This is mutually exclusive of all other arguments.

-resourcepool
Makes all members of the specified resource pool the responsible party for all select-signoff-team tasks in the workflow process.
Accepts a single valid Teamcenter resource pool name. This is mutually exclusive of all other arguments. You can define resource pools in the form of group::role, role, or group::.

PLACEMENT

Place on the Start action of the root task.

RESTRICTIONS

None.
Appendix A  Workflow handlers

EXAMPLES

- This example assigns the user **jim** all **select-signoff-team** tasks in that process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-user</td>
<td>jim</td>
</tr>
</tbody>
</table>

- This example assigns the person **Jim Smith** all **select-signoff-team** tasks in that process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-person</td>
<td>Jim Smith</td>
</tr>
</tbody>
</table>

- This example assigns the owner of the process all **select-signoff-team** tasks in that process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-owner</td>
<td></td>
</tr>
</tbody>
</table>
**CR-fill-in-reviewers (legacy)**

**DESCRIPTION**

**Caution**

Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Automatically assigns signoff reviewers that meet specified user, group, or role criteria for the specified Review task. This criteria populates the signoff profiles.

This handler compares the assigned user with the profile definition in the corresponding select-signoff-team task. If the assigned user does not match the profile definition, automatic assignment does not occur and the select-signoff-team task must be performed manually.

**SYNTAX**

```
CR-fill-in-reviewers -reviewer={User: {user-id | $USER | $PROCESSOWNER | TARGETOWNER} | Group: {group | $GROUP | PROCESSGROUP | TARGETGROUP} | Role: {role | $ROLE} | ResourcePool: {resourcepool | $GROUP:: | ::$ROLE | $GROUP::$ROLE | $REVIEWERS | $PROCESSGROUP:: | $TARGETGROUP:: | $ROLEINGROUP} } [-level=level-name]
```

**ARGUMENTS**

- **-reviewer**
  Assigns the defined users, role members, group members, and/or resource pool members to the signoff team.

  To specify the type of reviewer required, use these options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong>: user-id</td>
<td>user-id is a specific user ID or keyword.</td>
</tr>
<tr>
<td><strong>Group</strong>: group-id</td>
<td>group-id is a specific group ID or keyword.</td>
</tr>
<tr>
<td><strong>Role</strong>: role</td>
<td>role is a specific role name or keyword.</td>
</tr>
<tr>
<td><strong>ResourcePool</strong>: resourcepool</td>
<td>resourcepool is a specific resource pool or keyword combination.</td>
</tr>
</tbody>
</table>

Accepts valid Teamcenter user IDs, roles, groups and resource pools. You can define resource pools in the form of group::role, role, or group::.

Also accepts these **keywords**:

- **$USER**
- **$GROUP**
- **$ROLE**
- **$PROCESSOWNER**
- **$PROCESSGROUP**
- **$TARGETGROUP**
- **$ROLEINGROUP**
- **$TARGETOWNER**

If **$USER** is used, and the current user belongs to several groups and roles, the behavior of the **$USER** keyword depends on the value of the **SIGNOFF_fill_in_reviewers** site preference, as follows:
Appendix A  Workflow handlers

-  1
   Attempts to match the current user’s group/role values with the profile first, default values second, then any other groups/roles of which the current user is a member. This is the default setting.

-  2
   Attempts to match the current user’s group/role values first, default values of which the current user is a member second.

-  3
   Attempts to match the current user’s group/role values.

-level
Release level-name to which the reviewers are added.

PLACEMENT
Place on the Start action of the relevant select-signoff-team task.

RESTRICTIONS
Use only with the select-signoff-team task; do not place on other tasks.

EXAMPLES
- This example designates user tom and all the members of the engineering group as reviewers for the Review task called level_1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-reviewer</td>
<td>User:tom,Group:engineering</td>
</tr>
<tr>
<td>-level</td>
<td>$ROOTTask.level_1</td>
</tr>
</tbody>
</table>

- This example shows the initiator of the process being added as a reviewer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-reviewer</td>
<td>User:$USER</td>
</tr>
<tr>
<td>-level</td>
<td>level_1</td>
</tr>
</tbody>
</table>

- These examples show how the SIGNOFF_fill_in_reviewers site preference, the user’s current group/role, the default group/role, and the profile defined in the select-signoff-team task affect the automatic assignment:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-reviewer</td>
<td>User:$USER</td>
</tr>
<tr>
<td>-level</td>
<td>level_1</td>
</tr>
</tbody>
</table>

Available groups and roles for the user TESTER:

<table>
<thead>
<tr>
<th>Group</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>Designer</td>
</tr>
<tr>
<td></td>
<td>Engineer</td>
</tr>
<tr>
<td></td>
<td>Checker</td>
</tr>
<tr>
<td>Group</td>
<td>Role</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>Body</td>
<td>Designer</td>
</tr>
<tr>
<td></td>
<td>Supervisor</td>
</tr>
<tr>
<td>Quality</td>
<td>Checker</td>
</tr>
<tr>
<td></td>
<td>Engineer</td>
</tr>
</tbody>
</table>

Scenario A
- Current group/role: **Quality/Checker**
- Default group: **Body**
- Valid profile: */Engineer*

The job is automatically assigned under settings 1, 2, and 3 because there is an **Engineer** role under the user's current group.

Scenario B
- Current group/role: **Body/Designer**
- Default group: **Quality**
- Valid profile: */Engineer*

The job is automatically assigned under settings 1 and 2 to **Quality/Engineer**. It is not automatically assigned in 3.

Scenario C
- Current group/role: **Development/Designer**
- Default group: **Body**
- Valid profile: */Engineer*

The job is automatically assigned under settings 1, 2, and 3 because there is an **Engineer** role under the user's current group.

Scenario D
- Current group/role: **Body/Designer**
- Default group: **Body**
- Valid profile: */Engineer*

The job is automatically assigned under setting 1. It is assigned to either **Development/Engineer** or **Quality/Engineer**.

The job is not automatically assigned under settings 2 and 3 because there is no matching in either current or default group.

Scenario E
- Current group/role: **Body/Designer**
- Default group: **Quality**
Valid profile: */Engineer

The job is automatically assigned under settings 1 and 2 to Quality/Engineer. It is not automatically assigned in 3.
CR-notify (legacy)

DESCRIPTION

Caution

Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Sends a report through the OS mail to all task reviewers. **CR-notify** cannot notify users through Teamcenter e-mail.

The **-report** argument differentiates **CR-notify** handler from the **notify** handler. In notification e-mail, the **-report** argument appends a report describing the signoff data associated with the **perform-signoff** task. **CR-notify** is designated for use on the **perform-signoff** task. The **notify** handler is used on any type of task.

SYNTAX

```
CR-notify -report={review | rejection | progress | level}
-recipient= {OS:user-name | User:username | Group:groupname | Role:rolename
            | AliasList:value | DistributionList:value | $USER | $OWNER
            | $GROUP | $ROLE
            | $ROLEINGROUP | $REVIEWERS | $RESPONSIBLE_PARTY
            | $UNDECIDED | $RESOURCE_POOL_ALL | $RESOURCE_POOL_NONE
            | $RESOURCE_POOL_SUBSCRIBED}
[-subject=string]
[-comments=string]
[-url = [rich | dhtml | html]}
```

ARGUMENTS

- **-report**
  Indicates the report type sent to recipients. Accepts one of the following values:
  
  - **review**
    Review required. Notifies all recipients when they must review target objects. The report lists target and reference object IDs and types.
  
  - **rejection**
    Rejection report. Notifies recipients that a process’ **Review** task has been rejected. The report lists target and reference object IDs, as well as the **Review** task reviewers, decisions, dates, and comments for each **Review** task. Do not use this value unless you want the process to always send a rejection notice.
  
  - **progress**
    Progress report. Notifies recipients of a process’ current state. The report lists the target and reference object IDs, as well as the **Review** task reviewers, decisions, dates, and comments for each **Review** task.
  
  - **level**
    Review task complete. Notifies recipients when a process’ **Review** task is completed. The report lists the target and reference object IDs, as well as the current **Review** task reviewers, decisions, dates, and comments.
Appendix A  Workflow handlers

-recipient
Specifies the task reviewers receiving notification. Accepts one of the following values:

• OS
  Sends a notification to the specified OS user name.
  \textit{user-name} is a single valid user name.

• User
  Sends notification to the specified user.
  \textit{user} is a single valid Teamcenter user ID.

• Group
  Sends a notification to the members of the specified group.
  \textit{group} is a single valid Teamcenter group.

• Role
  Sends a notification to the users who have the specified role.
  \textit{role} is a single valid Teamcenter group.

• AliasList/DistributionList
  Sends a notification to the users who are in the specified distribution list. Using a distribution list, any party outside the user’s network can be notified, using their e-mail address.
  \textit{value} is a single valid Teamcenter distribution list.

  \textbf{Note}
  
  \textit{AliasList} and \textit{DistributionList} are called an \textbf{Address List} in the user interface.

• $USER
  Sends e-mail to the current user.

• $OWNER
  Sends e-mail to the task owner.

• $GROUP
  Sends e-mail to all users in the current user’s group.

• $ROLE
  Sends e-mail to all users with the same role as the current user’s.

• $ROLEINGROUP
  Sends e-mail to all users with the same group and role as the current user’s.

• $REVIEWERS
Builds a list of all users who are reviewers in the same task level as the current reviewer and sends e-mail to all of them.

- **$RESPONSIBLE_PARTY**
  Sends e-mail to the designated responsible party for the task.

- **$UNDECIDED**
  Sends e-mail to the users who have not set the decision for the task.

- **$RESOURCE_POOL_ALL**
  Identifies all members of the resource pool.
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.

  When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, e-mail is sent to all the members of that resource pool.

  When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer, and no signoff decision has been made for this resource pool assignment, all members of that resource pool are notified.

  When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as responsible party, the e-mail is sent to all members of that resource pool.

- **$RESOURCE_POOL_NONE**
  Identifies all members of the resource pool.
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.

  When this argument is used along with $REVIEWERS or $UNDECIDED, and if a resource pool is assigned as a reviewer, e-mail is not sent to members or subscribers of the resource pool.

  When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as responsible party, the e-mail is not sent to members or subscribers of resource pool.

- **$RESOURCE_POOL_SUBSCRIBED**
  Identifies the users who have subscribed to resource pool.
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.

  When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, the e-mail is sent to users who have subscribed to the resource pool.

  When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer and no signoff decision has been made for this resource pool assignment, e-mail is sent to users who have subscribed to the resource pool.

  When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as a responsible party, the e-mail is sent to users who have subscribed to the resource pool.
Appendix A  Workflow handlers

Note
If the $RESOURCE_POOL_XXXX argument is not defined and the $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY arguments are used for a case where assignments are made to resource pools, the e-mail is sent using the EPM_resource_pool_recipients preference.

EPM_resource_pool_recipients can take one of the following values:
- all
  Sends mail to all members of resource pool.
- none
  Does not send a mail to members or subscribers of resource pool.
- subscribed
  Sends mail to Teamcenter users who have subscribed to resource pool.

If the $RESOURCE_POOL_XXXX argument is defined, the argument takes precedence over preference value.

If this argument is not defined and the EPM_resource_pool_recipients preference is not set, subscribed is considered the default value.

-subject
Displays the task name enclosed in brackets, followed by the string identified by this argument, on the OS mail's subject line.

-comments
Embeds user-defined comments in the body of the e-mail.

-url
Inserts a DHTML link to the workflow process into the notification e-mail, based on the value for -url. If no value is specified for -url, both links are added into the notification e-mail.

If the -url argument is not defined, the notification e-mail contains links depending on the values set in the EPM_notify_url_format preference.

EPM_notify_url_format can take the following values:
- rich
  Inserts a rich client link to the workflow process into the notification e-mail.
- dhtml
  Inserts a thin client (DHTML) link to the workflow process into the notification e-mail.
- html
  Inserts a thin client (HTML) link to the workflow process into the notification e-mail.
If the `-url` argument is not defined and the `EPM_notify_url_format` preference is not set in the preference file, rich client and thin client links are added to the notification e-mail as a default. The URL is generated only when the `WEB_default_site_server` preference is set to the thin client server node name.

**Note**

Rich client URL functionality must be enabled for links to rich client workflow processes to launch the rich client.

**Placement**

Placement depends on the value of the `-report` argument.

- **Review**
  
  The recommended placement is on the Complete action of the select-signoff-team task.

- **Rejection**
  
  The recommended placement is on the Perform or Undo action of the perform-signoff task.

  When placed on a Perform action, an e-mail is sent on a Reject action.

- **Progress**
  
  The recommended placement is on the Start or Complete action of the perform-signoff task.

- **Level**
  
  The recommended placement is on the Complete action of the perform-signoff task.

**Restrictions**

Use only on the perform-signoff task.

**Examples**

- This example designates user smith, member of the group manufacturing, OS users peters and john, users with role manager, and members of the address list VendorList as recipients of a progress report with the subject Manufacturing Release Process Completed.

  The CR-notify handler should be placed on Complete action of perform-signoff task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>progress</td>
</tr>
<tr>
<td>-subject</td>
<td>Manufacturing Release Process Completed</td>
</tr>
</tbody>
</table>

- This example designates the task owner as the recipient of a progress report with the subject Manufacturing Release Process Completed.
Appendix A  Workflow handlers

The **CR-notify** handler should be placed on **Complete** action of **perform-signoff** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>progress</td>
</tr>
<tr>
<td>-subject</td>
<td>Manufacturing Release Process Completed</td>
</tr>
<tr>
<td>-recipient</td>
<td>$OWNER</td>
</tr>
</tbody>
</table>

- This example builds a list of all users assigned as reviewers for the **perform-signoff** task.

The **CR-notify** handler should be placed on **Start** action of **perform-signoff** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>progress</td>
</tr>
<tr>
<td>-subject</td>
<td>Manufacturing Release Process Completed</td>
</tr>
<tr>
<td>-recipient</td>
<td>$REVIEWERS</td>
</tr>
</tbody>
</table>

- This example designates the task owner and task reviewers as recipients of a review report with the subject **TASK REVIEW NOTIFICATION**.

If any resource pool is assigned as a reviewer, then all users who have subscribed to that resource pool receive notification e-mail.

Place the **CR-notify** handler on the **Start** action of the **perform-signoff** task.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>review</td>
</tr>
<tr>
<td>-subject</td>
<td>TASK REVIEW NOTIFICATION</td>
</tr>
<tr>
<td>-comments</td>
<td>Please review the task.</td>
</tr>
<tr>
<td>-recipient</td>
<td>$OWNER, $REVIEWERS, $RESOURCE_POOL_SUBSCRIBED</td>
</tr>
</tbody>
</table>

- This example illustrates creating a process template with a **Review** task. Add the **CR-notify** handler in the **Undo** action of the **perform-signoff** task. Place a **demote-on-reject** handler on the **Perform** action of the **perform-signoff** task.

The notification is sent to task owner, responsible party, and reviewers. If any resource pool is assigned as a responsible party and/or as a reviewer, then notification is sent to all group members of that resource pool.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-report</td>
<td>reject</td>
</tr>
<tr>
<td>-subject</td>
<td>TASK REJECTION &amp; DEMOTE NOTIFICATION</td>
</tr>
<tr>
<td>-recipient</td>
<td>$RESOURCE_POOL_ALL, $OWNER, $RESPONSIBLE_P, $REVIEWERS</td>
</tr>
</tbody>
</table>
notify (legacy)

DESCRIPTION

Caution

Do not use this syntax when you add new handlers. The syntax and arguments are valid only for handlers created in systems prior to Teamcenter 8.

Informs users of a task’s status through e-mail.

The CR-notify (legacy) -report argument differentiates the CR-notify (legacy) handler from the notify (legacy) handler. In notification e-mail, the -report argument appends a report describing the signoff data associated with the perform-signoff task. Therefore, you should use the CR-notify (legacy) handler on the perform-signoff task, whereas the notify (legacy) handler is more generic and can be used on any type of task.

If the -attachment argument is defined, recipients also receive Teamcenter e-mail, and the OS e-mail includes a notification that the user has received program e-mail.

SYNTAX

notify

-recipient= [OS:user-name | User:user | Group:group | Role:role
| DistributionList:value | $USER | $OWNER | $GROUP | $ROLE
| $ROLEINGROUP | $REVIEWERS | $RESPONSIBLE_PARTY
| $UNDECIDED | $RESOURCE_POOL_ALL | $RESOURCE_POOL_NONE
| $RESOURCE_POOL_SUBSCRIBED}
[-subject=string]
[-comments=string]
[-url=[rich|dhtml|html]]
[-attachment=$TARGET | $PROCESS | $REFERENCE]

ARGUMENTS

-recipient

Specifies the task reviewers receiving notification. Accepts one of the following values:

- OS
  Sends a notification to the specified OS user name.
  
  user-name is a single valid user name.

- User
  Sends notification to the specified user.
  
  user is a single valid Teamcenter user ID.

- Group
  Sends a notification to the members of the specified group.
  
  group is a single valid Teamcenter group.

- Role
  Sends a notification to the users who have the specified role.
**Workflow handlers**

*role* is a single valid Teamcenter group.

- **DistributionList**
  Sends a notification to the users who are in the specified distribution list. Using a distribution list, any party outside the user's network can be notified, using their e-mail address.
  
  *value* is a single valid Teamcenter distribution list.

  **Note**
  DistributionList is called an **Address List** in the user interface

- **$USER**
  Sends e-mail to the current user.

- **$OWNER**
  Sends e-mail to the task owner.

- **$GROUP**
  Sends e-mail to all users in the current user's group.

- **$ROLE**
  Sends e-mail to all users with the same role as the current user's.

- **$ROLEINGROUP**
  Sends e-mail to all users with the same group and role as the current user's.

- **$REVIEWERS**
  Builds a list of all users who are reviewers in the same task level as the current reviewer and sends e-mail to all of them.

- **$RESPONSIBLE_PARTY**
  Sends e-mail to the designated responsible party for the task.

- **$UNDECIDED**
  Sends e-mail to the users who have not set the decision for the task.

- **$RESOURCE_POOL_ALL**
  Identifies all members of the resource pool.

  This argument has an effect only when it is used along with **$REVIEWERS**, **$UNDECIDED**, or **$RESPONSIBLE_PARTY**.

  When this argument is used along with **$REVIEWERS**, and if a resource pool is assigned as a reviewer, e-mail is sent to all the members of that resource pool.

  When this argument is used along with **$UNDECIDED**, and if a resource pool is assigned as a reviewer, and no signoff decision has been made for this resource pool assignment, all members of that resource pool are notified.
When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as responsible party, the e-mail is sent to all members of that resource pool.

• $RESOURCE_POOL_NONE
  
  Identifies all members of the resource pool.
  
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.
  
  When this argument is used along with $REVIEWERS or $UNDECIDED, and if a resource pool is assigned as a reviewer, e-mail is not sent to members or subscribers of the resource pool.
  
  When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as responsible party, the e-mail is not sent to members or subscribers of resource pool.

• $RESOURCE_POOL_SUBSCRIBED
  
  Identifies the users who have subscribed to resource pool.
  
  This argument has an effect only when it is used along with $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY.
  
  When this argument is used along with $REVIEWERS, and if a resource pool is assigned as a reviewer, the e-mail is sent to users who have subscribed to the resource pool.
  
  When this argument is used along with $UNDECIDED, and if a resource pool is assigned as a reviewer and no signoff decision has been made for this resource pool assignment, e-mail is sent to users who have subscribed to the resource pool.
  
  When this argument is used along with $RESPONSIBLE_PARTY, and if a resource pool is assigned as a responsible party, the e-mail is sent to users who have subscribed to the resource pool.

  **Note**

  If the $RESOURCE_POOL_XXXX argument is not defined and the $REVIEWERS, $UNDECIDED, or $RESPONSIBLE_PARTY arguments are used for a case where assignments are made to resource pools, the e-mail is sent using the EPM_resource_pool_recipients preference.

  EPM_resource_pool_recipients can take one of the following values:
  
  o **all**
    
    Sends mail to all members of resource pool.
  
  o **none**
    
    Does not send a mail to members or subscribers of resource pool.
  
  o **subscribed**
    
    Sends mail to Teamcenter users who have subscribed to resource pool.
If the $RESOURCE_POOL_XXXXX argument is defined, the argument takes precedence over preference value.

If this argument is not defined and the EPM_resource_pool_recipients preference is not set, subscribed is considered the default value.

-subject
Displays the task name enclosed in brackets, followed by the string identified by this argument, on the OS mail's subject line.

-comments
Embeds user-defined comments in the body of the e-mail.

-url
Inserts a DHTML link to the workflow process into the notification e-mail, based on the value for -url. If no value is specified for -url, both links are added into the notification e-mail.

If the -url argument is not defined, the notification e-mail contains links depending on the values set in the EPM_notify_url_format preference.

**EPM_notify_url_format** can take the following values:

- **rich**
  Inserts a rich client link to the workflow process into the notification e-mail.

- **dhtml**
  Inserts a thin client (DHTML) link to the workflow process into the notification e-mail.

- **html**
  Inserts a thin client (HTML) link to the workflow process into the notification e-mail.

If the -url argument is not defined and the EPM_notify_url_format preference is not set in the preference file, rich client and thin client links are added to the notification e-mail as a default. The URL is generated only when the WEB_default_site_server preference is set to the thin client server node name.

**Note**
Rich client URL functionality must be enabled for links to rich client workflow processes to launch the rich client.

-attachment
Adds an attachment to a Teamcenter mail. This argument does not have any affect on operating system e-mail. The attachment value can be any of the following:

- **$TARGET**
  The workflow target attachments are included in the mail.

- **$PROCESS**
  The workflow process is included in the mail.
• **$REFERENCE**
  The task attachments reference list is included in the mail.

**PLACEMENT**
When `$REVIEWERS` or `$UNDECIDED` is used as the key word, place on the Start or Complete action of the `perform-signoff` task.

**REQUIREMENTS**
None.

**Note**
To specify multiple attachments, you must call the `-attachment` argument multiple times. For multiple recipients using the `-recipient` argument, you must call the argument multiple times.

**EXAMPLES**

• This example sends an e-mail with the subject **Lower Right Sub-Assembly Review** to all users on the **design** and **qualityControl** address lists. The comment described in the example appears in the body of the e-mail text. In addition to the e-mail, the recipients also receive a Teamcenter mail that contains both the workflow target attachments and the current process.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-subject</td>
<td>Lower Right Sub-Assembly Review</td>
</tr>
<tr>
<td>-recipient</td>
<td>DistributionList:design, DistributionList:qualityControl</td>
</tr>
<tr>
<td>-comment</td>
<td>Please review new sub-assembly and report any concerns directly to the Product Manager.</td>
</tr>
<tr>
<td>-attachment</td>
<td>$TARGET, $PROCESS</td>
</tr>
</tbody>
</table>

• This example sends an e-mail to the designated responsible party for the task. If the responsible party is a resource pool, no e-mail is sent.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-recipient</td>
<td><code>$RESPONSIBLE_PARTY</code>, <code>$RESOURCE_POOL_NONE</code></td>
</tr>
</tbody>
</table>

• This example designates OS users **peters** and **john**, user **Smith**, members of the group **manufacturing**, and members of the address list **purchasing** as recipients of an e-mail with the subject **Manufacturing Release Procedure Completed**.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-subject</td>
<td>Manufacturing Release Procedure Completed</td>
</tr>
</tbody>
</table>
Rule handlers

Rule handlers integrate workflow business rules into EPM workflow processes at the task level. They attach conditions to an action.

Many conditions defined by a rule handler are binary (that is, they are either true or false). However, some conditions are neither true nor false. EPM allows two or more rule handlers to be combined using logical AND/OR conditions. When several rule handlers are combined using a logical Or condition, rule handler quorums specify the number of rule handlers that must return go for the action to complete.

This section provides information about each rule handler.
assert-signoffs-target-read-access

DESCRIPTION
Checks if all the selected reviewers have read access to the attached target attachments.

SYNTAX
assert-signoffs-target-read-access

ARGUMENTS
None.

PLACEMENT
Place on the Complete action of a select-signoff-team task.

RESTRICTIONS
None.
check-condition

DESCRIPTION

By default, this handler is placed on the Complete action of the Condition task, and on the successor tasks of the Validate task. When placed on these tasks, no arguments should be used. When placed on the Complete action of the Condition task, the handler confirms the result of the Condition task is either true or false or the specified custom result. The handler prevents the Condition task from completing until the default setting of unset has been modified to true or false. When placed on the successor tasks of the Validate task, the handler confirms whether errors occurred (either any error, or the specified errors.)

This handler can also be placed on the Start action of all tasks immediately succeeding the Condition task. Use the task-name argument to specify the name of the preceding Condition task and the value (true, false, or specified custom result) that must be met. (This value is defined during the workflow process template design, when the two or more flow paths that branch from the Condition task are created.) The handler returns EPM_go when the value matches or EPM_nogo when the value does not match. The immediately succeeding tasks only start if they match the required value, resulting in the conditional branching of the workflow process flow.

This handler exists as part of the workflow conditional branching functionality. Manually adding this handler to a task other than a Condition task, a task succeeding a Condition task, or the successor task of a Validate task has no advantage and is not recommended.

SYNTAX

check-condition task-name={true | false | custom-result | ANY | error-code}

ARGUMENTS

True | False
This argument is required if you place the handler on the Start action of a task succeeding a Condition task.

You must omit this argument if you place the handler on the Complete action of a Condition task.

custom-result
Use this argument in conjunction with a Condition task, placing this handler on a successor task.

Valid values are any string. When the Condition task’s task results return a value matching the value defined for this argument, the successor task starts when the Condition task completes. Multiple values are accepted, separated by commas.

Note

This argument is automatically set when you use the Set Custom Result option to configure the flow path from the Condition task to the successor task.

For more information about this configuration, see Set Condition task paths.
ANY
Use this argument in conjunction with a Validate task, placing this handler on a successor task. Indicates that if any error occurs on the Validate task, the workflow process starts the successor task.

Note
This argument is automatically set when you use the Set to Error Path option to configure a failure path from the Validate task to the successor task.

For more information about this configuration, see Creating failure paths.

text
Use this argument in conjunction with a Validate task, placing this handler on a successor task. Indicates that if the specified error codes occur on the Validate task, the workflow process starts the successor task.

Note
This argument is automatically set when you use the Set Error Codes option to configure a failure path from the Validate task to the successor task.

For more information about this configuration, see Creating failure paths.

PLACEMENT
Place on the Complete action of a Condition task, the Start action of any successor tasks of a Condition task, or the successor tasks of a Validate task.

RESTRICTIONS
None.

Note
The default placement of this handler on the Complete action of the Condition task was implemented in Teamcenter engineering process management 8.1. When working with a pre-Teamcenter engineering process management 8.1 installation that has been upgraded to Teamcenter engineering process management 8.1 or later, you must manually add this handler to the Complete action of the Condition task to replicate the behavior documented for this handler.

Note
Workflow Designer provides a number of prepackaged task templates, such as the Review task, Route task, and Acknowledge task templates. Adding subtasks below any of these tasks to implement a branching condition is not recommended as this may jeopardize the integrity of the task's structure, and doing so may result in unpredictable behavior.
check-process-completion

DESCRIPTION

Note

The Sync task and related check-process-completion handler are deprecated, and will be obsolete in a future release. Obtain equivalent functionality by creating subprocesses from parent processes.

For more information about this functionality, see What are subprocesses?

Checks for completion of the specified workflow processes in the reference attachment. If no argument is specified, all workflow processes in the reference attachment are checked for completion. If complete, the handler returns a EPM_go.

SYNTAX

check-process-completion [$TEMPLATE=template-name] [$PROCESS=process-name]

ARGUMENTS

$TEMPLATE
Any workflow process based on the selected template is checked for completion.

$PROCESS
The selected workflow process is checked for completion.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.
**check-responsible-party**

**DESCRIPTION**
Verifies that the current user is the responsible party for the task (every task has a default responsible party). If not, it verifies whether the current user meets the criteria specified in the argument of the handler.

**SYNTAX**
```
check-responsible-party [-responsible={User|Group|Role}:value]
```

**ARGUMENTS**

- **-responsible**
  (Optional) Defines an additional responsible party.

**PLACEMENT**
Place on the Perform action of the task.

**RESTRICTIONS**
This handler cannot be placed on the Perform action of the root task.

**EXAMPLES**
This example shows user **george**, members of group **dba**, and the responsible party being allowed to perform the action associated with this handler.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-responsible</td>
<td>User:george, Group:dba</td>
</tr>
</tbody>
</table>
check-signoff

DESCRIPTION
Checks decisions of all the signoffs attached to this task. If the number of approvals is greater than, or equal to, the quorum, then EPM_go is returned. If it is possible to obtain enough approvals from those signoffs without a decision, EPM_undecided is returned. Otherwise, there are too many rejections and the function EPM_nogo is returned.

SYNTAX
check-signoff -QUORUM=n

ARGUMENTS
-QUORUM
Specifies the approval quorum, where n is an integer specifying the quorum. A value of -1 sets the quorum equal to the total number of signoffs; in other words, a unanimous decision is required.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.
CR-assert-targets-checked-in

**DESCRIPTION**
Verifies that all target objects in this workflow process are checked in.

**SYNTAX**
CR-assert-targets-checked-in

**ARGUMENTS**
None.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
None.
CR-check-item-status

DESCRIPTION
Verifies that all ImanRelations of a target item or item revision have been released or that these ImanRelations are also target objects in this workflow process. If the target object is an item, this handler checks the item’s Requirements folder; if the target object is an item revision, this handler checks the item revision’s Specification folder. All objects in these folders must satisfy these requirements for the handler to return EPM_go. The relation, type, and status arguments verify their relation, type, and status, respectively.

SYNTAX
CR-check-item-status [-relation=relation-name] [-type=object-type] [-status=status-name-to-check]

ARGUMENTS
- relation
Relation name.
- type
Object type.
- status
Status to check.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
• This example verifies the text datasets in the Requirements folder of a target object have the status of X:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-relation</td>
<td>IMAN_requirement</td>
</tr>
<tr>
<td>-type</td>
<td>Text</td>
</tr>
<tr>
<td>-status</td>
<td>X</td>
</tr>
</tbody>
</table>

• This example verifies all the UGPART datasets of a target object have been assigned status. For example, that the datasets are released, or are the target object of the present job:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>UGPART</td>
</tr>
</tbody>
</table>
debug-rule

DESCRIPTION
Notifies a user that an action is executing. Attaching debug-rule to any EPM action notifies the user when that task action executes by printing that action name to the standard output device.

SYNTAX
debug-rule string

ARGUMENTS
string
Additional descriptive string appended to the action name.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
This example notifies the user when the Complete action executes by printing Complete, action is executing to the standard output device.

debug-rule action is executing

Note
This example assumes you have attached this handler to a Complete action.
disallow-adding-targets

DESCRIPTION

Disallows adding targets interactively after a workflow process is initiated. A switch can be used to specify the types of objects to be excluded. If you configure other handlers to add targets programmatically, they are added during the workflow process even if this handler is used.

Note

The EPM-attach-related-objects and EPM-attach-assembly-components handlers are dependent on this handler.

For more information, see the documentation for those handlers.

SYNTAX

disallow-adding-targets [-exclude_type=type-of-object [, type-of-object2 ]]

ARGUMENTS

-exclude_type=type-of-object [, type-of-object2 ]
Types of objects that are allowed to be added as targets after the workflow process is initiated.

This argument is optional.

PLACEMENT

Place on the Perform action of the root task.

RESTRICTIONS

Use the EPM-set-job-protection handler to ensure that the required change access is asserted.

EXAMPLES

Note

It is good practice to add this handler to the root task Perform action to ensure that target objects are not added from a workflow process once it is started. If you want to allow the addition of objects of all types as targets, this handler should be removed from the respective workflow process template, and you must ensure that the desired users have change access to the workflow process (job) object. You may need to use the EPM-set-job-protection handler to ensure that the required change access is asserted.

This example allows only BOM view revisions to be added interactively as targets after the workflow process is initiated.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-exclude_type</td>
<td>PSBOMViewRevision</td>
</tr>
</tbody>
</table>
disallow-removing-targets

DESCRIPTION

Prevents targets from being removed from a workflow process after the workflow process has been started.

It is good practice to add this handler to the root task of the Perform action. This prevents target objects from being removed from a workflow process once it is started. To allow the removal of targets, verify that this handler has been removed from the respective workflow process template (if it has not been removed, do so) and ensure that the desired users have change access to the workflow process object. You may need to use the EPM-set-job-protection handler to ensure that the required change access is asserted.

Note

The named ACL must have change access to provide the proper protection.

SYNTAX

disallow-removing-targets

ARGUMENTS

None.

PLACEMENT

Place on the Perform action of the root task.

RESTRICTIONS

None.
**ECM-check-target-is-ec**

**DESCRIPTION**
Verifies that at least one engineering change (EC) revision has been added as a target object of the process. If not, this rule handler fails. The user must add an EC revision as a target to go forward with the process.

**Note**
Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

**SYNTAX**
ECM-check-target-is-ec

**ARGUMENTS**
None.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
Do not use this handler with non-EC processes.
ECM-is-all-affected-irs-released

**DESCRIPTION**

Verifies that all affected revisions of the target engineering change (EC) revision have been released. If an optional list of release statuses is given, it also checks that the release status of all affected revisions is in the input status list. Siemens PLM Software recommends that you use this handler in conjunction with the **ECM-start-new-sub-processes** action handler. You may use the combination of these two handlers several times in an EC process.

**Note**

Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

**SYNTAX**

```
ECM-is-all-affected-irs-released -status=status-name [-types=type-to-check] [-check_effectivity=effectivity-type] [-folder_name=folder-name]
```

**ARGUMENTS**

- **-status**
  Status name to check for affected revisions.

- **-types**
  If specified, system only checks for the item revisions of the given type.

- **-check_effectivity**
  If specified, check for the specified type effectivity, such as Date, Unit and End Item Effectivity or ANY.

- **-folder_name**
  If specified, system checks for the revisions in the specified folder. Otherwise checks for the affected revisions.

**PLACEMENT**

Requires no specific placement.

**RESTRICTIONS**

Do not use this handler with non-EC processes.

**EXAMPLES**

This example verifies all affected revisions of the targeted EC have already been released with a status set as **Released** or **Production Ready**. If any of the revisions are not released, or are released but not with one of these two statuses, this rule handler fails.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>Released, Production Ready</td>
</tr>
</tbody>
</table>

If no argument is passed, this handler verifies that all affected revisions have been released with any release status.
**ECM-is-all-affected-irs-target**

**DESCRIPTION**
Verifies that all affected revisions of the targeted engineering change (EC) revision have been added as target objects of the EC process. Siemens PLM Software recommends that this handler be used in conjunction with the **ECM-add-affected-irs-as-target** handler. The combination of these two handlers may be used several times in an EC process.

**Note**
Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

**SYNTAX**
ECM-is-all-affected-irs-target

**ARGUMENTS**
None.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
Do not use this handler with non-EC processes.
ECM-is-valid-ec-process

DESCRIPTION
Validates the workflow process template initiated on the change object.

While creating a change type from the Business Modeler IDE, you must specify the associated workflow process templates for that change type. For example, when creating the CN change type, CMII Change Notice is the associated workflow process template. Use this rule handler to restrict users from initiating any other workflow processes on the change type other than the ones specified during creation.

Note
Change Manager does not support the use of ECM workflow handlers. They can only be used by Change Viewer.

SYNTAX
ECM-is-valid-ec-process

ARGUMENTS
None.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
Do not use this handler with non-EC processes.
EPM-assert-target-classified

DESCRIPTION
Checks whether an item is classified by verifying that target objects of the specified types in this workflow process are classified. If the item is classified, the rule handler returns `EPM_go`. If the item is not classified, it returns `EPM_nogo`. The user then has the option of associating this rule handler with the selected workflow completion process, therefore, preventing the state transition if the item does not comply with the classified business rule.

SYNTAX

```
EPM-assert-target-classified [-allowed_type =type-of-workspace-object [, type-of-workspace-object2...]]
```

ARGUMENTS

- `allowed_type`
  Must be valid workspace object types. For example: `ItemRevision` and `ITEM`.

  If this argument is specified as `Dataset`, any type of dataset (UGMASTER, UGPART, Text, and so on) is considered.

  If this argument is specified as `ItemRevision`, any type of item revision (DocumentRevision, and so on, and any custom item revision types) is considered.

PLACEMENT

Place on any action and on any task.

REstrictions

None.

EXAMPLES

This example checks item revisions as targets:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-allowed_type</td>
<td>ItemRevision</td>
</tr>
</tbody>
</table>

This handler is very useful in restricting unclassified items and item revisions from being released.
EPM-check-action-performer-role

DESCRIPTION
Checks whether the user performing this action matches the criteria specified in the handler arguments.

SYNTAX
EPM-check-action-performer-role [owner | $OWNER] | [group | $GROUP] | [$RESPONSIBLE_PARTY] | [privileged | $PRIVILEGED] | [group::* | role] | [role]

ARGUMENTS
- **owner | $OWNER**
  Specifies the owner of the task.
- **group | $GROUP**
  Specifies the user be in the same group as the responsible party of the task.
- **$RESPONSIBLE_PARTY**
  Specifies the responsible party of the task.
- **privileged | $PRIVILEGED**
  Specifies the responsible party of the task and the owner of the workflow process. If the task does not have a responsible party, the handler ascends the hierarchy of tasks to find the first assigned responsible party.
  - **group::* | role**
    Specifies a group name and role name to match.
  - **role**
    Specifies a role name to match.

PLACEMENT
Requires no specific placement. Typically place on the **Assign**, **Skip**, or **Undo** actions to control access to those actions.

RESTRICTIONS
There must be no role in the database with the name **privileged**.

EXAMPLES
- This example allows the owner of the workflow process and the responsible party to trigger the action.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>privileged</td>
<td></td>
</tr>
</tbody>
</table>

- This example allows any member of the **engineering** group to trigger the action.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>engineering::*</td>
<td></td>
</tr>
</tbody>
</table>

- This example allows any user with the role of **manager** to trigger the action.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>manager</td>
<td></td>
</tr>
</tbody>
</table>
EPM-check-assembly-status-progression

DESCRIPTION
Enforces status value progression for BOM assemblies. When an assembly is selected for release to a specific status, this handler checks if all its components are at or above the status of the assembly.

An item revision is required as the target of the workflow process. Additional targets are derived by traversing the BOM attached to the target item revision. The handler then compares the targeted release status to the release status of its components. The latest release status of the components must be the same or later in the status progression, in relationship to the targeted release status of the assembly.

This handler traverses only one level. If every subassembly of the target were previously released by this handler, all subassemblies would have been forced to align to the progression path.

**Note**
If the target release status of the assembly must be checked against the latest release status of its own preceding revisions, use the EPM-check-status-progression handler before using this handler.

If the workflow process contains several Condition tasks that apply different release statuses at different levels, the value provided in the -status argument can be used. If this argument is not used in this situation, the status applied to the target object is applied to the object. There is no validation ensuring the value provided by this argument is a valid status being applied by the current release procedure.

You can check the BOM components for a specific status, rather than for any status. In this case, the handler traverses the BOM, checking for the specific release status of each individual component, rather than any status; the progression path is not read.

SYNTAX

ARGUMENTS
- **-rev_rule**
  Specifies the name of the revision rule to be applied for BOM traversal. If not supplied, the default revision rule is used.

- **-saved_var_rule**
  Specifies the name of the saved variant rule to be applied on BOM window for BOM traversal.

- **-status**
  Defines the status being applied to the target object.

- **-check_component_status**
  Checks if all the components have this status.
Appendix A  Workflow handlers

-check_unconfigured
Returns NO-GO in case the applied revision rule on the assembly results in unconfigured children.

PLACEMENT
Place on any task action. However, if the target assembly is very large, placing it on the Start action of the root task could affect performance. With this placement, the Create Process dialog box does not close until the entire assembly is traversed.

RESTRICTIONS
If there are separate release progression tables for assemblies and for components, there must be common statuses between these two tables. If there are no common statuses between these two tables, this handler returns an EPM_nogo and aborts the release process of the assembly when the workflow process is initiated. See the fourth example below.

EXAMPLES

- In this example, assume that the revision rule is Working and the variant rule is GMC 300 Rule.
  If an assembly target object has to be checked against the status of its components, using a specific revision rule and saved variant rule to configure the assembly, define the arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rev_rule</td>
<td>Working</td>
</tr>
<tr>
<td>-saved_var_rule</td>
<td>GMC 300 Rule</td>
</tr>
</tbody>
</table>

- In this example, if the assembly target object being released has to check if each of its components are at Design status, rather than any status, define the following argument. In this case, the progression path is not read:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-check_component_status</td>
<td>Design</td>
</tr>
</tbody>
</table>

- In this example, assume a workflow process contains several Condition tasks, which apply different release statuses at different levels, and Design is a status at one of the levels. To check the status of Design against the progression path, rather than deriving the status being applied to the target object, define the following argument:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>Design</td>
</tr>
</tbody>
</table>

- In this example, consider the scenario:
  o Assy1/A is a CORP_Product item revision, at Design status
  o 002/A is a CORP_Part item revision, at Design status
  o 003/A is a CORP_Part item revision, at Design status
  o CORP_Product progression path: Assembly Quote, Experimental, Development, Design, Prototype, Manufacturing, Production
Workflow handlers

- **CORP_Part** progression path: Quote, Experimental, Development, Design, Manufacturing, Production

If Assy1/A is now being released to **Prototype** status, the handler returns an **EPM_nogo** because the component’s progression path (and therefore the component progression table) does not contain the **Prototype** status. The assembly process would be aborted.

**ADDITIONAL INFORMATION**

- If the target release status of the assembly has to be checked against the latest release status of its own preceding revisions, the best practice is to use the **EPM-check-status-progression** handler before this handler.

- The progression path must be manually defined in the **ProgressionPath.plmxml** file before the handler can reference the path. The file is stored in the **TC_DATA** directory. Create a backup copy of this file before editing it.

All target types that you want to follow the progression path must be set in this file. A **UserData** block must be created for each type that follows a progression path. For example, to define the progression path for the **ItemRevision**, **PSBOMView**, and **MSWord** types, the **UserData** blocks can be defined as follows:

```xml
<UserData id="id1">
    <UserValue title="Type" value="ItemRevision"/>
    <UserValue title="ReleaseProgressionList"
        value="Quote,Development,Prototype,Production">
        </UserValue>
    </UserData>
<UserData id="id2">
    <UserValue title="Type" value="PSBOMView"/>
    <UserValue title="ReleaseProgressionList"
        value="Quote1,Development1,Prototype1,Production1">
        </UserValue>
    </UserData>
<UserData id="id3">
    <UserValue title="Type" value="MSWord"/>
    <UserValue title="ReleaseProgressionList"
        value="Quote2,Development2,Prototype2,Production2">
        </UserValue>
    </UserData>
```

**Note**

- Add the **UserData** blocks between the **<PLMXML>** and **</PLMXML>** tags.

- Ensure you increment the **UserData id** value when you add a new entry.

- After adding a new **UserData** block, change the value for **Type** to a type you are defining.

- You can modify the value of the release status to meet your requirements.
Appendix A  Workflow handlers

EPM-check-object-properties

DESCRIPTION
Checks that a required or non-null value has been entered for the specified properties of the specified object type that is attached to the current workflow process. If any specified properties do not have the required values, an error displays, listing all the specified properties not containing the required values.

If the specified object type is a form, this handler also checks for form attributes. If the -check_first_object_only argument is specified, it only checks the property on the first attached target type. You can use this handler to ensure that you are not releasing the form without defining the mandatory attributes.

SYNTAX
EPM-check-object-properties -type=object-type
-props=property-names-separated-by-comma
[values=required-values-separated-by-commas]
[-att_type=attachment-type]
[check_first_object_only]

ARGUMENTS
- type
Defines the type of object to be checked. This object can be attached to the task either as a target or as a reference.

- props
Defines the properties to be checked. Enter a list separated by commas.

- values
Specifies the required values to be checked. Enter a list separated by commas. The order of these values must match the order of properties listed in the -props argument.

This argument is optional.

- att_type
Specifies the type of attachment to be checked.

• TARGET
  Checks the targets attachment.

• REFERENCE
  Checks the reference attachment.

• BOTH
  Checks both types of attachments.

If this argument is not used, the target attachment is checked.

This argument is optional.

- check_first_object_only
If specified, only the first object of type specified by type is considered. This argument is optional.

PLACEMENT
Place on any action except the Perform action.


RESTRICTIONS

None. Both empty and null values are treated as null values.

EXAMPLES

- This example checks the target **CMII CR Form** for nonempty values for **cr_priority** and **prop_soln** properties:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>CMII CR Form</td>
</tr>
<tr>
<td>-props</td>
<td>cr_priority, prop_soln</td>
</tr>
<tr>
<td>-att_type</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

- This example checks the target **CMII CR Form** for the specific value 1 = **High** for the **cr_priority** property, and the specific value **Corrective Action** for the **cr_type** property:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>CMII CR Form</td>
</tr>
<tr>
<td>-props</td>
<td>cr_priority, cr_type</td>
</tr>
<tr>
<td>-values</td>
<td>1 = High, Corrective Action</td>
</tr>
<tr>
<td>-att_type</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

- This example checks the target **CMII CR Form** for the specific value 1 = **High** for the **cr_priority** property, and the specific value **Corrective Action** for the **cr_type** property, and any nonempty value for the **prop_soln** property:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-props</td>
<td>cr_priority, prop_soln, cr_type</td>
</tr>
<tr>
<td>-values</td>
<td>1 = High, Corrective Action</td>
</tr>
<tr>
<td>-type</td>
<td>CMII CR Form</td>
</tr>
<tr>
<td>-att_type</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

**Note**

Not placing a value between two commas instructs the system to check for any non-null values for the corresponding property. In the previous example, the second of the three properties to be checked, the **prop_soln** property, corresponds to the empty value. Therefore, any non-null values for this property are checked.

- This example checks the target **CMII CR Form** for the specific value 1 = **High** for the **cr_priority** property, and the specific value **Corrective Action** for the **cr_type** property, and any nonempty value for the **prop_soln** property:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>CMII CR Form</td>
</tr>
<tr>
<td>-props</td>
<td>cr_priority, cr_type, prop_soln</td>
</tr>
</tbody>
</table>
### Appendix A  
**Workflow handlers**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-values</td>
<td>1 = High, Corrective Action</td>
</tr>
<tr>
<td>-att_type</td>
<td>TARGET</td>
</tr>
</tbody>
</table>

**Note**

An alternative method of checking for nonvalues as illustrated in example 3 is to place the property that needs to be checked for nonvalues at the end of the properties list, as in the previous example. This also instructs the system to check for any non-null values for the corresponding property.

- This example checks the target *and* reference **CMII CR Form** for the specific value `1 = High` for the **cr_priority** property, and the specific value **Corrective Action** for the **cr_type** property and any nonempty value for the **prop_soln** property:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>CMII CR Form, CMII CN Form</td>
</tr>
<tr>
<td>-props</td>
<td>cr_priority, prop_soln, cr_type</td>
</tr>
<tr>
<td>-values</td>
<td>1 = High,,Corrective Action</td>
</tr>
<tr>
<td>-att_type</td>
<td>BOTH</td>
</tr>
<tr>
<td>-check_first_object_only</td>
<td></td>
</tr>
</tbody>
</table>
EPM-check-occ-notes

DESCRIPTION
Checks whether a value has been entered for the specified occurrence note types on the occurrences of a given assembly.

SYNTAX
EPM-check-occ-notes -note_types=occurrence-note-type-names

ARGUMENTS

-note_types
Defines the occurrence note types to be validated.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
This example checks if the given assembly has the Torque and Power occurrence note types defined in all its BOM lines:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-note_types</td>
<td>Torque,Power</td>
</tr>
</tbody>
</table>
EPM-check-oramfg-transfer-status

DESCRIPTION
Checks the OverallTransferStatus field in the OraMfgTransferInfo form for the transfer status. If the transfer status is Success, it promotes the job to the next task in the workflow. If the transfer status is Failed, it demotes the job to the previous task in the workflow.

SYNTAX
EPM-check-oramfg-transfer-status

ARGUMENTS
None.

PLACEMENT
Attach it to the Complete action of a task.

RESTRICTIONS
None.
EPM-check-related-objects

DESCRIPTION
Checks whether the specified target object contains the required attachments, and whether the object is in process or has achieved a valid status. Use this handler to define the type of the target object. You can check only one type of target object per handler. You can check for either a primary or secondary attachment type; the validation confirms the attachment is the specified type and specified relation.

SYNTAX
EPM-check-related-objects [-target_type=type-of-target-object]
{-primary_type=type-of-target-object | -secondary_type=secondary-object-type}
-relation=relation-type [-status=status-names | ANY | NONE | IN_PROCESS]
[-check_first_object_only]

ARGUMENTS
-target_type
Specifies the type of the target object.

-primary_type
Specifies the type of the primary attachment.

This argument is mutually exclusive of the -secondary_type argument. You may specify only one of these arguments.

-secondary_type
Specifies the type of the secondary attachment. This argument is mutually exclusive of the -primary_type argument. You may specify only one of these arguments.

-relation
Specifies the relation to be checked. The relation is between the specified target object and the specified attachment (either the primary attachment or the secondary attachment).

- Specify verification of a manifestation relationship with IMAN_manifestation.
- Specify verification of a specification relationship with IMAN_specification.
- Specify verification of a requirement relationship with IMAN_requirement.
- Specify verification of a reference relationship with IMAN_reference.
- Specify verification of a BOM view attachment with PSBOMViewRevision.
- Specify verification of an affected item of an engineering change (EC) object with EC_affected_item_rel.
- Specify verification of a solution item of an EC object with EC_solution_item_rel.
- Specify verification of a problem item of an EC object with EC_problem_item_rel.
Appendix A  Workflow handlers

- Specify verification of a reference item of an EC object with EC_reference_item_rel.
- Specify verification of an addressed by item of an EC object with EC_addressed_by_rel.

_status
Specifies the target object status to be verified:
- Specify any Teamcenter status with ANY.
- Specify no status, or working status, with NONE.
- Specify in process with IN_PROCESS.

This argument is optional.

-check_first_object_only
If specified, only the first object of type specified by -target_type is considered.

This argument is optional.

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.

EXAMPLES
- This example checks for a secondary attachment of type xyz, with a release status of Released, with an IMAN_Specification relation to the target item revision:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-secondary_type</td>
<td>xyz</td>
</tr>
<tr>
<td>-relation</td>
<td>IMAN_Specification</td>
</tr>
<tr>
<td>-status</td>
<td>Released</td>
</tr>
</tbody>
</table>

- This example checks for a primary attachment that is an EngChange revision, currently in process, and attached to the target item revision with an EC_affected_item_rel relation:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-primary_type</td>
<td>EngChange revision</td>
</tr>
<tr>
<td>-relation</td>
<td>EC_affected_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>IN_PROCESS</td>
</tr>
</tbody>
</table>

- This example checks for a primary EngChange revision attachment that is either a change request (CR) or change notification (CN), that is in process, and attached to the target item revision with an EC_affected_item_rel relation.
This checks for both CR and CN EngChange revisions, whether in process or not:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-primary_type</td>
<td>EngChange Revision::CR-CN</td>
</tr>
<tr>
<td>-relation</td>
<td>EC_affected_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>IN_PROCESS</td>
</tr>
</tbody>
</table>

- This example checks for any released secondary xyz attachment with an IMAN_specification relation to the type1 target object:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>type1</td>
</tr>
<tr>
<td>-secondary_type</td>
<td>xyz</td>
</tr>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-status</td>
<td>ANY</td>
</tr>
</tbody>
</table>

- This example checks for a secondary xyz attachment with no status in the Affected Items folder of the target EC revision:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>EngChange Revision</td>
</tr>
<tr>
<td>-secondary_type</td>
<td>xyz</td>
</tr>
<tr>
<td>-relation</td>
<td>EC_affected_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>NONE</td>
</tr>
</tbody>
</table>

- This example checks for a secondary dataset attachment with a working attached to the target item revision. Defining the secondary_type as dataset checks for all dataset types of the defined relation:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>EngChange Revision</td>
</tr>
<tr>
<td>-secondary_type</td>
<td>Dataset</td>
</tr>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-status</td>
<td>NONE</td>
</tr>
</tbody>
</table>

- This example checks for a secondary attachment of type xyz, with a release status of Released, with an IMAN_Specification relation to the target item revision only:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-target_type</td>
<td>ItemRevision</td>
</tr>
<tr>
<td>-secondary_type</td>
<td>xyz</td>
</tr>
<tr>
<td>Argument</td>
<td>Values</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>-relation</td>
<td>IMAN_specification</td>
</tr>
<tr>
<td>-status</td>
<td>Released</td>
</tr>
<tr>
<td>-check_first_object_only</td>
<td></td>
</tr>
</tbody>
</table>
**EPM-check-status-progression**

**DESCRIPTION**

Checks the complete release status progression of a specific object. It can also check whether the object follows a nonlinear progression. A nonlinear progression does not require every subsequent release status of an object to follow the progression path in the same order, though the latest release status must always be greater than the previous release status. For example, if the progression path is Experimental, Quote, Design, Manufacture, Production, the object can achieve Experimental, Quote, and then Production release statuses, skipping Design and Manufacture.

If the workflow process contains several Condition tasks that apply different release statuses at different levels, the value provided in the -status argument can be used. If this argument is not used in this situation, the status applied to the target object is applied to the object.

**SYNTAX**

```
EPM-check-status-progression
[-status=status-being-applied-to-the-target-object]
[-rev=current_rev | previous_rev | latest_rev | greatest_rev]
```

**ARGUMENTS**

- `-status`
  Derives the status being applied to the target object.

- `-rev`
  Checks for one of the following:
  
  - Only the current revision, use `current_rev`. Even if the previous revision is released to a production status, the current revision is released to a lesser status than production.
  
  - The latest release status of the immediately previous revision, use `previous_rev`.
  
  - The greatest release status of all the revisions of the target, use `latest_rev`.
  
  - The greatest release status of all release statuses of the target object, use `greatest_rev`.

**PLACEMENT**

Place on any task action. Typically placed on the Complete action of the perform-signoffs task.

**RESTRICTIONS**

None.

**EXAMPLES**

- This example checks the status of design against the progression path when the workflow process contains several Condition tasks, which apply different release statuses at different levels:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>Design</td>
</tr>
</tbody>
</table>
In this example, consider the scenario:
  
  Progression path: Quote, Experimental, Development, Design, Manufacturing, Production

  IR ABC123

  IR ABC123/001 has Experimental status

  IR ABC123/002 in Working state

  IR ABC123/003 status not yet applied

To release IR ABC123/003 based on the current revision status only, define the following arguments. Previous revision statuses are not checked. Even if the previous revision was released to a Production status the current revision can be released to a lesser status than Production. In this scenario, IR ABC123/003 can be released to Quote status or upward, even though IR ABC123/001 is released to Experimental status.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rev</td>
<td>current_rev</td>
</tr>
</tbody>
</table>

In this example, consider the previous scenario. To release IR ABC123/003 based on the latest release status of its immediate previous revision, define the following arguments. The previous revision is IR ABC123/002, which is in Working state and does not have a status applied. In this case, IR ABC123/003 can be released to Quote status or upward.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rev</td>
<td>previous_rev</td>
</tr>
</tbody>
</table>

In this example, consider the previous scenario. To release IR ABC123/003 based on the last status of the latest released revision, define the following arguments. The latest released revision is IR ABC123/001, its last status was Experimental. In this case, IR ABC123/003 can be released only to Experimental status or upward.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rev</td>
<td>latest_rev</td>
</tr>
</tbody>
</table>

In this example, consider the progression path and values:

  Progression path: Quote, Experimental, Development, Design, Manufacturing, Production.

  IR XYZ123

  IR XYZ123/001 has Design status

  IR XYZ123/002 has Experimental status

  IR XYZ123/003 has Development status
To release IR XYZ123/004 based on the greatest release status among all the revisions of the target object, define the following arguments. IR XYZ123/004 releases as Design.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rev</td>
<td>greatest_rev</td>
</tr>
</tbody>
</table>

The progression path must be manually defined in the `ProgressionPath.plmxmxml` file before the handler can reference the path. The file is stored in the `TC_DATA` directory. Create a backup copy of this file before editing it.

All target types that you want to follow the progression path must be set in this file. A `UserData` block must be created for each type that follows a progression path. For example, to define the progression path for the `ItemRevision`, `PSBOMView`, and `MSWord` types, the `UserData` blocks can be defined as follows:

```xml
<UserData id="id1">
  <UserValue title="Type" value="ItemRevision"/>
  <UserValue title="ReleaseProgressionList" value="Quote,Development,Prototype,Production">
  </UserValue>
</UserData>

<UserData id="id2">
  <UserValue title="Type" value="PSBOMView"/>
  <UserValue title="ReleaseProgressionList" value="Quote1,Development1,Prototype1,Production1">
  </UserValue>
</UserData>

<UserData id="id3">
  <UserValue title="Type" value="MSWord"/>
  <UserValue title="ReleaseProgressionList" value="Quote2,Development2,Prototype2,Production2">
  </UserValue>
</UserData>
```

**Note**

- Add the `UserData` blocks between the `<PLMXML>` and `</PLMXML>` tags.

- Ensure you increment the `UserData id` value when you add a new entry.

- After adding a new `UserData` block, change the value for `Type` to a type you are defining.

- You can modify the value of the release status to meet your requirements.
EPM-check-target-attachments

DESCRIPTION
Checks that the specified target object contains the required attachment with the required status or statuses. You can provide the target object type, relation type, attached object type, and valid statuses as handler arguments.

This handler can be used with an LOV to specify different types of targets and attachments to be checked, requiring just one occurrence of the handler.

Note
Enable debugging functionality for this handler with the TC_HANDLERS_DEBUG environment variable.

For more information about implementing this environment variable, see the Preferences and Environment Variables Reference.

SYNTAX
EPM-check-target-attachments { [-type=target-object-type -att_type=attached-object-type -relation=relation-type] | -lov=lov-name} [-status=valid-status-names | ANY | NONE]

ARGUMENTS
-type
Defines the type of target object to be checked.

-att_type
Defines the type of attachment to be checked.

-relation
Specifies the relation between the target object and the attachment:
- Specify a manifestation relationship with IMAN_manifestation.
- Specify a specification relationship with IMAN_specification.
- Specify a requirement relationship with IMAN_requirement.
- Specify a reference relationship with IMAN_reference.
- Specify a BOM view attachment with PSBOMViewRevision.
- Specify an affected item of an engineering change object with EC_affected_item_rel.
- Specify a solution item of an engineering change object with EC_solution_item_rel.
- Specify a problem item of an engineering change object with EC_problem_item_rel.
- Specify a reference item of an engineering change object with EC_reference_item_rel.
• Specify an **addressed by** item of an engineering change object with \texttt{EC\_addressed\_by\_rel}.

**-status**

Specifies the required status of the attachment. Multiple statuses can be checked by listing valid Teamcenter statuses separated by commas.

\textbf{ANY} checks for any status. \textbf{NONE} checks for working status.

**-lov**

Specifies the list of values (LOVs) used to define which objects are attached to which target objects.

This argument is mutually exclusive of the **-type**, **-att_type**, and **-relation** arguments. It can be used with the **-status** argument to check relation status.

For an overview of using LOVs in handlers, see *Using lists of values (LOVs) as handler arguments*. See the LOV row, next, for the required LOV format.

**LOV**

The LOV can contain multiple optional lines: a line for each type of target to check, followed by one or more multilevel object path lines specifying the relations required for that target type.

If the system does not find any targets for one of the target types, it checks the next target type line.

When a target exists for the specified type, then each relation listed must exist. An error is reported for each relation type missing.

\texttt{[\$TARGET.]target-(class)-or-type-1}

\texttt{relation1.sec-obj-(class)-or-type-in-target-1}

\texttt{relation2.sec-obj-(class)-or-type-in-target-1}

\texttt{[\$TARGET.]target-(class)-or-type-2}

\texttt{relation1.sec-obj-(class)-or-type-in-target-2}

\texttt{relation2.sec-obj-(class)-or-type-in-target-2}

\texttt{...}

**Note**

When using a LOV with this handler, you can improve readability and clarity by indenting the relation lines with spaces. You can also add line numbers in square brackets. For an overview of using LOVs in handlers, see *Using lists of values (LOVs) as handler arguments*.

\texttt{[\$TARGET.]target-(class)-or-type-1}

Defines the type/class of target to check, using a comma-separated list of types/classes in the format shown next.

Target lines are prefixed with \texttt{\$TARGET} or identified by their lack of dots (.).

\texttt{[(Class)[!Type1],[,(Class2)[,Type1[,....]]]}

For example, to specify that all item revisions are checked except software revision:
(ItemRevision)\Software Revision

For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

*relation1.sec-obj-(class)-of-type-in-target-1*

A multilevel object path that must start with a relation (such as `IMAN_specification`). Defines a secondary object that must exist in the specified relation for the target line.

Relation lines always contain a dot (.)

For example, to check that a `UGMASTER` and `UGPART` dataset exist in all revision targets of the design revision type:

$TARGET.Design Revision

`IMAN_specification.UGMASTER`

`IMAN_specification.UGPART`

For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

**PLACEMENT**

Requires no specific placement.

**RESTRICTIONS**

If checking multiple statuses through LOVs, this handler must be used once for each status.

**EXAMPLES**

- This example checks the targeted change revision for an item revision with any status in the **Problem Items** folder:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>EngChange revision</td>
</tr>
<tr>
<td>-att_type</td>
<td>Item revision</td>
</tr>
<tr>
<td>-relation</td>
<td>ECM_problem_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>ANY</td>
</tr>
</tbody>
</table>

- This example checks the targeted change revision for an item revision with no status in the **Affected Items** folder:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>EngChange revision</td>
</tr>
<tr>
<td>-att_type</td>
<td>Item revision</td>
</tr>
<tr>
<td>-relation</td>
<td>ECM_affected_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>NONE</td>
</tr>
</tbody>
</table>

- This example checks the targeted change revision for the **CORP_Part** revision with a released status in the **Solution Items** folder:
Alternatively, you can use these LOV settings:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>EngChange revision</td>
</tr>
<tr>
<td>-att_type</td>
<td>Item revision</td>
</tr>
<tr>
<td>-relation</td>
<td>ECM_solution_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>Released</td>
</tr>
</tbody>
</table>

where the `SYS_EPM_check_target_attachments` LOV contains this data:

$TARGET.EngChange revision

`ECM_solution_item_rel.CORP_Part Revision`

- This example checks the targeted change revision for an item revision for any status of the following statuses (`Concept Approval`, `Funding Approval`, `Design Approval`) in the Solution Items folder:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>EngChange revision</td>
</tr>
<tr>
<td>-att_type</td>
<td>Item revision</td>
</tr>
<tr>
<td>-relation</td>
<td>ECM_solution_item_rel</td>
</tr>
<tr>
<td>-status</td>
<td>Concept Approval,Funding Approval,Design Approval</td>
</tr>
</tbody>
</table>

- This example checks the targeted change revision for an item revision in the Solution Items folder, irrespective of status:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>EngChange revision</td>
</tr>
<tr>
<td>-att_type</td>
<td>Item revision</td>
</tr>
<tr>
<td>-relation</td>
<td>ECM_solution_item_rel</td>
</tr>
</tbody>
</table>

- This example performs specific relation checks for particular revision type targets and other relation checks for the remaining revision types all with no status:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-love</td>
<td>SYS_EPM_check_target_attachments</td>
</tr>
<tr>
<td>-status</td>
<td>NONE</td>
</tr>
</tbody>
</table>

where the `SYS_EPM_check_target_attachments` LOV contains this data:
<table>
<thead>
<tr>
<th>LOV contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[01] Software Revision, DocumentRevision</td>
<td>Check that any software and document revision targets have a text dataset attached in the IMAN_specification relation.</td>
</tr>
<tr>
<td>[02] IMAN_specification.Text</td>
<td>Check that any DocumentRevision targets also have a Word, Excel OR PowerPoint dataset attached in the IMAN_specification relation.</td>
</tr>
<tr>
<td>[03] DocumentRevision</td>
<td>Check that any other targets of class ItemRevision, (in other words, that are not Software Revision or DocumentRevision) have a UGMASTER and UGPART attached in the IMAN_specification relation.</td>
</tr>
<tr>
<td>[04] IMAN_specification.Word, Excel, PowerPoint</td>
<td>Check that any revision targets also have a project item attached to the custom Proj relation.</td>
</tr>
<tr>
<td>[05] (ItemRevision)!Software Revision! DocumentRevision</td>
<td></td>
</tr>
<tr>
<td>[06] IMAN_specification.UGMASTER</td>
<td></td>
</tr>
<tr>
<td>[07] IMAN_specification.UGPART</td>
<td></td>
</tr>
<tr>
<td>[08] (ItemRevision)</td>
<td></td>
</tr>
<tr>
<td>[09] Proj.Project</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

This sample LOV uses optional line numbers in square brackets and the relation lines are indented for clarity.
**EPM-check-target-object**

**DESCRIPTION**
Checks the status of the object being pasted to determine whether to allow the action.

**Note**
Enable debugging functionality for this handler with the `TC_HANDLERS_DEBUG` environment variable.

For more information about implementing this environment variable, see the Preferences and Environment Variables Reference.

**SYNTAX**
```
EPM-check-target-object -status=status-name | -status_allow=
status-name | -status_disallow=status-name
```

**ARGUMENTS**

- **-status**
  Defines statuses to check against target objects. If a potential target matches any of the statuses defined with this argument, paste is unavailable.

  A warning message is displayed indicating noncompliance to the business rule when you click **OK**. Additionally, if the argument passed to the handler is incorrect, this warning message is also displayed when you click **OK**.

  Accepts one or more valid Teamcenter status names.

  Indicate any status with one of the following:

  * | all | ALL | any | ANY

  Indicate no status with one of the following:

  null | NULL | none | NONE

- **-status_allow**
  Defines statuses to check against target objects. If a potential target matches any of the statuses defined with this argument, paste is available.

  Accepts one or more valid Teamcenter status names.

  Indicate any status with one of the following:

  * | all | ALL | any | ANY

  Indicate no status with one of the following:

  null | NULL | none | NONE

- **-status_disallow**
  Defines statuses to check against target objects. If a potential target matches any of the statuses defined with this argument, paste is unavailable. Can use in place of **-status** for clarity. A warning message is displayed indicating noncompliance to the business rule when you click **OK**. Additionally, if the argument passed to the handler is incorrect, this warning message is also displayed when you click **OK**.

  Accepts one or more valid Teamcenter status names.

  Indicate any status with one of the following:
Appendix A  Workflow handlers

* | all | ALL | any | ANY
Indicate no status with one of the following:
null | NULL | none | NONE

**PLACEMENT**
Place on the **Perform** action of the root task.

**RESTRICTIONS**
None.

**EXAMPLES**
- This example allows any target to be attached with a status of **Pending** or with no status (work in progress):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status_allow</td>
<td>Pending, NONE</td>
</tr>
</tbody>
</table>

- This example disallows any targets from being attached with a status of **Released** or **Obsolete**:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status_disallow</td>
<td>Released, Obsolete</td>
</tr>
</tbody>
</table>
EPM-check-validation-result

DESCRIPTION
Evaluates the validation result of each target before releasing the object. The handler first looks for all results relative to all targets. If no validation result is found, or any result is outdated or failed, the handler reports the corresponding error message and returns an EPM_nogo and the workflow is cancelled. If all validation results are successful and current, the handler returns an EPM_go and the workflow proceeds.

There are five situations in which validation results are checked:

- If the target object is an item revision, the handler finds all the validation targets by the closure rule specified in the NX Agent and then finds all the results relative to these validation targets.

- If the target object is an item, the handler runs on the latest revision, searching for validation results as specified in the previous situation. You may also supply a handler specifying the item revisions. After the first handler runs, the second handler runs on the specified item revisions as specified in the previous situation.

- If the target object is a dataset, the handler finds the validation results relative to the dataset.

- If the target object is a folder, the handler includes all secondary objects under the folder in its search for validation results.

- If there are multiple objects as targets, (for example, if multiple item revisions are selected as targets of a workflow), the handler finds all the validation results relative to all the validation targets by closure rule.

For more information, see the Validation Manager Guide.

SYNTAX
EPM-check-validation-result

ARGUMENTS
None.

PLACEMENT
Place on the Start action of the root task. The workflow process is aborted if a target is not validated, or if its validation result is not Pass.

An alternative is to place on the Complete action of the root task. The release status is not added to a target if it is not validated, or if its validation result is not Pass.

RESTRICTIONS
None.
EPM-check-validation-result-with-rules

DESCRIPTION

Leverages validation rule and validation object applications from the workflow process and checks target NX datasets validation result status. To add this handler to a workflow process template, the user must have a well-defined validation rule set file that best describes the user’s business process in terms of what NX datasets should run what checks at what time and what conditions that the check must meet. The handler returns a EPM_go or EPM_nogo decision based on overall result status of the verification (EPM_go is returned only when all target NX datasets satisfy all rules defined in validation rule set file).

The handler logs validation rules and validation result checks. The format of the log file name is First-target-name_Time-stamp. The log file is stored in the directory specified by the TC_TMP_DIR environment variable. If TC_TMP_DIR is not defined, it is stored in the %TEMP% directory (Windows) or /tmp directory (Linux).

Note

The system will not process a log file name longer than 32 characters when the TC_Allow_Longer_ID_Name preference is set to false. In this situation, if the log file name is longer than 32 characters, the log file name is automatically truncated.

For more information, see the Validation Manager Guide.

SYNTAX

EPM-check-validation-result-with-rules
-rule_item_revision=item-rev-id [-current_event=event-value] [-pass_item_revision_only]

ARGUMENTS

-rule_item_revision
The item/revision ID that the validation rule set dataset is attached under.

-current_event
A value that is used to select validation rules from the rule file by comparing with the event values list of each rule. When -current_event is not provided, all rules from the rule file are selected at the first step. When a rule is defined without the event values list, the rule is also selected at the first step. The event values list can contain a wildcard (* only). The event values list also can be marked as exclusive (inclusive by default).

-pass_item_revision_only
When this argument is added to an input list, only item revision targets are passed to the handler. NX datasets are searched from each item revision and verified according to rules.

PLACEMENT

Do not place this handler on the Start action of the root task. Place on any other action of the root task. Or on any action on any other task.
Note
If the handler is placed on the **Start** action of the root task, and the handler fails to complete, the workflow process itself is not created. No log file is created.

**RESTRICTIONS**

- **-rule_item_revision** cannot be NULL.
Appendix A  Workflow handlers

EPM-hold

DESCRIPTION

Pauses the task, requiring the user to perform an action on the task before the task can complete. Typically, a task completes automatically once started. EPM-hold prevents this automatic completion.

Use this rule handler with custom tasks that require customized Perform actions, or to require the user to manually perform a Complete action to complete the task.

Using this rule handler with the argument set to true returns EPM_nogo until a Complete action is triggered manually from the user interface, in which case the handler returns EPM_go.

You can set the argument to false, though this deactivates the handler. In fact, setting the argument to false is what the Complete action does temporarily, allowing this handler to return EPM_go.

Configuring a task to display forms using EPM-display-form, EPM-hold, and EPM-create-form

To configure a task to display a form when a user performs a specified action, use the EPM-hold handler. This handler pauses the task, requiring the user to perform an action on the task before the task can complete. Without the use of this handler, a task completes automatically once started.

To create an instance of a specified form and attach the form to the specified task, use the EPM-create-form handler.

Therefore, the EPM-create-form handler creates the form when the Start action is initiated, the EPM-display-form handler displays the form when the Perform action is initiated, and the EPM-hold handler prevents the task from automatically completing, allowing the form to be completed by the user.

Variations on the above example may be required for a more sophisticated interaction when it is required that the task not complete until required fields are entered in the form. This type of configuration requires the creation of customized rule handlers.

SYNTAX

EPM-hold true|false

ARGUMENTS

true or false

PLACEMENT

Place on the Complete action of any task with which you want the user to interact before the task completes.

RESTRICTIONS

None.

ADDITIONAL INFORMATION

- By default, this handler is placed in the Do task template, pausing the task to allow the Do Task dialog box to display when the user performs the Perform action on a selected Do task.

- Use this handler with custom tasks that present custom forms when the user performs the Perform action.
For information about configuring custom tasks to present custom forms when the **Perform** action is invoked, see the description of the **EPM-display-form** handler.
EPM-validate-target-objects

DESCRIPTION
Restricts the types of objects that can be added as target objects. It always prevents the Home, Newstuff, and MailBox folders from being added as target objects.

Note
Enable debugging functionality for this handler with the TC_HANDLERS_DEBUG environment variable.

For more information about implementing this environment variable, see the Preferences and Environment Variables Reference.

SYNTAX
EPM-validate-target-objects
[-allowed_type = type-of-workspace-object[, type-of-workspace-object2,..]]
[-disallowed_type = type-of-workspace-object[, type-of-workspace-object2,..]]
[-latest_rev]

ARGUMENTS
-allowed_type
Defines the type of objects that can be added as target objects to a workflow process. You can define more than one type by using commas between the types. This argument is optional.

Accepts valid Teamcenter object types, such as ItemRevision, UGMASTER, and UGPART.

If this argument is specified as ItemRevision, any type of item revision (for example, DocumentRevision, and so on, and any custom item revision types) is allowed.

Does not accept bracketed () class notation to distinguish between classes and types.

-disallowed_type
Defines the type of objects that cannot be added as target objects to a workflow process. You can define more than one type by using commas between the types.

Accepts valid Teamcenter object types, such as ItemRevision, UGMASTER, and UGPART.

If this argument is specified as ItemRevision, any type of item revision (for example, DocumentRevision, and so on, and any custom item revision types) is disallowed.

-latest_rev
Ensures any revisions added to the workflow process are the latest revision within their owning item. This argument is optional.

PLACEMENT
Place on any action in any task.

RESTRICTIONS
None.

EXAMPLES
- This example allows only item revisions as targets:
- **Argument** | **Values**
---|---
-allowed_type | ItemRevision

- This example allows item revisions, document revisions, and a custom part revision type as targets:

- **Argument** | **Values**
---|---
-allowed_type | ItemRevision, DocumentRevision, G4GTACPartRevision

- This example allows only the latest item revisions as targets:

- **Argument** | **Values**
---|---
-allowed_type | ItemRevision
-latest_rev
**ERP-check-effective-date-RH**

**DESCRIPTION**
Checks the **Effect In** date on the release status attached to the process does not have a value before the current date.

**SYNTAX**

```
ERP-check-effective-date-RH
```

**ARGUMENTS**
None.

**PLACEMENT**
Place on the **Perform Signoff** task.

**RESTRICTIONS**
None.
ERP-check-target-status-RH

DESCRIPTION
Checks that the release status for target item revisions is specified.

SYNTAX
ERP-check-target-status-RH -status_name=name

ARGUMENTS
- status_name
Specifies the name of the release status.

RESTRICTIONS
None.
ERP-validate-data-RH

DESCRIPTION
Applies the validation criteria specified in the mapping schema on all forms attached to the process’s transfer folders and related BOMComponent data. The following validations are performed:

- For each attribute:
  - If the attribute parameter is required, the field must have a value.
  - If the attribute definition has an LOV, the value in the field must match one in the list. Although this is checked at entry time, this allows for LOVs that changed in the mapping since the data was originally entered.
  - For string attributes, the length of string entered must be no more than that defined in the schema.
  - If there is a custom validation function defined using the custom_check attribute parameter, call the function.

- For each BOMHeader to be sent to ERP:
  - Check a corresponding BOMView revision of the correct type exists, as described for the SAP-check-forms-attached-RH handler.
  - Check all components with the same item ID have the same attribute values (for those attributes specified in the mapping schema, except quantity).
  - Check component attribute values conform to parameters in the mapping schema (mandatory, LOV, length). Although LOVs cannot be presented to the user for Structure Manager notes, values can still be validated with this handler.

SYNTAX
ERP-validate-data-RH

ARGUMENTS
None.

PLACEMENT
Call this handler after you attach data with ERP-attach-targets-AH. Place this handler on the perform-signoff task.

RESTRICTIONS
None.
**invoke-system-rule**

**DESCRIPTION**

Executes an external command (specified with the `-system` argument) such as Perl scripts, shell scripts, or external ITK programs, then continues or halts the workflow process based on the return code of the external command.

Use this handler for increased control of the workflow process. For example, to synchronize NX attributes and structure with Teamcenter, or to generate JT tessellation from CAD files.

This handler writes process-related information to an XML file. The file is passed to the external script or program as `-f XML-file-name`. APIs are provided (in the form of Perl modules) to read the XML file and perform functions on its data objects. The APIs are located in the `Workflow.pm` file in the `TC_ROOT/bin/tc` directory.

Write Perl scripts (for example, `TC_ROOT/bin/iman_check_renderings.pl` for background tessellation of CAD data) using the provided APIs to read the XML file and perform required functions on its data objects. Then use the Perl script as the value of the `-system` argument in the workflow process template.

**Note**

Siemens PLM Software recommends you place the Perl scripts in the `TC_ROOT/bin` folder.

Alternatively, you can place the script in an alternate location and provide an absolute path to the location (for example, `c:\temp\test.bat`). However, using an absolute path requires that you update the template if there are any changes. In the previous example, `c:\temp\test.bat` is a path on a Windows platform. If you were to change to a UNIX platform, the template would need to be updated. This second method is not recommended.

The handler returns a code that is mapped to:

- **EPM_go** when the external script returns 0 or **EPM_go** and no other errors are returned
- **EPM_nogo** when the external script/program returns error or **EPM_nogo**
- **EPM_undecided** when the external script/program returns **EPM_undecided**

**SYNTAX**

```
invoke-system-rule -system=name-of-the-external-program
[-trigger_on_go= [task:action]]
[-trigger_on_nogo= [task:action]]
[-trigger_on_undecided= [task:action] [-skip_unreadable_objs]]
[-change_status_on_go= [old-status-name]:[new-status-name]]
[-change_status_on_nogo= [old-status-name]:[new-status-name]]
[-change_status_on_undecided= [old-status-name]:[new-status-name]]
[-add_occurrence_notes] [-signoff=signoff-comment]
[-responsible_party= [User:responsible-party]| [Task:task-name]]
[-reviewer= [User:user-id] | [Group:group] | [Role:role] | [Level:level]]
[-send_mail= user-ids] [-initiate_process] [-where_used=item-revision-type]
[-expand=item-revision-type] [-list_sibling_processes= wildcarded-procname]
```
**Appendix A  Workflow handlers**

[-depth=maximum-recursion-depth] [-debug]

**ARGUMENTS**

- **-system**
  Name of the external executable. This executable can be an external Perl script that reads and modifies the XML file that this handler writes, or an ITK program to perform specific functionality.
  This argument is required.

- **-trigger_on_go**
  Triggers an action in the same workflow process when **EPM_go** is returned.
  Trigger an action in another task by specifying an action and task name. If another task name is unspecified, the specified action in the current task is triggered.
  The system supports the following actions:

  **ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP, ABORT, REFUSE, UNDO, REJECT, APPROVE, PROMOTE, DEMOTE.**

  Action names are not case sensitive.
  Task names cannot contain a colon or a period. If the task name is ambiguous (for example, **select-signoff-team**), hierarchical notation is required.
  This argument is optional.

- **-trigger_on_nogo**
  Triggers an action in the same workflow process when **EPM_nogo** is returned.
  Trigger an action in another task by specifying an action and task name. If another task name is unspecified, the specified action in the current task is triggered.
  The system supports the following actions:

  **ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP, ABORT, REFUSE, UNDO, REJECT, APPROVE, PROMOTE, DEMOTE.**

  Action names are not case sensitive.
  Task names cannot contain a colon or a period. If the task name is ambiguous (for example, **select-signoff-team**), hierarchical notation is required.
  This argument is optional.

- **-trigger_on_undecided**
  Triggers an action in the same workflow process when **EPM_undecided** is returned.
  Trigger an action in another task by specifying an action and task name. If another task name is unspecified, the specified action in the current task is triggered.
  The system supports the following actions:

  **ASSIGN, START, PERFORM, COMPLETE, SUSPEND, RESUME, SKIP, ABORT, REFUSE, UNDO, REJECT, APPROVE, PROMOTE, DEMOTE.**

  Action names are not case sensitive.
  Task names cannot contain a colon or a period. If the task name is ambiguous (for example, **select-signoff-team**), hierarchical notation is required.
  This argument is optional.
-skip_unreadable_objs
Unreadable objects are not processed. The handler does not attempt to write information about unreadable objects into the XML file; the objects are skipped.
If this argument is not specified, the handler displays an error when a failure occurs when there is no read access.

-change_status_on_go
Adds, removes, or changes the status of attachments when EPM_go is returned.
Both the old and new status names are optional.

  - If both status names are specified, the new status name replaces the old status name.
  - If only the new status name is specified, the corresponding status is added.
  - If only the old status name is specified, the corresponding status name is removed.
  - If neither status name is specified, no action is taken.

If a value is not provided for this argument, the value set by the external Perl script is read.

-change_status_on_nogo
Adds, removes, or changes the status of attachments when EPM_nogo is returned.
Both the old and new status names are optional.

  - If both status names are specified, the new status name replaces the old status name.
  - If only the new status name is specified, the corresponding status is added.
  - If only the old status name is specified, the corresponding status name is removed.
  - If neither status name is specified, no action is taken.

If a value is not provided for this argument, the value set by the external Perl script is read.

-change_status_on_undecided
Adds, removes, or changes the status of attachments when EPM_undecided is returned.
Both the old and new status names are optional.

  - If both status names are specified, the new status name replaces the old status name.
  - If only the new status name is specified, the corresponding status is added.
  - If only the old status name is specified, the corresponding status name is removed.
  - If neither status name is specified, no action is taken.
If a value is not provided for this argument, the value set by the external Perl script is read.

**-add_occurrence_notes**
Sets occurrence notes of target assemblies. Can be used in combination with the **-expand** argument to set **OccurrenceNotes** for components of assembly structures.
This argument is optional.

**-signoff**
The signoff decision is set depending on the return code of the external program:

- 0=Approve
- 1=Reject
- 2=No Decision

If a value is not provided for this argument, the value set by the external Perl script is read.
This argument is optional.

**-responsible_party**
Assigns a responsible party. If no user ID is specified for this argument, the value set by the external Perl script is read.
This argument is optional.

**-reviewer**
Assigns a reviewer for a release level. If no reviewer is specified for this argument, the value set by the external Perl script is read.
This argument is optional.

**-send_mail**
Sends target, reference, or sibling objects through program mail. If one or more user IDs are defined for this argument, the workflow process is sent to the specified users through program mail.
Separate multiple user IDs with a space or a comma.
If no user IDs are defined for this argument, the recipients and the contents of the envelope set by the external Perl script is read.
This argument is optional.

**-initiate_process**
Initiates a workflow process for another object. Target objects are defined by the values set by the external Perl script.
This argument is optional.

**-where_used**
Reports the where-used of item and item revision target attachments by writing the hierarchy of all parent and grandparent assemblies of item and item revision target attachments into the XML file to allow the external Perl script to perform required functions. If an **ItemRevision** type is specified, the type argument is compared to the corresponding item revision type. For example, **ItemRevision** matches objects of the **Item** type. If an item revision type is specified, the parent assemblies of only those target attachments that match this type are listed.
This argument is optional.

-expect
Reports the assembly of item and item revision target attachments by writing the hierarchy of all child and grandchild components of item and item revision target attachments into the XML file to allow the external Perl script to perform required functions.

If an ItemRevision type is specified, the type argument is compared to the corresponding item revision type. For example, ItemRevision matches objects of the Item type. The assembly structure is expanded for all item revision of all matching item target attachments.

If an item revision is specified, the child components of only those target attachments are listed that match this type.

This argument is optional.

-list_sibling_processes
Writes information regarding processes that belong to the same Change item into the XML file to allow the external Perl script to perform required functions. The information concerns processes sharing the same Change item as a reference attachment.

If a process template name is specified in the procedure definition, only the processes that match the procedure name are included.

This argument is optional.

-depth
Increases the maximum incursion depth. The -trigger_on_go or -initiate_process arguments could cause the triggered action to use the same handler in a deeper level of recursion. If this is intended, the maximum level of recursion must be set to the desired number. If necessary, it can be disabled by setting it to 0. The default is set to 1, to avoid infinite loops.

This argument is optional.

-debug
Enables debugging. Each occurrence of this argument increases the debug level by one. Debug messages are written to the Teamcenter error stack for display in the rich client user interface, as well as written to the syslog file.

This argument is optional.

PLACEMENT

Place only on a Review task.

RESTRICTIONS

Do not add to a workflow process containing any handler using resource pools.

EXAMPLES

This example shows how to execute the iman_check_renderings_pl script using the -system argument. Do not list the file extension in the value. This value runs either the iman_check_renderings_pl.bat (Windows) or iman_check_renderings_pl (UNIX) script, depending on which platform the server is running.
Appendix A  Workflow handlers

Note

The script should be placed in the TC_ROOT/bin directory.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-system</td>
<td>iman_check_renderings_pl</td>
</tr>
</tbody>
</table>
SAP-check-forms-attached-RH

DESCRIPTION

Makes the following checks:

- For each BOM, check that the master data for each component and the assembly itself is created in ERP at the plant specified in the associated BOMHeader form or is a target of the current process. This prevents the upload failing, which it would if the component data did not already exist. This handler does not make any calls to ERP; it simply checks the Sent to ERP box.

  Note

  If the process has both component and assembly item revisions, the material data is created first, and then the BOMs.

- For each BOMHeader form, there must be a corresponding BOM view revision with the view type specified by the TC_view_type attribute in the form.

- Complete sets of ERP forms are attached to each item revision as a target of the process. The mapping schema allows data for an erp_object, typically plant-specific, to be split across several form types. As the upload is expecting a complete set of attribute values for an erp_object, a complete set of forms must be transferred (for example, an instance of each form type defined for the erp_object).

- For a BOM, check that the parent and all components have had their master data Sent to ERP for the plant in which the BOM is created or are part of the process.

  Note

  If the erp_object defines a key field with the is_key_fld parameter, the value in this field is used to distinguish between different instances of data for the same erp_object. For example, all forms having value 1000 in the plant field for form types with erp_object PlantSpecific constitute the set of forms defining the plant-specific data for plant 1000.

This handler only searches for ERP forms defined in the mapping schema attached by the relation types listed by the -reln_names argument. This list should be consistent with that used in the ERP-attach-targets-AH. Only those forms whose state has not yet been transferred to ERP (for example, those for which the Sent_to_ERP field is empty) are checked.

SYNTAX

SAP-check-forms-attached-RH -reln_names = reln1,reln2,...

ARGUMENTS

-reln_names

A comma-separated list of the relation types used to relate ERP forms to item revisions.
Appendix A  Workflow handlers

Note
Relation names are case sensitive and should be named, for example, 
tc_specification not TC_Specification.

ERP_Data is the special relation supplied for attaching ERP forms.

PLACEMENT
Place this handler on the Review task.

RESTRICTIONS
None.
SAP-check-forms-to-download-RH

DESCRIPTION
Checks to make certain all form sets in transfer folders are valid, with the same rules as the SAP-check-forms-attached-RH rule handler. However, the SAP-check-forms-to-download-RH handler is intended for final checking of the form sets to be sent, rather than an initial input validation set.

SYNTAX
SAP-check-forms-to-download-RH

ARGUMENTS
None.

PLACEMENT
Call this handler after data is attached using the ERP-attach-targets-AH handler. Place this handler on the Perform Signoff task.

RESTRICTIONS
None.
TCX-check-approver

DESCRIPTION
Compares users of set A to users of set B. If there is a user in both sets, the handler displays an error or warning message. In this case, **EPM\_nogo** is returned. The members of each set are assigned using the parameters.

SYNTAX

```
TCX-check-approver -A_level = [levelname | $PREVIOUS | $NEXT]
-A_user = [userid | $USER] -A_jobowner -B_level = [levelname | $PREVIOUS | $NEXT]
-B_user = [userid | $USER] -B_jobowner -stop= [Y | N]
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Values</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A_level</td>
<td>[levelname</td>
<td>$PREVIOUS</td>
</tr>
<tr>
<td>-A_user</td>
<td>[userid</td>
<td>$USER]</td>
</tr>
<tr>
<td>-A_jobowner</td>
<td>None.</td>
<td>The owner of the job is added the quantity of A.</td>
</tr>
<tr>
<td>-B_level</td>
<td>[levelname</td>
<td>$PREVIOUS</td>
</tr>
<tr>
<td>-B_user</td>
<td>[userid</td>
<td>$USER]</td>
</tr>
<tr>
<td>-B_jobowner</td>
<td>None.</td>
<td>The owner of the job is added the quantity of B.</td>
</tr>
<tr>
<td>-stop</td>
<td>[Y</td>
<td>N]</td>
</tr>
</tbody>
</table>

PLACEAMENT
Place at the start of the workflow.

RESTRICTIONS
None.
TCX-check-bom-precise

DESCRIPTION
Checks whether all BOM view revisions are precise.

SYNTAX
TCX-check-bom-precise [-stop=[y|n]] [-maxdepth=depth]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-stop</td>
<td>Valid type of form.</td>
</tr>
<tr>
<td>-maxdepth</td>
<td>Levels to be checked. The value 0 corresponds to all levels.</td>
</tr>
</tbody>
</table>

PLACEMENT
Must be set in the Complete action of the perform-signoffs task.

RESTRICTIONS
None.
TCX-check-bomchild-statuslist

DESCRIPTION
Checks all components of a target assembly in a BOM view revision for a valid status.

SYNTAX
TCX-check-bomchild-statuslist -rule=configurationrule
-statelist=status[,status] [—check_job=[y | n]]. [—log=[error<all]]
-stop=[y | n]] [-maxdepth=depth]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rule</td>
<td>Configuration rule.</td>
<td></td>
</tr>
<tr>
<td>-statelist</td>
<td>Comma-separated list of valid status names.</td>
<td></td>
</tr>
<tr>
<td>-check_job</td>
<td>Defines the terms of the component status.</td>
<td></td>
</tr>
<tr>
<td>-log</td>
<td>Log data record.</td>
<td></td>
</tr>
<tr>
<td>-stop</td>
<td>Warning in the event of an error (=n) or Workflow with error stop (=y)</td>
<td>Warning in the event of an error (=n) or Workflow with error stop (=y)</td>
</tr>
<tr>
<td>-maxdepth</td>
<td>Level in the assembly to be checked.</td>
<td></td>
</tr>
</tbody>
</table>

- n = All components must possess a correct status or be target objects in the same workflow.
- y = All components must possess a correct status and be target objects in the same or another workflow.
- error = Record incorrect components only.
- all = Record all component examinations.
- 1 = First level
- 2 = Second level
- 0 = All levels

PLACEMENT
Must be set in the Complete action of the perform-signoffs task. After this handler is used, no changes should be made to the BOM view revisions.

RESTRICTIONS
None.
TCX-check-comps-against-pattern

DESCRIPTION
Checks the components against a specified pattern, where components include **Item**, **ItemRevision**, **Dataset**, **BOMView**, and **BOMViewRevision**. The pattern is as follows: the item ID should be eight characters and all characters should be digits. In addition, all of the target components should not have a status attached to it.

SYNTAX
TCX-check-comps-against-pattern -mode=[list | check_only]  
-file=dataset-name

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mode</td>
<td>Defines how the check should be performed. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• <strong>list</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lists all the components in the newly created dataset defined by the file argument. The dataset is attached as a reference to the workflow process.</td>
</tr>
<tr>
<td></td>
<td>• <strong>check_only</strong></td>
</tr>
<tr>
<td></td>
<td>The dataset named reference is replaced with the latest information.</td>
</tr>
<tr>
<td>-file</td>
<td>Specifies the name of the dataset that should be attached as a reference to the workflow process.</td>
</tr>
</tbody>
</table>

PLACEMENT
Must be set in the **Start** action.

RESTRICTIONS
Handler should not be put after the **Complete** action.
TCX-check-datasets

DESCRIPTION
Checks that the dataset attached to the target item revision is a given type. This handler also checks that the name of the dataset matches the given pattern.

SYNTAX
TCX-check-datasets
-type=[dataset-type -search_dataset_name=] |Pattern1:Pattern2 |
-check_include_dataset_name=name-of-dataset [-stop=y | n] 
[-dataset_to=target | job | item]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>Dataset type.</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>-search_dataset_name</td>
<td>The pattern of the dataset name. Multiple patterns can be separated using a semicolon (;).</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>-check_include_dataset_name</td>
<td>Identifies the dataset. This argument accepts a single value only.</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>-stop</td>
<td>Determines whether or not to stop checking if the datasets don’t match the search criteria.</td>
<td>y</td>
<td>No</td>
</tr>
<tr>
<td>-dataset_to</td>
<td>Defines the location of the text dataset containing the search results. Valid values are target object (target), an attachment to the item revision (item), or an attachment to the job.</td>
<td>target</td>
<td>No</td>
</tr>
</tbody>
</table>

PLACEMENT
Requires no specific placement.

RESTRICTIONS
All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

<table>
<thead>
<tr>
<th>Argument</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-type</td>
<td>UGPART</td>
</tr>
<tr>
<td>-search_dataset_name</td>
<td>EZ;GZ</td>
</tr>
<tr>
<td>Argument</td>
<td>Values</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>-check_include_dataset_name</td>
<td>ANT</td>
</tr>
<tr>
<td>-dataset_to</td>
<td>Item</td>
</tr>
</tbody>
</table>
TCX-check-itemrev-status

DESCRIPTION
Checks the status of target item revisions.

SYNTAX
TCX-check-itemrev-status [-status=status-type [-unreleased] [-latest] [-targetstatus]]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-status</td>
<td>All target objects should be released with this status type</td>
<td>No</td>
</tr>
<tr>
<td>-unreleased</td>
<td>All target objects should be without a release status</td>
<td>No</td>
</tr>
<tr>
<td>-latest</td>
<td>Validates that the target item revision is the latest released item revision.</td>
<td>No</td>
</tr>
<tr>
<td>-targetstatus</td>
<td>Stops the handler if any of the target objects are released</td>
<td>No</td>
</tr>
</tbody>
</table>

PLACEMENT
Requires no specific placement.

RESTRICTIONS
None.
TCX-check-jobowner

DESCRIPTION
Checks that the owner of a certain stage (level) of a release process cannot delegate approval.

SYNTAX
TCX-check-jobowner [-who=jobowner] [-level=this]

ARGUMENTS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-who</td>
<td>User ID to examine. Currently, jobowner is the only valid value. This parameter is reserved for possible future extensions.</td>
</tr>
<tr>
<td>-level</td>
<td>Level to examine. Currently, this is the only valid value. This parameter is reserved for possible future extensions.</td>
</tr>
</tbody>
</table>

PLACEMENT
Must be set in the Finish action of the select-signoff-team task.

RESTRICTIONS
The current default behavior allows the user to delegate their approval after the select-signoff-team task completes. The Finish action of the select-signoff-team task does not get called again; therefore, the newly assigned user is not validated. To allow this validation, Siemens PLM Software recommends that you include this check in the Finish action of the select-signoff-team task. Because the handler reports an error only after the user has approved, and a delegation at this point is not possible, the release process must be deleted and restarted.
TCX-check-prev-itemrev-status

DESCRIPTION
Tests the previous item revision for release status, revises the item revisions, and releases them with a status of 60 (release status).

SYNTAX
TCX-check-prev-itemrev-status -status=status | -latest | -unreleased | -firstrev

ARGUMENTS
-status
-status=release-status or any. When the argument is set to any, the handler checks the preceding revision for a status. If the previous item revision’s status is set, then the handler returns EPM_go. The default is any.

-latest
If multiple statuses are assigned to an item revision, the latest revision status is used.

-unreleased
If no preceding item revisions have been released, do not take any action.

-firstrev
If the first item revision’s argument is not set, the handler returns EPM_nogo. If the argument is set, EPM_go is returned and the first revision can be released.

Only the immediately preceding item revision is checked. If there is no preceding item revision, the -firstrev parameter determines the behavior. Past revisions are not considered.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
None.
TCX-check-signoff

DESCRIPTION
Checks the signoff users against signoffs from other levels.

SYNTAX
TCX-check-signoff -level=$PREVIOUS | $NEXT

ARGUMENTS
-level =$PREVIOUS | $NEXT
When the argument is set to $PREVIOUS, the handler checks the level before the current level. If set to $NEXT, the handler checks the level after the current level. The default is $PREVIOUS.

PLACEMENT
Place on the Start action of the root task.

RESTRICTIONS
None.
**TCX-check-status**

**DESCRIPTION**
Initiates a workflow process if the current and the previous revisions have a valid release status.

**SYNTAX**

```
TCX-check-status [-previous_status={any|none|Statuslist}] [-previous_check=all|last] [-current_status={any|none|status-list}] [-stop={Y|N}] [-current_check=all|last]
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-previous_status</td>
<td>Status on the predecessor revision to be tested (last release status of the predecessor revision).</td>
</tr>
<tr>
<td>-current_status</td>
<td>Status on the target revision to be checked.</td>
</tr>
<tr>
<td>-previous_check</td>
<td>For the previous revision, check either the last release status or the entire list for valid status.</td>
</tr>
<tr>
<td>-current_check</td>
<td>For the target revision, check either the last release status or the entire list for valid status.</td>
</tr>
<tr>
<td>-stop</td>
<td>Set to Y to stop the process or N to continue with warning.</td>
</tr>
</tbody>
</table>

**PLACEMENT**
Place on the **Start** action of a root task.

**RESTRICTIONS**
None.
## TCX-has-target-drawing

**DESCRIPTION**
Checks that the target item revisions have a CAD dataset associated with it. If the item revisions do not have an attached dataset, the handler returns an error.

**SYNTAX**

```
TCX-has-target-drawing
```

**ARGUMENTS**
None.

**PLACEMENT**
Requires no specific placement.

**RESTRICTIONS**
None.
validate-for-checkedout-asmaintained-physicalpartrevision

DESCRIPTION
Checks if any physical parts are checked out in the as-maintained structure by a user other than the creator or submitter of the workflow process.

SYNTAX
validate-for-checkedout-asmaintained-physicalpartrevision

ARGUMENTS
None.

PLACEMENT
Place at the entry of the workflow to validate that the target structure does not contain any checked out physical parts.

RESTRICTIONS
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager is licensed and installed.
**validate-for-checkedout-physicalpartrevision**

**DESCRIPTION**
Validates that the as-built structure does not contain any checked-out physical parts by any user other than the one submitting the physical part to a workflow.

**SYNTAX**
```
validate-for-checkedout-physicalpartrevision
```

**ARGUMENTS**
None.

**PLACEMENT**
Place at the entry of the workflow to validate that the target structure does not contained any checked out physical part revisions.

**RESTRICTIONS**
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager or As-Built Manager is licensed and installed.
validate-for-class

DESCRIPTION
Validates that the item revision submitted to the workflow is a physical part revision. If it is a physical part revision, the handlers returns `EPM_go`. If it is not a physical part revision, the handler displays an error, returns the decision as `EPM_nogo`, and stops further processing.

SYNTAX
`validate-for-class -class name=class-name`

ARGUMENTS
- **-class name**
  Specifies the class name to validate.

PLACEMENT
Place at the entry of the workflow to validate that the target object is the physical part revision for the as-built structure traversal.

RESTRICTIONS
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager or As-Built Manager is licensed and installed.
validate-for-latest-asmphysicalpartrevision

DESCRIPTION
Checks if the target physical part revision is the latest revision.

SYNTAX
validate-for-latest-asmphysicalpartrevision

ARGUMENTS
None.

PLACEMENT
Place at the entry of the workflow to validate that the target physical part revision is the latest one.

RESTRICTIONS
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager is licensed and installed.
validate-for-physicalpartrevision

DESCRIPTION
Validates that the submitted object is a physical part revision before traversing the as-built structure and releasing each of the physical part revisions.

SYNTAX
validate-for-physicalpartrevision

ARGUMENTS
None.

PLACEMENT
Place at the entry of the workflow to validate that the target object is a physical part revision for as-built structure traversal.

RESTRICTIONS
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager or As-Built Manager is licensed and installed.
validate-for-unserviceable-physicalpartrevision

DESCRIPTION
Checks the as-maintained structure for any unserviceable physical parts.

SYNTAX
validate-for-unserviceable-physicalpartrevision

ARGUMENTS
None.

PLACEMENT
Place at the entry of the workflow to validate that the target structure does not contain any unserviceable physical parts.

RESTRICTIONS
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager is licensed and installed.
validate-missing-asmaintained-structure

**DESCRIPTION**
Validates the as-maintained structure does not contain any missing or unidentified physical parts.

**SYNTAX**
validate-missing-asmaintained-structure

**ARGUMENTS**
None.

**PLACEMENT**
Place at the entry of the workflow to validate that the target structure does not contain any missing physical parts.

**RESTRICTIONS**
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager is licensed and installed.
validate-missing-structure

DESCRIPTION
Validates the as-built structure does not contain any missing or unidentified physical parts.

SYNTAX
validate-missing-structure

ARGUMENTS
None.

PLACEMENT
Place at the entry of the workflow to validate that the target structure does not contain any missing physical parts.

RESTRICTIONS
This handler is available only when Teamcenter maintenance, repair, and overhaul Service Manager or As-Built Manager is licensed and installed.

Handler examples

Handlers are simple programs used to perform specific actions. Though they are simple programs, very complex behavior can be generated from handlers when they are used in conjunction with each other and placed strategically on the various task actions.

Replace status of target objects workflow example

This workflow process example illustrates how to add status to objects which, for whatever reason, do not have the required status.

For example, after importing numerous objects from another system, a one-time change of status may be required so the status of the newly imported objects conform with the current system.

This workflow process applies a status of ACMERP to all target objects. If any targets have a different status, that status is replaced with ACMERP.
The **Start** node contains all the handlers for the root task. The root task contains all the other tasks within a workflow process. It is the first task to start and the last task to complete. Therefore, the handlers placed on the root task control the beginning and end of the workflow process itself, not merely the behavior of an individual task.

In this workflow example, handlers placed on the **Start** action of the root task:

- Confirm the workflow process is initiated by the correct role.
- Confirm the correct target objects are selected.
- Confirm the selected target objects are checked in.
- Automatically attach the correct target objects to the workflow.
- Attach all the components of the target assembly as targets of the workflow process.
- Configure the assembly to **Working**.
- Exclude any release objects from being attached.
- Attach all assembly components that were *not* added as targets as references.
- Attach all objects with various specified relations as targets of the workflow.

<table>
<thead>
<tr>
<th><strong>Start action</strong></th>
<th><strong>Rule handler:</strong> EPM-check-action-performer-role</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arguments:</strong></td>
<td><strong>Values:</strong></td>
</tr>
<tr>
<td></td>
<td>DBA</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Checks whether a member of the DBA or ME groups initiated the workflow. If not, the workflow does not proceed.</td>
</tr>
<tr>
<td><strong>Start action</strong></td>
<td><strong>Rule handler:</strong> EPM-validate-target-objects</td>
</tr>
</tbody>
</table>
Arguments:Values  -allowed_type:ACMEPartMfgRevision,ACMEMEProcessRevision,ACMEMEOPRevision

Description: Restrictions the types of objects that can be added as target objects to ACMEPartMfgRevision, ACMEMEProcessRevision and ACMEMEOPRevision.

<table>
<thead>
<tr>
<th>Start action</th>
<th>Rule handler: CR-assert-targets-checked-in</th>
</tr>
</thead>
</table>

Arguments:Values  No arguments set. (This handler does not accept arguments.)

Description: Confirms that all objects selected as targets of the workflow process are checked in.

<table>
<thead>
<tr>
<th>Start action</th>
<th>Action handler: EPM-attach-assembly-components</th>
</tr>
</thead>
</table>

Arguments:Values  -depth:1
   -exclude_released
   -rev_rule:Working
   -include_types:ACMETypes
   -add_excluded_as_ref

Description: Traverses one level into the assembly and attaches all the components of the target assembly as targets of the workflow process, and then configures the assembly to Working.

Excludes any release objects, collects only ACMETypes objects, and attaches all assembly components that were not added as targets as references.

<table>
<thead>
<tr>
<th>Start action</th>
<th>Action handler: EPM-attach-related-objects</th>
</tr>
</thead>
</table>

Arguments:Values  -relation:IMAN_METTarget
   -att_type:target

Description: Attaches all objects with an IMAN_METTarget relation as targets of the workflow.

<table>
<thead>
<tr>
<th>Start action</th>
<th>Action handler: EPM-attach-related-objects</th>
</tr>
</thead>
</table>

Arguments:Values  -relation:IMAN_Specification
   -att_type:target

Description: Attaches all objects with an IMAN_Specification relation as targets of the workflow.

<table>
<thead>
<tr>
<th>Start action</th>
<th>Action handler: EPM-attach-related-objects</th>
</tr>
</thead>
</table>

Arguments:Values  -relation:IMAN_Rendering
   -att_type:target

Description: Attaches all objects with an IMAN_Rendering relation as targets of the workflow.
Arguments:Values  

- relation:IMAN_Reference  
- att_type:target  

Description:  
Attaches all objects with an IMAN_Reference relation as targets of the workflow.

Start action  
Action handler: EPM-attach-related-objects

Arguments:Values  

- relation:PSBOMViewRevision  
- att_type:target  

Description:  
Attaches all objects with a PSBOMViewRevision relation as targets of the workflow.

ACMERP (Status task)

In this workflow example, handlers placed on the Start action of the ACMERP task:  

- Attach the ACMERP status to the ACMERP task.

Handlers placed on the Complete action of the ACMERP task:  

- Delete all existing statuses assigned to any target objects and replace them with the ACMERP status.

Start action  
Action handler: create-status

Arguments:Values  

ACMERP  

Description:  
Attaches the ACMERP status to the ACMERP task.

Note  
The ACMERP status should be already defined in the Business Modeler IDE.

For more information about defining status types, see the Business Modeler IDE Guide.
<table>
<thead>
<tr>
<th>Arguments:Values</th>
<th>REPLACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Deletes all existing statuses assigned to any target objects and replaces them with the <strong>ACMERP</strong> status.</td>
</tr>
</tbody>
</table>
Appendix

B  Glossary
Appendix

B  Glossary

A

access control entry (ACE)
In Access Manager, each pairing in the access control list of an accessor with the granted privileges.

access control list (ACL)
Access Manager component that contains a list of accessors and the privileges granted, denied, and not set for each accessor.

Access Manager (AM)
Teamcenter application that enables the system administrator to grant users access to Teamcenter objects.

ACE
See access control entry (ACE).

ACL
See access control list (ACL).

action handler
Handler used to extend and customize task actions. Action handlers perform such actions as displaying information, retrieving the results of previous tasks (inherit), notifying users, setting object protections, and launching applications. See also task handler.

add status task
Task template that creates and adds a release status to the target objects of the process. There is no dialog box associated with this template.

ad hoc process modification
Functionality that allows users to add tasks to, or delete tasks from, an active process.

AM
See Access Manager (AM).

approver
User who has a signoff in a workflow process regardless of role and group membership. In Access Manager, the approver accessor is used to allocate privileges that apply to all signoffs (for example, read access). See also RIG approver, role approver, and group approver.
Appendix B  Glossary

D

**Do task**
Task template that includes the **EPM-hold** handler, which stops the task from automatically completing when the task is started. This template has a customized dialog box that allows administrators to set a check box to indicate when the task is complete.

G

**group approver**
User who is a signoff in a workflow process with a specific group of users. In Access Manager, the group approver accessor is used in Workflow ACLs and matches the signoff definition (that is, group) for the release level associated with the Workflow ACL. The group approver accessor ensures that only signoffs are given privileges, not a user who matches the group. See also *approver*, *RIG approver*, and *role approver*.

P

**privileged user (workflow)**
Responsible party, process owner, or member of the system administration group. Privileged users have greater control over workflow tasks. For example, they can promote, demote, and skip workflow tasks.

**process owner**
User who initiates the workflow process; also known as the process initiator. When the process is initiated, the process owner becomes the responsible party for the process. Whenever any task in the process is not explicitly assigned to another user, person, or resource pool, the responsible party for the task defaults to the process owner.

**process template**
Blueprint of a workflow process defined by placing workflow and/or change management tasks (for example, do, perform signoff, route, and checklist) in the required order of performance. Additional process requirements, such as quorums and duration times are defined in the template using workflow handlers.

Q

**quorum**
Number of users who must vote to approve a task for that task to be approved.

R

**release status**
Status associated with a workspace object when it is released through a workflow process.

**review task**
Task template that includes the **select-signoff-team** and **perform-signoffs** subtasks. Each subtask contains a unique dialog box for executing the process.
RIG approver
User who is a signoff in a workflow process with a specified role and group. In Access Manager, the RIG approver accessor is used in Workflow ACLs and matches the signoff definition (that is, role in group) for the release level associated with the Workflow ACL. This accessor ensures that only signoffs are given privileges, not a user who matches the role in group. See also approver, group approver, and role approver.

goal approver
User who is a signoff in a workflow process with a specific role. In Access Manager, the role approver accessor is used in Workflow ACLs and matches the sign-off definition (that is, role in group) for the release level associated with the Workflow ACL. This accessor ensures that only signoffs are given privileges, not a user who matches the role. See also approver, group approver, and RIG approver.

rule handler
Handler used to integrate workflow business rules into Enterprise Process Modeling processes at the task level. Rule handlers attach conditions to an action. See also task handler.

task handler
Small Integration Toolkit program or function. Handlers are the lowest level building blocks in Enterprise Process Modeling. They are used to extend and customize tasks. There are two kinds of handlers: action handlers and rule handlers.

W

workflow
Automation of the concept that all work flows through one or more business processes to accomplish an objective. Using workflow, documents, information, and tasks are passed between participants during the completion of a particular process.

Workflow Designer
Teamcenter application that enables administrators to graphically design workflow process templates, incorporating company business practices and procedures into the templates. Teamcenter users initiate workflow processes using these templates.

Workflow Viewer
Teamcenter application that enables users to view the progress of a workflow process. Users are not required to be participating members of the process being viewed. Depending on site preference settings, Workflow Viewer also allows ad hoc process modification. See also ad hoc process modification.
Index

A
Acknowledge tasks, adding ................. 3-7
Action handlers .......................... 5-1
add-status .............................. A-14
adhoc-signoffs .......................... A-15
AI-process-export ...................... A-24
AI-process-import ...................... A-25
APPR-update-from-targets .............. A-26
approve-service-structure ............. A-27
auto-assign ............................. A-28
auto-assign-rest ....................... A-32
auto-relocate-file ..................... A-35
CAE-batch-meshing-handler .......... A-36
CAE-simulation-process-launch-handler .... A-37
change-all-started-to-pending .......... A-39
CR-assign-team-selector ............. A-40
CR-change-group-owner ............... A-43
CR-change-group-owner ............... A-44
CR-change-target-group .......... A-45
CR-fill-in-reviewers ................. A-46
CR-notify ................................ A-50
create-status .......................... A-59
debug .................................. A-60
demote .................................. A-61
demote-on-reject ...................... A-62
DOCMGT-render-document-revision .. A-64
DPV-export-device-to-ai .......... A-65
DPV-export-plant-to-ai ............. A-66
DPV-export-routine-to-ai .......... A-67
ECM-add-affected-irs-as-target .. A-68
ECM-att-new-status-for-aff-revs .. A-69
ECM-attach-components-to-change ........ A-70
ECM-copy-end-item-effectivity ...... A-71
ECM-create-base-revrule-form ...... A-72
ECM-notify-competing-changes ...... A-73
ECM-set-base-revrule ................. A-75
ECM-start-new-sub-processes ........ A-76
EPM-attach-assembly-components .. A-77
EPM-attach-item-revision-targets .. A-82
EPM-attach-related-objects .......... A-83
EPM-auto-check-in-out .............. A-91
EPM-change-ownership .............. A-92
EPM-check-signoff-comments .......... A-94
EPM-create-form ...................... A-95
EPM-create-relation .............. A-97
EPM-create-sub-process ........ A-98
EPM-delete-ugcgm-markup .... A-109
EPM-display-form .............. A-110
EPM-export-AI-AH .......... A-113
EPM-export-to-plxmlfile .......... A-114
EPM-generate-image ................ A-116
EPM-generate-ugcgm-drawing ...... A-117
EPM-make-mature-design- primary .......... A-118
EPM-mark-archive .............. A-119
EPM-perform-offline-export .... A-121
EPM-publish-target-objects ........ A-123
EPM-remove-objects .......... A-124
EPM-send-target-objects .......... A-139
EPM-send-to-oramfg .......... A-141
EPM-set-condition-by-check-validation-result ............. A-142
EPM-set-form-value-AH .......... A-143
EPM-set-job-protection .......... A-144
EPM-set-property .............. A-145
EPM-set-task-result-to-property .......... A-155
EPM-tessellation-handler .......... A-157
EPM-unpublish-target-objects .... A-159
ERP-att-logfile-as-dataset-RH .... A-160
ERP-attach-targets-AH .......... A-161
ERP-delete-log-dataset-AH .......... A-162
ERP-download-AH .......... A-163
ERP-post-upload-AH .......... A-164
ERP-set-pathnames-in-logds-AH .......... A-165
ERP-transform-AI-contents-AH .......... A-166
execute-follow-up .......... A-167
inherit ................................ A-169
invoke-system-action .......... A-170
ISSUEMG-T-check-review-decision .......... A-176
ISSUEMG-T-update-issue-status .......... A-177
late-notification .......... A-179
notify ................................ A-184
Index

notify-signoffs .................................. A-191
OBJIO-release-and-replicate .................. A-182
release-asbuilt-structure ...................... A-192
release-asmaintained-structure .............. A-193
require-authentication ......................... A-194
RM-attach-SM-tracelink requirement .......... A-195
RM-attach-tracelink requirement ............. A-196
SAP-set-valid-date-AH ......................... A-197
SAP-upload-AH .................................. A-198
schmt-approve-timesheetentries .............. A-199
schmt-revise-timesheetentries ............... A-200
set-condition .................................... A-201
set-duration ...................................... A-203
set-parent-result ................................. A-204
set-status ......................................... A-205
suspend-on-reject ................................ A-208
system ............................................. A-209
TCX-auto-approve-first-step ................... A-210
TCX-create-form .................................. A-211
TCX-Create-Print-Requests ..................... A-212
TCX-create-snapshot .............................. A-213
TCX-Create-Translation-Request .............. A-214
TCX-delete-dataset ............................... A-216
TCX-delete-log-datasets ....................... A-217
TCX-export-signoff-data ....................... A-218
TCX-IRM-cleanfields ............................. A-220
TCX-purge-dataset ................................ A-221
TCX-release-previous-itemrevs ............... A-222
TCX-remove-targets-with-status .............. A-223
TCX-set-bom-precise ............................ A-224
TCX-setstatus-EO-folder ....................... A-225
TCX-store-cr-data ................................ A-226
TCX-trigger-approve-first-step .............. A-228
trigger-action .................................... A-229
trigger-action-on-related-process-task ...... A-230
TSTK-CreateTranslationRequest ............... A-234
VAL-approve-result-overrides .................. A-235
VAL-reject-result-overrides ................... A-236
VAL-set-condition-result-overrides .......... A-237
Active processes updated in the background ........................................ 1-5
Active workflow process editing ................ 17
Ad hoc subprocesses ............................... 1-24
Add Status tasks, adding ....................... 3-40
add-status handler ................................. A-14
Adding
Acknowledge tasks ............................... 3-7
Add Status tasks .................................. 3-40
Checklist tasks ................................... 3-18
Condition tasks ................................... 3-19, 3-22
Custom tasks ...................................... 3-13
Do tasks .......................................... 3-6
generic tasks ....................................... 3-13
Impact Analysis tasks ............................ 3-14
Or tasks .......................................... 3-39
Prepare ECO tasks ................................. 3-16
Review tasks ....................................... 3-10
Route tasks ........................................ 3-12
Validate tasks ...................................... 3-27
Adding users to signoff team in ad hoc manner ........................................ A-15
adhoc-signoffs handler .......................... A-15
adhoc-signoffs legacy syntax ................... A-238
AI-process-export handler ....................... A-24
AI-process-import handler ....................... A-25
Applying status objects to target objects ........................................ A-205
APPR-update-from-targets handler ............ A-26
approve-service-structure handler ............ A-27
assert-signoffs-target-read-access handler ........................................ A-263
Assigning certain tasks to specified resource ........................................ A-40,
Assigning resources ............................... A-28
ASYNC_connection_retries preference ........ 1-6
ASYNC_connection_retry_interval preference ........................................ 1-6
async_invoker ....................................... 1-6
asyncservice service .............................. 1-5
Attaching components as targets .............. A-77
Attaching objects as attachments to process ........................................ A-83
Attaching objects as target objects .......... A-82
Attaching revisions as target object .......... A-68
Attaching status to a workflow process .................. A-14, A-155
Attaching status type to root task ............ A-59
Attributes .......................................... 5-1
auto-assign handler .............................. A-28
doi-assign-legacy legacy syntax .............. A-241
doi-assign-rest handler ......................... A-32
doi-assign-legacy syntax ......................... A-243
doi-relocate-file handler ......................... A-35
Automatic condition tasks
Creating ............................................. 3-21
Automatically assigning resources to unassigned subtasks ...................... A-32

B
Background updating of active processes ........................................ 1-5
| Browse mode                          | 13 |
| Buttons                              | 33 |

**C**

CAE-batch-meshing-handler
handler ........................................... A-36
CAE-simulation-process-launch-handler
handler ........................................... A-37
change-all-started-to-pending
handler ........................................... A-39
Changing group ownership of target
objects ............................................ A-44
Changing owner or group .................... A-45
Changing properties of item being
released ........................................... A-43
Changing status to pending .................. A-39
check-condition handler ..................... A-264
check-process-completion handler .......... A-266
check-responsible-party handler .......... A-267
check-signoff handler ......................... A-268
Checking attachments and status .......... A-289
Checking condition ............................ A-264
Checking for process completion .......... A-266
Checking for responsible party ............ A-267
Checking for signoffs ......................... A-268
Checking if components are at or above
status .............................................. A-281
Checking in/out objects
automatically ..................................... A-91
Checking item status .......................... A-270
Checking object properties .................. A-284
Checking occurrence notes ................... A-287
Checking read access .......................... A-263
Checking release status ...................... A-293
Checking status of pasted object .......... A-301
Checking target attachments and
status ............................................ A-296
Checking user against handler
criteria ............................................ A-279
Checklist tasks, adding ...................... 3-18
cleanup intervals in translator.xml file ... 1-6
Clearing signoff decisions from Review
tasks .............................................. A-61
Common keywords .............................. A-2
Condition tasks
Adding ............................................ 3-22
Condition tasks, adding ..................... 3-19
Configure applying template edits to
processes ........................................... 1-12
Configure background process updates .... 1-5
Configuring Workflow Designer ............. 7
Confirming password .......................... A-194
Controlling workflow process .............. A-313

Conventions
Syntax definitions ............................ 25
Copying effectivity ............................ A-71
Core templates ................................ A-1-25
CR-asser-targets-checked-in
handler ........................................... A-269
CR-assign-team-selector handler ........ A-40
CR-assign-team-selector legacy
syntax .............................................. A-245
CR-change-group-owner handler .......... A-43
CR-change-target-group handler .......... A-44
CR-change-target-group-owner
handler ........................................... A-45
CR-check-item-status handler .......... A-270
CR-fill-in-reviewers handler .......... A-46
CR-fill-in-reviewers legacy syntax ..... A-247
CR-notify handler ............................ A-50
CR-notify legacy syntax ..................... A-251
create-status handler ....................... A-59
Creating
Automatic condition tasks .................. 3-21
Forms .............................................. A-95
Manual condition tasks ..................... 3-20
Relations .......................................... A-97
Subprocesses .................................... A-98
Templates ......................................... 1-3
Creating subprocesses ....................... 1-15
Custom tasks, adding ....................... 3-13

**D**
debug handler ................................. A-60
debug-rule handler ............................. A-271
Debugging handlers ............................ A-13
Defining task duration ....................... A-203
Delete key in Edit more ...................... 21
Deleted links .................................... 4-2
Deleting
Datasets .......................................... A-109
Tasks .............................................. 3-42
Deleting templates ............................ 1-4
demote handler ................................ A-61
demote-on-reject handler ................... A-62
Determining ACL to use ........................ A-154,
A-157, A-167
disallow-adding-targets handler .......... A-272
disallow-removing-targets handler .... A-273
Displaying CM Base Configuration
form ............................................... A-72
Do tasks, adding ................................ 3-6
Docking the Handler dialog box .......... 21
DOCMGT-render-document-revision
handler ........................................... A-64
doing ............................................. A-92, A-110, A-124
<table>
<thead>
<tr>
<th>Index</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DPV-export-device-to-ai handler</td>
<td>A-65</td>
</tr>
<tr>
<td>DPV-export-plant-to-ai handler</td>
<td>A-66</td>
</tr>
<tr>
<td>DPV-export-routine-to-ai handler</td>
<td>A-67</td>
</tr>
<tr>
<td>Duration, setting</td>
<td>2-4</td>
</tr>
<tr>
<td>Dynamic participants</td>
<td>A-7</td>
</tr>
<tr>
<td>E</td>
<td>3-14</td>
</tr>
<tr>
<td>EC</td>
<td>A-68</td>
</tr>
<tr>
<td>ECM-add-affected-irs-as-target handler</td>
<td>A-69</td>
</tr>
<tr>
<td>ECM-att-new-status-for-aff-revs handler</td>
<td>A-70</td>
</tr>
<tr>
<td>ECM-copy-end-item-effectivity handler</td>
<td>A-71</td>
</tr>
<tr>
<td>ECM-create-base-revrule-form handler</td>
<td>A-72</td>
</tr>
<tr>
<td>ECM-is-all-affected-irs-released handler</td>
<td>A-73</td>
</tr>
<tr>
<td>ECM-is-all-affected-irs-target handler</td>
<td>A-74</td>
</tr>
<tr>
<td>ECM-is-valid-ec-process handler</td>
<td>A-75</td>
</tr>
<tr>
<td>ECM-notify-competing-changes handler</td>
<td>A-76</td>
</tr>
<tr>
<td>ECM-set-base-revrule handler</td>
<td>A-77</td>
</tr>
<tr>
<td>ECM-start-new-sub-processes handler</td>
<td>A-78</td>
</tr>
<tr>
<td>Edit active workflow processes</td>
<td>17</td>
</tr>
<tr>
<td>Edit menu</td>
<td>30</td>
</tr>
<tr>
<td>Edit mode effect on the Delete key</td>
<td>21</td>
</tr>
<tr>
<td>Editing templates</td>
<td>1-11</td>
</tr>
<tr>
<td>Editing templates online and offline</td>
<td>1-10</td>
</tr>
<tr>
<td>Editing workflow template mechanics</td>
<td>21</td>
</tr>
<tr>
<td>Editing workflow templates</td>
<td>1-9, 1-11</td>
</tr>
<tr>
<td>Edits applied to active processes</td>
<td>1-13</td>
</tr>
<tr>
<td>Enabling Workflow Designer</td>
<td>7</td>
</tr>
<tr>
<td>Engineering Change</td>
<td>3-14</td>
</tr>
<tr>
<td>Entering comments when making signoff decision</td>
<td>A-94</td>
</tr>
<tr>
<td>EPM--related-objects handler</td>
<td>A-289</td>
</tr>
<tr>
<td>EPM-assert-target-classified handler</td>
<td>A-290</td>
</tr>
<tr>
<td>EPM-attach-assembly-components handler</td>
<td>A-291</td>
</tr>
<tr>
<td>EPM-attach-item-revision-targets handler</td>
<td>A-292</td>
</tr>
<tr>
<td>EPM-attach-related-objects handler</td>
<td>A-293</td>
</tr>
<tr>
<td>EPM-auto-check-out handler</td>
<td>A-294</td>
</tr>
<tr>
<td>EPM-change-ownership handler</td>
<td>A-295</td>
</tr>
<tr>
<td>EPM-check-action-performer-role handler</td>
<td>A-296</td>
</tr>
<tr>
<td>EPM-check-assembly-status-progression handler</td>
<td>A-281</td>
</tr>
<tr>
<td>EPM-check-object-properties handler</td>
<td>A-282</td>
</tr>
<tr>
<td>EPM-check-occ-notes handler</td>
<td>A-283</td>
</tr>
<tr>
<td>EPM-check-oramfg-transfer-status handler</td>
<td>A-284</td>
</tr>
<tr>
<td>EPM-check-signoff-comments handler</td>
<td>A-285</td>
</tr>
<tr>
<td>EPM-check-status-progression handler</td>
<td>A-286</td>
</tr>
<tr>
<td>EPM-check-target-attachments handler</td>
<td>A-287</td>
</tr>
<tr>
<td>EPM-check-target-object handler</td>
<td>A-288</td>
</tr>
<tr>
<td>EPM-check-validation-result handler</td>
<td>A-289</td>
</tr>
<tr>
<td>EPM-check-validation-result-with-rules handler</td>
<td>A-290</td>
</tr>
<tr>
<td>EPM-create-form handler</td>
<td>A-291</td>
</tr>
<tr>
<td>EPM-create-relation handler</td>
<td>A-292</td>
</tr>
<tr>
<td>EPM-create-sub-process handler</td>
<td>A-293</td>
</tr>
<tr>
<td>EPM-delete-ugcgm-markup handler</td>
<td>A-294</td>
</tr>
<tr>
<td>EPM-enable_apply_template_changes preference</td>
<td>1-12</td>
</tr>
<tr>
<td>EPM-export-AI-AH handler</td>
<td>A-113</td>
</tr>
<tr>
<td>EPM-export-to-plxmlfile handler</td>
<td>A-114</td>
</tr>
<tr>
<td>EPM-generate-image handler</td>
<td>A-115</td>
</tr>
<tr>
<td>EPM-generate-ugcgm-drawing handler</td>
<td>A-116</td>
</tr>
<tr>
<td>EPM-make-maturity-make-primary handler</td>
<td>A-117</td>
</tr>
<tr>
<td>EPM-make-mature-design-primary handler</td>
<td>A-306</td>
</tr>
<tr>
<td>EPM-mark-archive handler</td>
<td>A-118</td>
</tr>
<tr>
<td>EPM-perform-offline-export handler</td>
<td>A-119</td>
</tr>
<tr>
<td>EPM-publish-target-objects handler</td>
<td>A-120</td>
</tr>
<tr>
<td>EPM-restore-objects handler</td>
<td>A-121</td>
</tr>
<tr>
<td>EPM-set-target-objects handler</td>
<td>A-122</td>
</tr>
<tr>
<td>EPM-set-task-result-to-property handler</td>
<td>A-123</td>
</tr>
<tr>
<td>EPM-unpublish-target-objects handler</td>
<td>A-124</td>
</tr>
<tr>
<td>Index-4 Workflow Designer Guide</td>
<td>PLM00037</td>
</tr>
</tbody>
</table>
Index

EPM-validate-target-objects handler .................................. A-308
ERP-att-logfile-as-dataset-RH handler .................................. A-160
ERP-attach-targets-AH handler .................................. A-161
ERP-check-effective-date-RH handler .................................. A-310
ERP-check-target-status-RH handler .................................. A-311
ERP-delete-log-dataset-AH handler .................................. A-162
ERP-download-AH handler .................................. A-163
ERP-post-upload-AH handler .................................. A-164
ERP-set-pathnames-in-logds-AH handler .................................. A-165
ERP-transform-AI-contents-AH handler .................................. A-166
ERP-validate-data-RH handler .................................. A-312
Error codes .................................. 3-25
Errors in workflow .................................. 19
Examples of handlers in workflows .................................. A-341
execute-follow-up handler .................................. A-167
Executing
Arguments .................................. A-209
External command .................................. A-170
Export data to AI object .................................. A-113
Export data to XML file .................................. A-114

F
Failure path .................................. 4-3
File menu .................................. 28

G
Generating
Archiving requests .................................. A-119
CGM images .................................. A-117
NX images .................................. A-116
Generic tasks, adding .................................. 3-13
Go menu .................................. 32

H
Handler dialog box docked versus undocked .................................. 21
Handler example 1
ACMERP (Status task) .................................. A-344
Start node .................................. A-341
Handler examples .................................. A-341
Handler keywords .................................. A-1
Handlers .................................. 5-1
Action .................................. 5-1
Rule .................................. 5-1
Impact Analysis tasks, adding .................................. 3-14
In-process workflow process editing .................................. 17
inherit handler .................................. A-169
Inheriting attachment types .................................. A-169
Interface .................................. 27
invoke-system-action handler .................................. A-170
invoke-system-rule handler .................................. A-313
isactive attribute in translator.xml file .................................. 1-6
ISSUEMGT-check-review-decision handler .................................. A-176
ISSUEMGT-update-issue-status handler .................................. A-177

K
Keywords .................................. A-1–A-3
Common .................................. A-2
Handler .................................. A-1

L
late-notification handler .................................. A-179
Legacy syntax for action handlers
adhoc-signoffs .................................. A-238
auto-assign .................................. A-241
auto-assign-rest .................................. A-243
CR-assign-team-selector .................................. A-245
CR-fill-in-reviewers .................................. A-247
CR-notify .................................. A-251
notify .................................. A-257
Links
Deleted .................................. 4-2
Fail .................................. 4-3
Links, deleting .................................. 4-2

M
Manual condition tasks
Creating .................................. 3-20
maxlimit attribute in translator.xml file .................................. 1-6
Menus
Edit .................................. 30
File .................................. 28
Go .................................. 32
Tools .................................. 32
View .................................. 32
Multi-Site Collaboration .................................. 6-1–6-2

N
notify handler .................................. A-184
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify legacy syntax ..................................................................</td>
</tr>
<tr>
<td>notify-signoffs handler ................................................................</td>
</tr>
<tr>
<td>Notifying about competing changes .............................................</td>
</tr>
<tr>
<td>Notifying of task status ............................................................</td>
</tr>
<tr>
<td>Notifying of task status through e-mail ......................................</td>
</tr>
<tr>
<td>Notifying Update Manager of release status ...................................</td>
</tr>
<tr>
<td>Notifying user of current action ................................................</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>OBJIO-release-and-replicate handler ...........................................</td>
</tr>
<tr>
<td>Offline template editing ................................................................</td>
</tr>
<tr>
<td>Online template editing ..................................................................</td>
</tr>
<tr>
<td>Or tasks, adding ............................................................................</td>
</tr>
<tr>
<td>origin_uid ....................................................................................</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>Parent tasks, viewing .....................................................................</td>
</tr>
<tr>
<td>Pausing the task ...........................................................................</td>
</tr>
<tr>
<td>Perspectives</td>
</tr>
<tr>
<td>In general ....................................................................................</td>
</tr>
<tr>
<td>polling interval in translator.xml file .......................................</td>
</tr>
<tr>
<td>Prepare ECO tasks, adding ............................................................</td>
</tr>
<tr>
<td>Prerequisites for Workflow Designer .............................................</td>
</tr>
<tr>
<td>Preventing deletion when process complete ....................................</td>
</tr>
<tr>
<td>Preventing target removal ............................................................</td>
</tr>
<tr>
<td>Printing information about last action .........................................</td>
</tr>
<tr>
<td>Process template ............................................................................</td>
</tr>
<tr>
<td>Publishing objects in ODS database ..............................................</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>Recipients list, setting ................................................................</td>
</tr>
<tr>
<td>release-asbuilt-structure handler ...............................................</td>
</tr>
<tr>
<td>release-asmaintained-structure handler .......................................</td>
</tr>
<tr>
<td>Releasing ......................................................................................</td>
</tr>
<tr>
<td>Relocating datasets ........................................................................</td>
</tr>
<tr>
<td>Removing objects from ODS ............................................................</td>
</tr>
<tr>
<td>Rendering a document ......................................................................</td>
</tr>
<tr>
<td>Replicating workflow templates ...................................................</td>
</tr>
<tr>
<td>require-authentication handler ....................................................</td>
</tr>
<tr>
<td>Restricting use of objects as target objects ..................................</td>
</tr>
<tr>
<td>Review tasks, adding ......................................................................</td>
</tr>
<tr>
<td>Rich client perspectives and views ................................................</td>
</tr>
<tr>
<td>Rich client views ..........................................................................</td>
</tr>
<tr>
<td>RM-attach-SM-tracelink-requirement .............................................</td>
</tr>
<tr>
<td>RM-attach-tracelink-requirement ................................................</td>
</tr>
<tr>
<td>Root tasks, viewing .......................................................................</td>
</tr>
<tr>
<td>Route tasks, adding ......................................................................</td>
</tr>
<tr>
<td>Rule handlers</td>
</tr>
<tr>
<td>assert-signoffs-target-read-access .............................................</td>
</tr>
<tr>
<td>check-condition ............................................................................</td>
</tr>
<tr>
<td>check-process-completion ............................................................</td>
</tr>
<tr>
<td>check-responsible-party ..................................................................</td>
</tr>
<tr>
<td>check-signoff ...............................................................................</td>
</tr>
<tr>
<td>CR-assert-targets-checked-in ......................................................</td>
</tr>
<tr>
<td>CR-check-item-status .....................................................................</td>
</tr>
<tr>
<td>debug-rule .....................................................................................</td>
</tr>
<tr>
<td>disallow-adding-targets ...............................................................</td>
</tr>
<tr>
<td>disallow-removing-targets ...........................................................</td>
</tr>
<tr>
<td>ECM-check-target-is-ec ..................................................................</td>
</tr>
<tr>
<td>ECM-is-all-affected-irs-released ................................................</td>
</tr>
<tr>
<td>ECM-is-all-affected-irs-target ....................................................</td>
</tr>
<tr>
<td>ECM-is-valid-ec-process ...............................................................</td>
</tr>
<tr>
<td>EPM-assert-target-classified .......................................................</td>
</tr>
<tr>
<td>EPM-check-action-performer-role ..................................................</td>
</tr>
<tr>
<td>EPM-check-assembly-status-progression .......................................</td>
</tr>
<tr>
<td>EPM-check-object-properties .......................................................</td>
</tr>
<tr>
<td>EPM-check-oce-notes ......................................................................</td>
</tr>
<tr>
<td>EPM-check-oramfg-transfer-status .................................................</td>
</tr>
<tr>
<td>EPM-check-related-objects ...........................................................</td>
</tr>
<tr>
<td>EPM-check-status-progression .......................................................</td>
</tr>
<tr>
<td>EPM-check-target-attachments .....................................................</td>
</tr>
<tr>
<td>EPM-check-target-object ...............................................................</td>
</tr>
<tr>
<td>EPM-check-validation-result ........................................................</td>
</tr>
<tr>
<td>EPM-check-validation-result-with-rules ........................................</td>
</tr>
<tr>
<td>EPM-check-error ............................................................................</td>
</tr>
<tr>
<td>EPM-validate-target-objects .........................................................</td>
</tr>
<tr>
<td>ERP-check-effective-date-RH ........................................................</td>
</tr>
<tr>
<td>ERP-check-target-status-RH ..........................................................</td>
</tr>
<tr>
<td>ERP-validate-data-RH ....................................................................</td>
</tr>
<tr>
<td>invoke-system-rule ........................................................................</td>
</tr>
<tr>
<td>SAP-check-forms-attached-RH ......................................................</td>
</tr>
<tr>
<td>SAP-check-forms-to-download-RH ..................................................</td>
</tr>
<tr>
<td>TCX-check-approver .......................................................................</td>
</tr>
<tr>
<td>TCX-check-bom-precise .................................................................</td>
</tr>
<tr>
<td>TCX-check-bomchild-status-list ....................................................</td>
</tr>
<tr>
<td>TCX-check-comps-against-pattern .................................................</td>
</tr>
<tr>
<td>TCX-check-datasets ....................................................................</td>
</tr>
<tr>
<td>TCX-check-itemrev-status .............................................................</td>
</tr>
<tr>
<td>TCX-check-jobowner .....................................................................</td>
</tr>
</tbody>
</table>
TCX-check-prev-itemrev-status . . . A-330
TCX-check-signoff . . . . . . . . . A-331
TCX-check-status . . . . . . . . . A-332
TCX-has-target-drawing . . . . . A-333
validate-for-checkedout-asmaintained-
physicalpartrevision . . . . . . . A-334
validate-for-checkedout-
physicalpartrevision . . . . . . . A-335
validate-for-class . . . . . . . . . A-336
validate-for-latest-
asphysicalpartrevision . . . . . A-337
validate-for-physicalpartrevision . . A-338
validate-for-unserviceable-
physicalpartrevision . . . . . . . A-339
validate-missing-asmaintained-
structure . . . . . . . . . . . . . . . A-340
validate-missing-structure . . . . . A-341
Running workflow process editing . . . 17

S
Sample workflows
Replace status of target objects . . . A-341
SAP-check-forms-attached-RH
handler . . . . . . . . . . . . . . A-319
SAP-check-forms-to-download-RH
handler . . . . . . . . . . . . . . A-321
SAP-set-valid-date-AH handler . . . A-197
SAP-upload-AH handler . . . . . . . A-198
schmgt-approve-timesheetentries
handler . . . . . . . . . . . . . . A-199
schmgt-revise-timesheetentries
handler . . . . . . . . . . . . . . A-200
Security Services for background
processing . . . . . . . . . . . . 1-7
Selecting a template . . . . . . . . 2-1
Sending CR notification . . . . . . . A-50
Sending objects to sites . . . . . . . A-139
Set path names of files . . . . . . . A-165
set-condition handler . . . . . . . A-201
set-duration handler . . . . . . . A-203
set-parent-result handler . . . . . A-204
set-status handler . . . . . . . . A-205
Setting condition of parent task . . . A-204
Setting condition tasks . . . . . . . A-201
Setting up late notifications . . . . A-179
SOA URL for background processing . 1-7
Specifying objects to exclude . . . A-272
Start node
description . . . . . . . . . . . . . 1-2
Handler example 1 . . . . . . . . A-341
handlers . . . . . . . . . . . . . . 1-2
Starting Workflow Designer . . . . . 7
Subprocesses . . . . . . . . . . . . 1-15

From a workflow template . . . . . . 1-16
From an in-process task . . . . . . 1-24
Subprocesses, viewing . . . . . . . 2-3
suspend-on-reject handler . . . . . A-208
Suspending task when no quorum . . A-208
Synchronizing workflow templates . . 6-2
Syntax definition conventions . . . . 25
system handler . . . . . . . . . A-209

T
Task attributes, editing . . . . . . 5-2
Task attributes, viewing . . . . . . 2-3
Task handlers . . . . . . . . . . . 5-4
Creating . . . . . . . . . . . . . 5-5–5-6
Deleting . . . . . . . . . . . . . 5-7
Editing . . . . . . . . . . . . . . 5-7
Viewing . . . . . . . . . . . . . 5-4
Task handlers, viewing . . . . . . 2-5
Task template . . . . . . . . . . . 15
Tasks . . . . . . . . . . . . . . . . . . 11
Attributes . . . . . . . . . . . . 5-1
Deleting . . . . . . . . . . . . . 3-42
Linking manually . . . . . . . . . 4-2
TCX-auto-approve-first-step
handler . . . . . . . . . . . . . A-210
TCX-check-approver handler . . . . A-322
TCX-check-bom-precise handler . . A-323
TCX-check-bomchild-statuslist
handler . . . . . . . . . . . . . A-324
TCX-check-comps-against-pattern
handler . . . . . . . . . . . . . A-325
TCX-check-datasets handler . . . . A-326
TCX-check-itemrev-status handler . A-328
TCX-check-jobowner handler . . . . A-329
TCX-check-prev-itemrev-status
handler . . . . . . . . . . . . . A-330
TCX-check-signoff handler . . . . A-331
TCX-check-status handler . . . . A-332
TCX-create-form handler . . . . . A-211
TCX-Create-Print-Requests
handler . . . . . . . . . . . . . A-212
TCX-create-snapshot handler . . . A-213
TCX-Create-Translation-Request
handler . . . . . . . . . . . . . A-214
TCX-delete-dataset handler . . . . A-216
TCX-delete-log-datasets handler . . A-217
TCX-export-signoff-data handler . A-218
TCX-has-target-drawing handler . A-333
TCX-IRM-cleanfields handler . . . A-220
TCX-purge-dataset handler . . . . A-221
TCX-release-previous-itemrevs
handler . . . . . . . . . . . . . A-222
Index

TCX-remove-targets-with-status handler ......................... A-223
TCX-set-bom-precise handler ......................... A-224
TCX-setstatus-EO-folder handler ......................... A-225
TCX-store-cr-data handler ......................... A-226
TCX-trigger-approve-first-step handler ......................... A-228
Teamcenter perspectives and views ......................... 39
Template editing mechanics ......................... 21
Template edits ......................... 1-9, 1-11
Template edits applied to active processes ......................... 1-13
Template edits applied to processes configuration ......................... 1-12
Templates
Core ......................... 1-25
Creating ......................... 1-3
Definition ......................... 1-2
Deleting ......................... 1-4
Editing ......................... 1-11
Selecting ......................... 2-1
Viewing ......................... 2-2
Workflow process ......................... 13
Workflow task ......................... 15
Templates edited online and offline ......................... 1-10
Tools menu ......................... 32
Transforms AI objects ......................... A-166
translator.xml file ......................... 1-5
trigger-action handler ......................... A-229
trigger-action-on-related-process-task handler ......................... A-230
Triggering action on task ......................... A-229
Triggering action on task in related process ......................... A-230
TSTK-CreateTranslationRequest ......................... A-234

U
Undocking the Handler dialog box ......................... 21

V
VAL-approve-result-overrides ......................... A-235
VAL-reject-result-overrides ......................... A-236
VAL-set-condition-result-overrides ......................... A-237
Validate tasks, adding ......................... 3-27
validate-for-checkedout-asmmaintained-
physicalpartrevision handler ......................... A-334
validate-for-checkedout-physicalpartrevision handler ......................... A-335
validate-for-class handler ......................... A-336
validate-for-latest-asmpartrevision handler ......................... A-337
validate-for-physicalpartrevision handler ......................... A-338
validate-for-unserviceable-
physicalpartrevision handler ......................... A-339
validate-missing-asmmaintained-structure handler ......................... A-340
validate-missing-structure handler ......................... A-341
Validates As-Built structures ......................... A-335
Validates As-Maintained structures ......................... A-334
Verifying addition of all EC revisions ......................... A-276
Verifying addition of at least one EC revision ......................... A-274
Verifying check-in status ......................... A-269
Verifying if objects are classified ......................... A-278
Verifying template validity ......................... A-277
Verifying that revisions were released ......................... A-275
View menu ......................... 32
Viewing
  Parent tasks ......................... 2-2
  Root tasks ......................... 2-3
  Subprocesses ......................... 2-3
  Subtasks ......................... 2-2
  Task attributes ......................... 2-3
  Task handlers ......................... 2-5
Workflow process templates ......................... 2-2
Views
  Rich client ......................... 39

W
Workflow Designer
  Interface ......................... 27
  Refreshing ......................... 4-1
  Tasks ......................... 11
Workflow errors ......................... 19
Workflow process template ......................... 13
Workflow subprocesses ......................... 1-15
Workflow task template ......................... 15
Workflow template editing ......................... 21, 1-11
Workflow template edits ......................... 1-9
Workflow template edits applied to active processes ......................... 1-13
Workflow templates
  Exporting ......................... 6-5
  Importing ......................... 6-3
  Multiple sites ......................... 6-1
  Replicating ......................... 6-1
  Synchronizing ......................... 6-2
Workflow templates and the Delete key ......................... 21