SIEMENS

Teamcenter Rapid Start 11.6

Setting Up Workflows for Product Development

Contents

Chapter 1: What is Workflow Designer?

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. 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.

Before you begin

Prerequisites	Using the Workflow Designer application in **Edit** mode requires Teamcenter administrator privileges.
Enable Workflow Designer	To enable the Workflow Designer feature, select it during installation.
	If you have trouble accessing Workflow Designer, see your system administrator.

> **Note**
>
> You can log on to Teamcenter only once. If you try to log on to multiple workstations, an error message appears.

Configure Workflow
Designer

You can accept Workflow Designer's default configuration settings, or modify
them using workflow preferences.
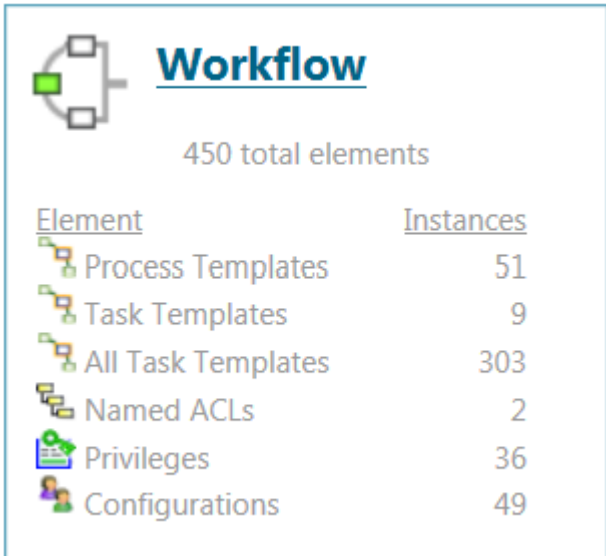
> **Note**
>
> Either the **Process view** or **Task view** can be set as the default in
> the **TCVIEWER_default_workflow_view** preference.

Start Workflow
Designer

In the navigation pane, click **Workflow Designer** .

> **Note**
>
> The Administration Data Report site provides a list of default administration values.
>
> Select the Workflow tile to expand the view of all default elements and values.

> Select the Preferences tile for information on preferences and default values in Configuring
> Teamcenter > Managing Preferences.

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
```

What is a workflow?

Introduction to 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. Typically, documents, information, or tasks are passed from one participant to another in a way that is governed by rules or procedures.

A *workflow process* is initiated by a user, and workflow tasks are assigned to users.

As shown in the following diagram, in a basic workflow the initial **Start** step leads to the active **Do** task, **Create Design**. The **Do** task leads to a pending **Review** task, **Design Signoff**, and then to the final **Finish** step.
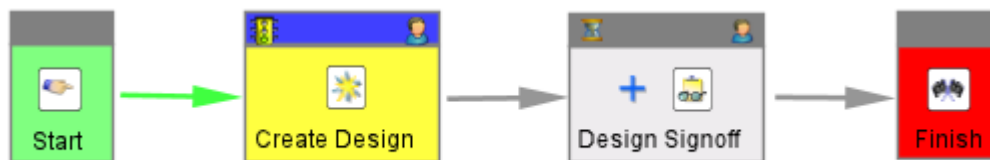
Workflow benefits

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.

Workflow examples

For example, you can create a simple review workflow in which an object is reviewed. Depending on the outcome of the review, one of two tasks is then required. When either of the tasks is performed, the workflow is complete. At completion, the object is granted a specified status.

Typically, an object sent through a review workflow is granted *Released* status after successful completion. Standard workflow behavior for released objects are that their release time and date is marked and the object is made read-only.

In this example, if an item revision containing a design part and its accompanying documentation is sent through design review, and the **Post Approval** task completes (rather than the **Post Rejection** task), the item revision part is marked as **Released** when the workflow finishes. The item revision and the objects it contains (the design part, and the documentation) are made read-only. No further changes can be made to the design, enforcing the review that was just performed.
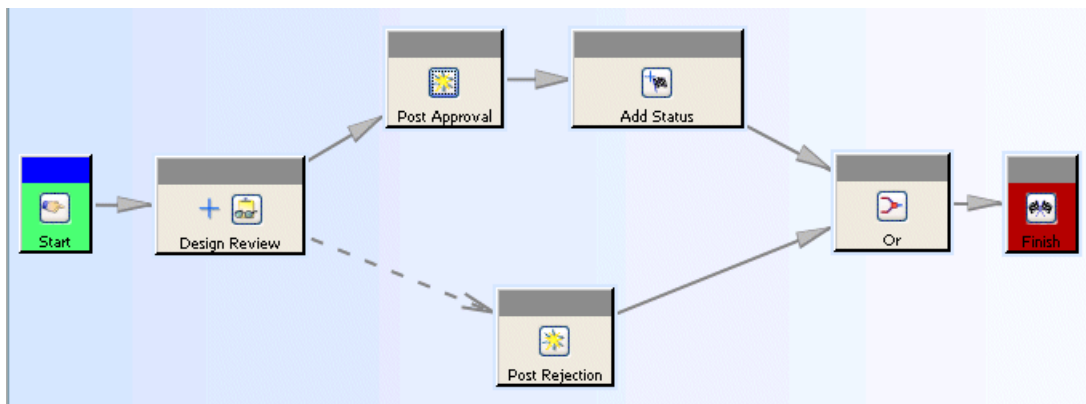
In another example, you can create a more complicated workflow containing a **Condition** task. In this workflow, whether a specified condition is met or not determines the second round of tasks. Which tasks are required depend on whether the condition was met.
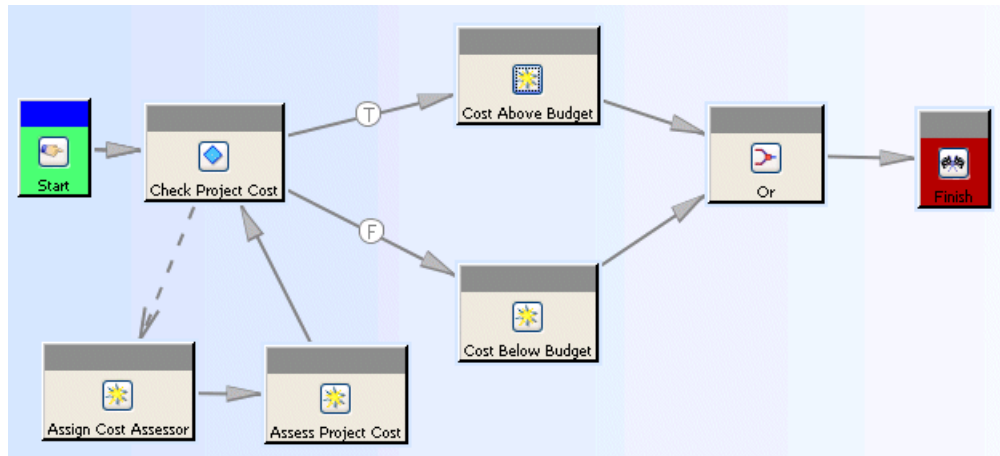
Teamcenter workflows are extensible by handlers; small ITK programs used to extend and customize the tasks. are essential to the creation of highly functional, flexible workflows.

*   Action handlers perform an action, such as attaching objects or sending an e-mail.

*   Rule handlers confirm a defined rule has been satisfied.

Using workflows

You can use workflows in Teamcenter to manage your processes and changes in many applications, such as:

*   Change Manager

    Workflows are ideal for managing your change process as problem reports lead to change requests which lead to change notices. With a well-designed change process and matching workflow process template, you can ensure that the right people perform the correct tasks in the proper order.

*   Systems Engineering

    A typical Systems Engineering workflow is the requirements, functional, logical, and physical design (RFLP) process. The process is iterative and may be repeated during the design or development of a product.
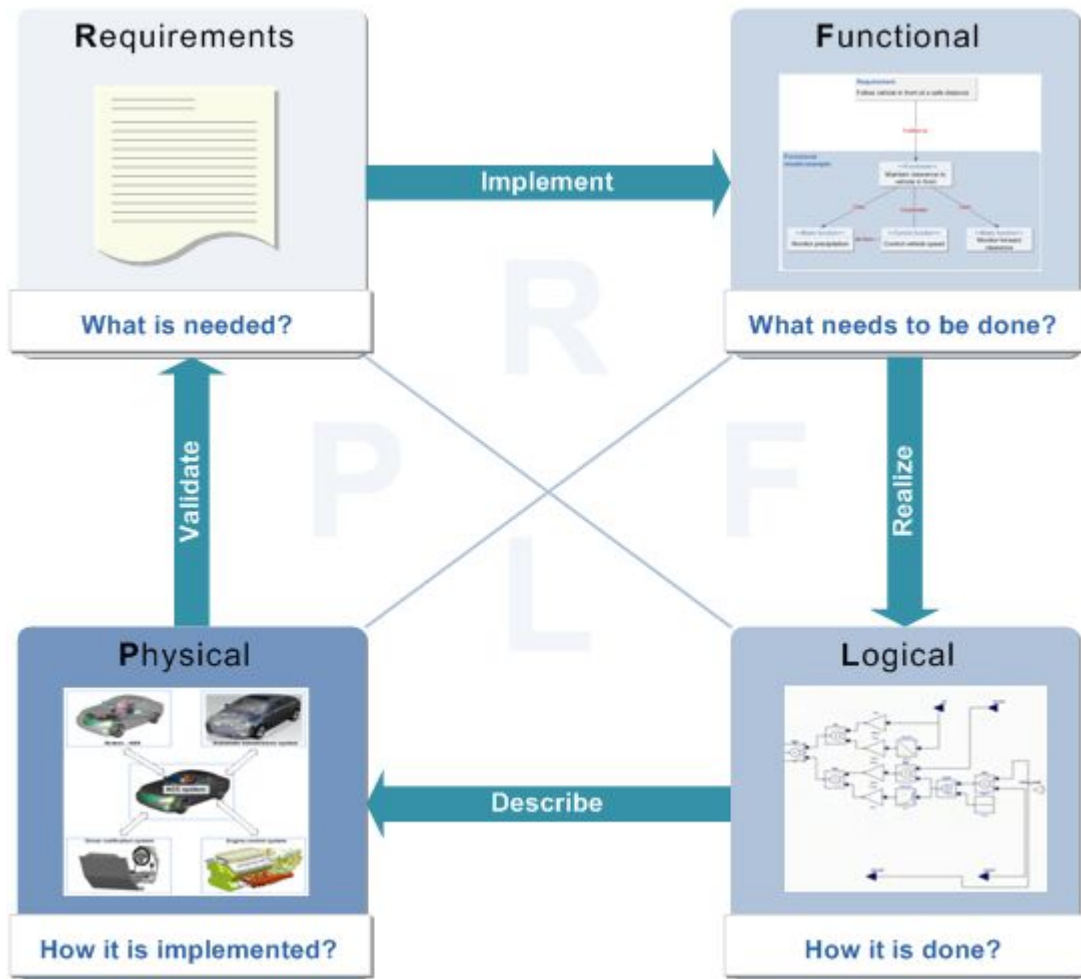
You can construct a workflow process template that matches your organization's version of the RFLP process.

> **Note**
>
> For ease of use, Siemens PLM Software recommends using My Teamcenter to initiate and complete workflow processes because the entire procedure can be accomplished from within your inbox in **My Worklist**. You can also initiate workflows from the Workflow Viewer application.

Workflow elements

Workflows pass documents, information, and tasks between participants during the completion of a particular process. A workflow process can be large and complicated or simple and straightforward.

> **Note**
>
> Certain privileged users can perform administrative actions in a process, such as removing a user who is no longer with the company. A privileged user may be the responsible party, the process owner, or a member of a system administration group. You may be a privileged user in certain processes but only be able to perform standard user actions in other processes.

Workflow element	Description
Workflow template	Blueprints of workflow processes. Your administrator creates process templates. A specific process is defined by placing tasks in the template in the required order of performance. Additional requirements, such as quorums and duration times, may also be included in the template. For more information about creating and managing templates, see *Workflow Designer*.
Container tasks	Tasks that contain other include tasks: • **Review** Contains **select-signoff-team** and **perform-signoffs** tasks. The **Decision** options are **Approve**, **Reject**, and **No Decision**. • **Acknowledge** Contains **select-signoff-team** and **perform-signoffs** tasks. The **Decision** options are **Acknowledged** and **Not Acknowledged**. • **Route** Contains **Review**, **Acknowledge** and **Notify** tasks.
Interactive tasks	Tasks that require user interaction display in the affected user's worklists. Different types of tasks have different interactive requirements. Typical tasks include: • **select-signoff-team** The assigned user is required to select a signoff team to sign off the target object of the task. • **perform-signoffs** Assigned users are required to review and sign off the target object of the task. • **Do** The assigned user is required to review and perform the task instructions, then mark the task complete. • **Notify**

Workflow element	Description
	The assigned user is required to reply.
Process tasks	Tasks that perform noninteractive functions, such as branching the workflow, specifying query criteria, and error handling. When you view a workflow using the **Process View**, these tasks are displayed. These tasks require no user interaction, so they do not appear in user worklists.
Parent processes	Workflow processes can contain child workflow processes. In these situations, the initial workflow process is the parent workflow process, and it contains a subprocess. Parent workflow processes are dependent upon subprocesses; they cannot complete until the subprocess completes.
Workflow handlers	Small ITK programs used to extend and customize workflow tasks. Action handlers perform actions, such as attaching objects, sending email, or determining whether a rule has been satisfied.
Task attributes	Attributes that further configure task behavior. You can set security attributes, customize task symbols, and define condition results.
Quorum requirements	Values that specify the number of approvals required before **perform-signoffs** tasks can complete and workflows can proceed.

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.

Task	Definition
Do 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.
	Uses the **EPM-hold** handler, which stops the task from automatically completing when started.
Acknowledge Task	Uses the **Acknowledged** and **Not Acknowledged** subtasks, each of which has its own dialog box.
Review Task	Uses the **select-signoff-team** and **perform-signoffs** subtasks, each of which has its own dialog box.
	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.
Route Task	Uses the **Review**, **Acknowledge**, and **Notify** subtasks, each of which has its own dialog box.
Task	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 **EPMTask** template.
Condition Task	Branches a workflow according to defined query criteria. Requires that the succeeding task contains an **EPM-check-condition** handler that accepts a Boolean value of either **True** or **False**.
Validate Task	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.
Add Status Task	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.
Or 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.

Workflow privileged user

System administrators can create access rules and assign access privileges for workflow tasks.

- Access privileges are required to permit a workflow user to perform certain workflow tasks:

  o Removing a user from an active workflow.

  o Promoting or demoting a task in an active workflow.

- Workflow task permissions are:

  o   Specific to the workflow process template.

  o   Granted to a user by an Access Manager ACL on the workflow task, or by the rule tree.

  > **Note**
  >
  > Configure the **WRKFLW_modify_completed_workflow** preference to **true** to allow users to modify tasks from completed or aborted workflows.
  >
  > To specify the access privileges needed to modify workflow targets configure the **WRKFLW_modify_target_list_access_privilege** preference

- Typically, the named-ACL used to grant permissions to promote or demote a task is the **EPM-set-rule-based-protection** handler.

  > **Note**
  >
  > **WRKFLW_skip_root_task_from_acl_evaluation** controls whether the access management rights evaluation will include the workflow acccessors related to the Root Task or not.

For more information about setting permissions, see *Acess Manager*.

For more information about the **EPM-set-rule-based-protection** workflow handler, see *Workflow Designer*.

Workflow Designer interface

Workflow Designer view

Workflow Designer uses the standard Teamcenter rich client interface.
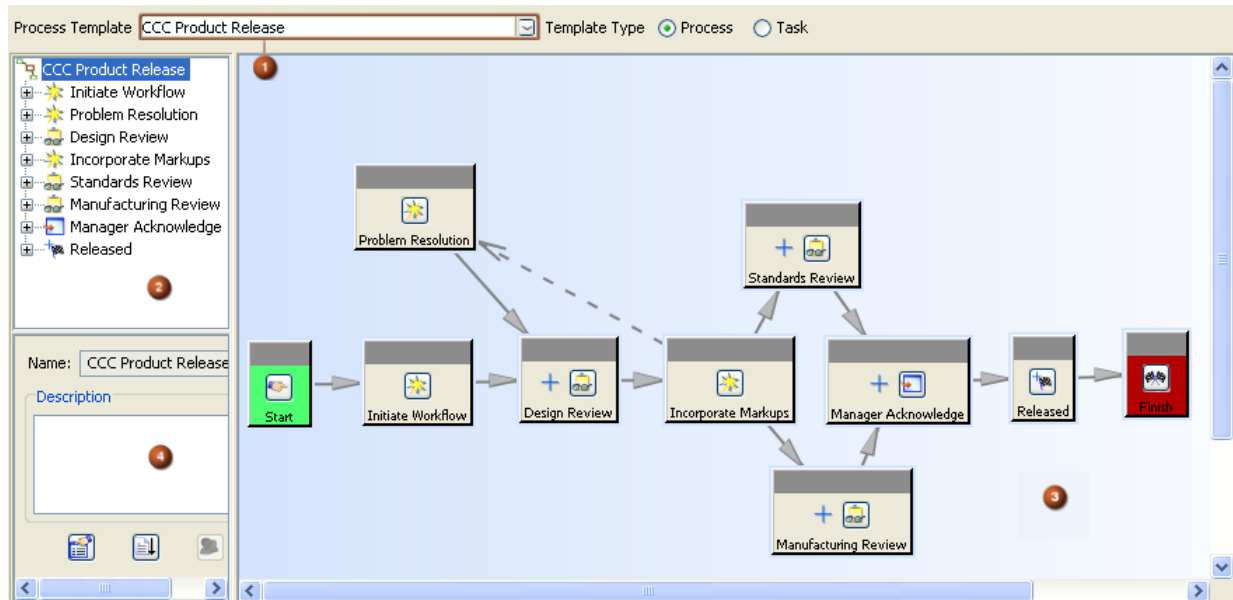
1	**Process Template** box	Lists either all process or all task templates, depending on whether **Process** or **Task** is selected for the **Template Type**.
2	Task hierarchy tree	Displays hierarchical tasks of the template shown in the **process template** box. The tree shows the relationship of all tasks in a process template or subtasks in a task template.

> **Note**
>
> The hierarchy tree does not indicate the task execution order.

3	Process flow pane	Displays a sequential, 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. Elements displayed in the window are dependent on the status and configuration of the selected template.
		If a template stage is set to **Under Construction**, the template is visible only to users with administrative privileges. **Under Construction** templates have the **Set Stage to Available** check box. 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.

Command	Description
New Root Template	Allows you to create a new workflow process and task templates.

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

Element	Description
New Root Template Name	Type a name for the new template. The default name is **New Process #**, where **#** is the next number available to make the template name unique.
Based On Root Template	Choose a template from the list. The default choice is **Empty Template**, which provides a blank template on which to build. **Core templates** 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.
Template Type	Choose the type of template to create: • **Process template** 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. • **Task template** Contains only a single task.
Task hierarchy tree	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. When creating a template, you can view, but you cannot modify, the task hierarchy.
Name	Lists the name of the selected template. When creating a template, you can view, but you cannot modify, the **Name** box of the selected template.
Description	Lists descriptive notes added by users. When creating a template, you can view, but you cannot modify, the **Description** box.

Element	Description
Task Attributes button	Click to view the task attributes for the selected template.
	When creating a template, you can view, but you cannot modify, the task attributes.
Task Handlers button	Click to view the task handlers for the selected template.
	When creating a template, you can view, but you cannot modify, the task handlers.
Task Signoff button	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.
Process flow pane	Shows the task flow of the selected template.
	When creating a template, you can view, but you cannot modify, the tasks.
OK button	Click to finish creating the new template and close the dialog box.
Apply button	Click to finish creating the new template. The dialog box remains open, allowing you to create additional templates.
Cancel button	Click to cancel the operation.

Edit menu

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

Command	Description
Template	Lists the task templates available in Teamcenter.
Task	Workflow Designer default template setting. The **Task** template is synonymous with the **EPMTask** template.
Do 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.
	Uses the **EPM-hold** handler, which stops the task from automatically completing when started.

Command	Description
Review Task	Uses the **select-signoff-team** and **perform-signoffs** subtasks, each of which has its own pane in the **Viewer** view. **Wait for Undecided Reviewers** is an option to set the **Review** task to wait for all reviewers to submit their decisions before completing and following the appropriate path. **Tip** A Teamcenter administrator can customize the **perform-signoffs** pane to add boxes and buttons, validate users' input in the new boxes, configure the summary table, and configure the **Signoff Decision** dialog box.
Add Status 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. **Note** The **WRKFLW_retain_ACL_objects_on_release** preference determines the state of access control list objects on a target when a release status is applied. Valid values are **true** or **false**.
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 an **EPM-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 give you the ability to respond to errors by providing an alternate path which the workflow process traverses when an error occurs.

Command	Description
Template Filter	Associates a list of workflow process templates with a designated target object type and user group. You can apply the list to only one type and group at a time. Subtypes and subgroups do not inherit this association.

> **Caution**
>
> This feature is deprecated as of Teamcenter 11.2. Siemens PLM Software recommends that you associate templates by using Business Modeler IDE conditions. Conditions offer greater versatility, with criteria such as session group, role, and user; target project and target release status; and custom criteria (both session-specific and target-specific) that a Teamcenter administrator can create.

Command	Description
Mode	Lists the two working modes: **Edit** and **Browse**.
Browse	Allows you to view the workflow process data and inspect the details of the workflow process. You cannot make any modifications in this mode. Browse mode is the default mode.
Edit	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.

> **Note**
>
> Access may be restricted even if you have administrator privileges.

View menu

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

Command	Description
Task Properties	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.

Tools menu

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

Command	Description
Export	Exports a workflow template to a file.
Import	Imports a workflow template from a file.
Purge Templates	Deletes old workflow templates.

Go menu

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

Command	Description
Up a Level	Opens the parent task of the currently selected task from the task hierarchy tree.
Down a Level	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.
Top Level	Opens the root task of the workflow process.

Workflow Designer buttons

Button	Description
Task Properties	Displays the name, description, attributes, and handlers of the selected task.
Task Attributes	Displays and opens for edit the named ACL, task type, and quorum requirements for the selected task.
Task Handlers	Displays and opens for edit task handlers for the selected task.
Task Signoffs	Displays and opens for edit the group, role, quorum, and number of reviewer requirements for the selected task.
Task	Inserts an empty task with no handlers into the workflow template for you to customize.
Do Task	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.
	This task uses the **EPM-hold** handler, which stops the task from automatically completing once started.
Review Task	Inserts a **Review** task into the workflow template. This task uses the **select-signoff-team** and **perform-signoffs** subtasks, each of which has its own dialog box.
	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.
Add Status Task	Inserts an **Add Status** 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.

Button	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 template. 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 template. This task requires that the succeeding task contains an **EPM-check-condition** handler that accepts a Boolean value of either **True** or **False**.
Route Task	Inserts a **Route** task into the workflow template. 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 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.
Up a Task Level	Displays the task one level higher than the current task.
Down a Task Level	Displays the task one level lower than the current task.

Workflow Designer panes

Task attributes

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

Element	Description
Named ACL	Click to display the **Named ACL** dialog box.
Task Type	Lists the type of task template assigned to the selected task.
Icons	Displays the symbol that has been assigned to the selected task. You can also add custom symbols to this list.
Condition Query	Displays when a **Condition** 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 **Condition Query** dialog box, which you can use to change, modify, or delete the defined query. |

Element	Description
Duration	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 **Set** to display the **Set Duration** 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.
Recipients	Displays the names of users selected to receive program mail when the selected task becomes overdue.
	Click **Set** to display the **Select Recipients** dialog box, which you can use to select users who will receive program mail if the selected task becomes overdue.
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.
Process in Background	Indicates if the task is to be run in the background.

Task handlers

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

Element	Description
Task action tree	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.
Move Handler Up ▲	Moves the selected handler up within a folder.
Move Handler Down ▼	Moves the selected handler down within a folder.
Expand All Folders	Expands all folders.
Collapse All Folders	Collapses all folders.
Handler Type	Indicates an action handler or rule handler.

Element	Description
Quorum	In **Browse** mode, when a predefined rule handler is selected, displays an integer representing the number required for the approval quorum. In **Edit** mode, you can type or modify the approval quorum number, but only when a rule handler is selected as the **Handler Type**.
Task Action	The selected task action from the list receives a handler when it is created.
Action/Rule Handler	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.
Argument	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.
Value(s)	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.
Create	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.
Delete	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.
Modify	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.
Help	Selecting a handler from the **Handler** box and clicking **Help** displays the documentation for the selected handler.
Performs a Copy action	Places the selected handler in the clipboard.

Element	Description
Performs a Paste action 📋	In **Edit** ✖️ mode, places the copied handler in the selected location.
	• To paste on another action in the task, select the target action in the task action tree.
	• To paste on another task in the same template, select the target task in the task hierarchy tree.
	• To paste on a task in another template, select the target template from the **Process Template** list.

Task signoffs

The following table lists the elements available in the **Signoff Profile** pane.

Element	Description
Signoff Profiles	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.
Group	Lists the user responsible for the task.
Role	Lists the roles responsible for the task.
Number of Reviewers	Click the menu to select an to be associated with the selected task.
Allow sub-group members	Grants members of subgroups permission to sign off instead of members of the designated group.
Signoffs Quorum	**Numeric**: Select numeric and type a whole number or **ALL**.
	Percentage: Enter a percentage.
	Wait for Undecided Reviewers: 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.
Create	This button is available only when Workflow Designer is in **Edit** mode.
	Click **Create** to create a new signoff profile using the data currently displayed in the signoff profile display area.

Element	Description
Delete	This button is available only when Workflow Designer is in **Edit** mode.
	Click **Delete** to remove the selected profile from the current list of signoff profiles for the task.
Modify	This button is available only when Workflow Designer is in **Edit** mode.
	Click **Modify** to update the selected to reflect the data currently displayed in the signoff profile display area.
Close	Clicking **Close** dismisses the dialog box.
	As you make selections, the system enters into the database all selections made within the dialog box.

Migrating workflow attachments

Administrators can use the **migrate_wf_attachments** utility to migrate workflow attachments from VLA property-based attachments to GRM relation-based workflow task attachments. GRM relations are used for change related objects and proposed replica objects for remote workflows. It is possible to add the same object to the same workflow using different VLA property-based attachments and GRM relation-based workflow task attachments.

GRM relations are created between **EPMTask** as the primary object and the attachment as the secondary object. The attachment type determines the GRM relation.

Attachment type	GRM relation
Target	**Fnd0EPMTarget**
Reference	**Fnd0EPMReference**
Signoff	**Fnd0EPMSignoff**
ReleaseStatus	**Fnd0EPMReleaseStatus**
InterProcess Task	**Fnd0EPMInterProcessTask**
Schedule Task	**Fnd0EPMScheduleTask**
Replica Proposed Target	**Fnd0EPMReplicaTarget**

To support user attachments, a GRM relation type corresponding to a user attachment type is required.

For example, for a user attachment of type **1100**, there must be GRM relation named **Fnd0EPM_user_attach_1100**. Migration support is provided by the **migrate_wf_attachments** migration utility, as well as by run-time migration.

Note

The VLA data on the **EPMTask** is removed automatically during runtime migration only when all the attachments on the task are migrated successfully.

This improves search and reporting performance when large numbers of target attachments are added and removed by two synchronized VLAs rather than a GRM. This impacts the criteria for which the attachments are searched using saved queries.

For example, to search all the targets attached to a task with the task name, the query is constructed as shown. This an example of a reverse reference type of **Query** on **ItemRevision**.
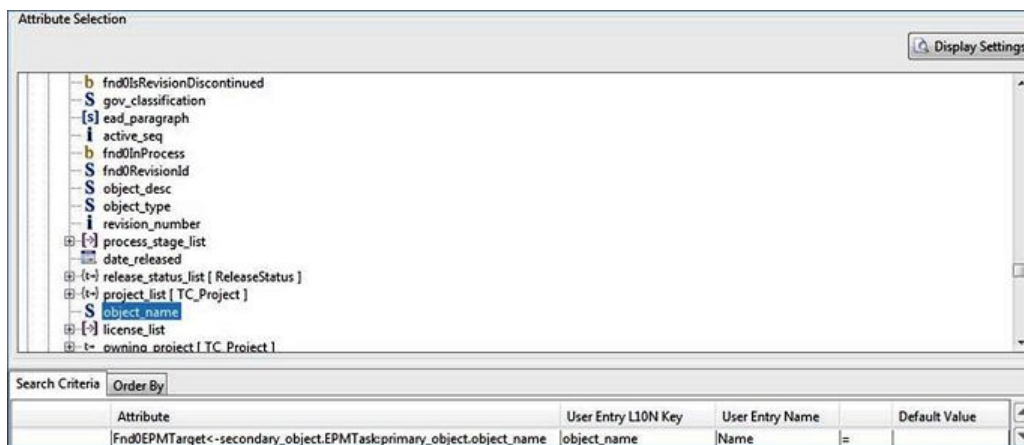
The **ItemRevision**s are added with a GRM relation **Fnd0EPMTarget** as secondary objects to the root **EPMTask**.

Example

You can create two workflow templates **SimpleDoExample** and **TestDoExample** and run each on two different item revisions, **IR1** and **IR2**. When the query is run with **SimpleDoExample** as the name in the query, only **IR1** is returned.

Use the **migrate_wf_attachments** utility to:

• Migrate active jobs during the upgrade process.

• Migrate completed jobs after the upgrade.

Note

Migration on-demand is supported. When a workflow process that has not been migrated is opened, the attachments are migrated automatically.

After successful migration, all the attachment data on the **EPMTask** should be obtained using the new GRM relations. The **EPMTask** attachment and the attachment types attributes are deprecated and are not to be used.

> **Note**
>
> The attachments attribute on **EPMTask** is deprecated in 11.x. Queries related to this attribute need to use the above GRM relations to refer to the attachment objects.

As an example, the following query uses the attachments attribute on the **EPMTask** from **process_stage_list** of **ItemRevision**. It checks the **user_name** of the **Signoff** objects added on these **EPMTasks**.

Create a simple **Review Task** process template and in the **Task** properties panel for both **Select-Signoff Task** and **Perform-Signoff Task**, check the option **Show Task in Process Stage List** so that the tasks appear in the **Process Stage List**.

Login as *tcadmin* and run the above Workflow on an Item Revision. Select **tcadmin** as reviewer and complete the **Select-Signoff Task**. Run the Teamcenter 10.x query that uses the attachments attribute of EPMTask, as shown below, and the **Item Revision** is listed as expected.

> **Note**
>
> This method works with 10.x.

Modify the query to use the corresponding GRM relation **Fnd0EPMSignoff** that is used for attaching the Signoff objects to Signoff tasks for Tc11.x.
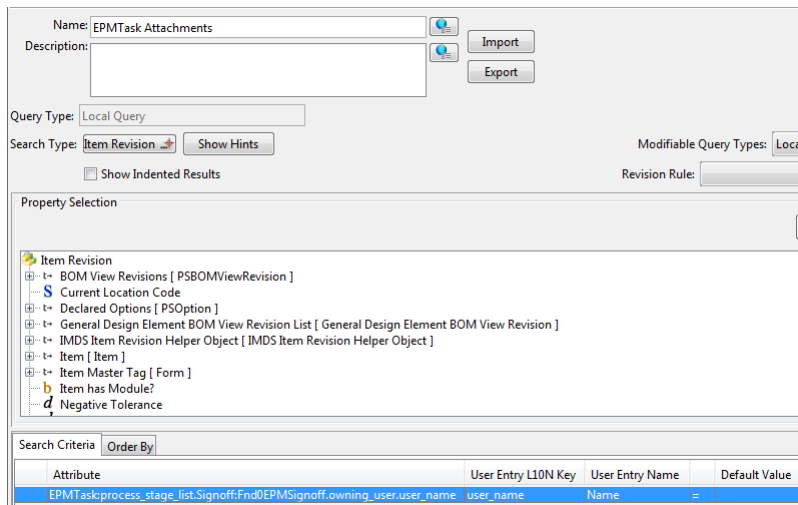
Editing active workflow processes

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.

- 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.

  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** preference to either **OPTIONAL** or **AUTOMATIC**.

Background processing for processes and tasks

Requirements for background processing

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 background prevents Workflow Designer from pausing until the edits complete.

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.

Background processing of workflow objects requires the following:

- 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.

- Configuration of asynchronous services.

You can also configure individual tasks in a workflow process to execute in the background with asynchronous processing.

> **Note**
>
> When a task is submitted for background processing, the task is removed from all inboxes.

Configure tasks for background processing

You can configure individual tasks in a workflow process to run in the background. If they are configured for background processing, all of those tasks' actions, except **Perform** and **Assign**, are processed asynchronously.

> **Note**
>
> Your system must meet the requirements for background processing.

1. Set the **EPM_task_execution_mode** preference to one of the following values:

   BACKGROUND   All tasks run in the background.

   This value overrides the **Execute Asynchronously** property value of each task template.

   CONFIGURABLE   Each task template's **Execute Asynchronously** property determines that task's processing.

   - If the value is **True**, the task runs in the background.

   - If the value is **False**, the task runs concurrently with your Workflow Designer interactions.

2. If you set the **EPM_task_execution_mode** preference value to **CONFIGURABLE**, open Workflow Designer and select the process template with the tasks you want to run in the background.

3. In **Edit** mode, select the task, and then click the **Task Attributes** button.

4. Select the **Process in Background** check box and close the **Attributes** dialog box.

    This action changes the **Execute Asynchronously** property value to **True** in the **Properties** dialog box.

    Repeat this step for each task you want to run in the background

    •   Child tasks of those chosen to process in the background are processed in the background also.

    •   You can set only the root task and its children to background processing.

5. When you have configured all the tasks in the workflow process template you want to run in the background, select the **Set Stage to Available** check box and click **Yes** in the **Stage Change** dialog box.

When you create a workflow using the process template, the workflow runs the tasks that have the **Process in Background** check box selected in the background.

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.

Delete key removes workflow objects and backspace key removes text

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.

Save time when creating multiple tasks of the same type

When creating a workflow process template, sometimes the process calls for several of the same types of tasks, such as several **Do** tasks, that have the same or similar set of handlers and arguments.

Instead of adding the tasks, selecting the handlers, and typing the arguments and values individually, you can do the following:

1. Add the first task to the process template.

2. Select the handlers you want to add and type the arguments and values for each one.

3. Copy the task and paste it back in the process template.

4. Edit the handler arguments and values in the new copy of the task.

This saves you the time and effort of retyping arguments and values as well as reduces the possibility of typos when creating your process template.

Move and resize 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** ![icon] button to open the **Handler** dialog box.

2. Double-click anywhere in the dialog box to undock it.

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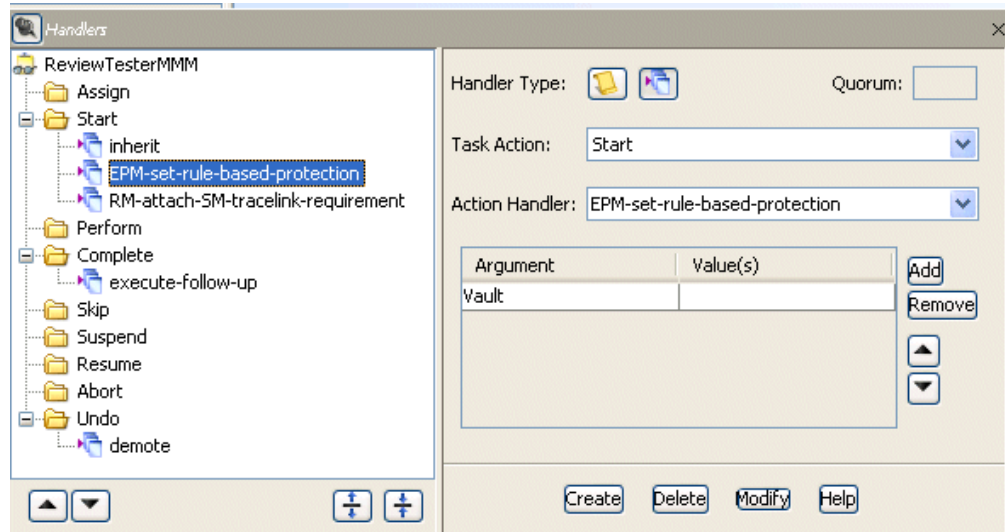
Behavior	Example
Undocked	
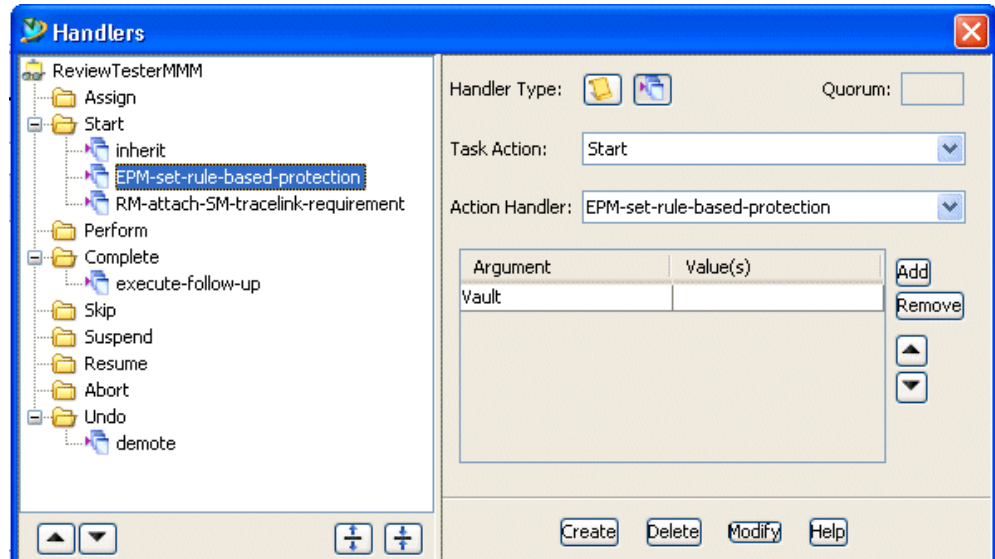
When you leave the **Handler** dialog box docked, you can move between one task's handlers and another task's handlers by selecting a different task in the task hierarchy tree. For example:

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.
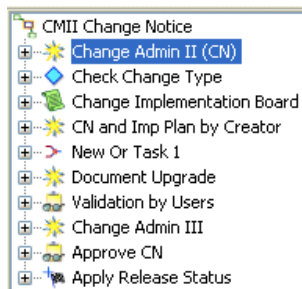
Task hierarchy tree   **Handler dialog box**
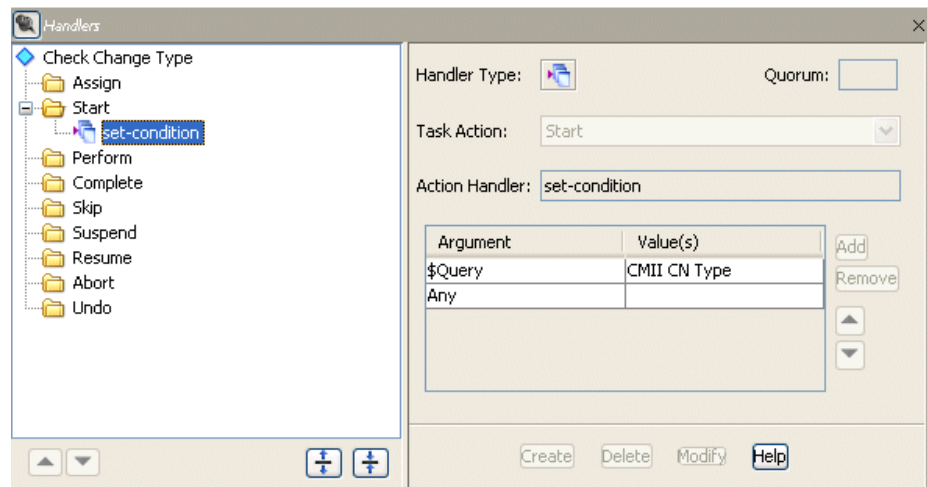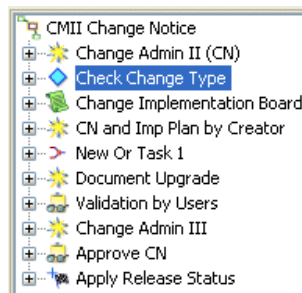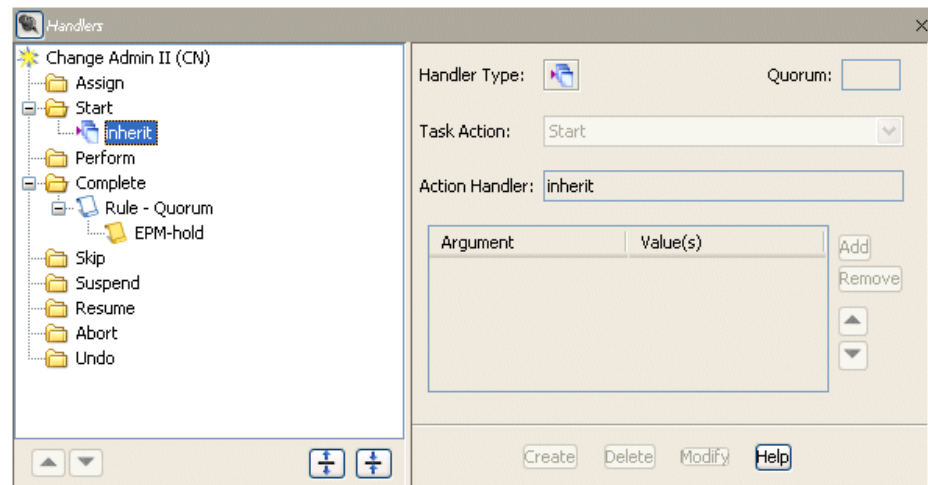
Workflow errors

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

• If there are two tasks in a workflow process template and a handler on the **Start** action of the second task generates an error after the first task completes successfully, the workflow displays a **Warning** dialog box with the following error message instead of an **Error** dialog box.

```
The action was successful. Additional information has been included on the
error stack.
```

> **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.

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.

    o   A perspective exists in a window along with any number of other perspectives, but only one perspective can be displayed at a time.

    o   In applications that use multiple views, you can add and rearrange views to display multiple sets of information simultaneously within a perspective.

    o   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, using rich client views and view networks, you can navigate to a hierarchy of information, display information about selected objects, open an editor, or display properties.

    o   Views that work with related information typically react to selection changes in other views.

    o   Changes to data made in a view can be saved immediately.

    o   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.

    o   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.

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

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.

Chapter 2: Creating workflow process templates

Structuring a workflow process

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. With these characteristics, individual tasks can run sequentially (serially) or asynchronously (in parallel).

A *workflow process template* is a predefined workflow structure, which you can use as a pattern for your own workflow processes. You can define a specific workflow process by placing workflow tasks 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*.
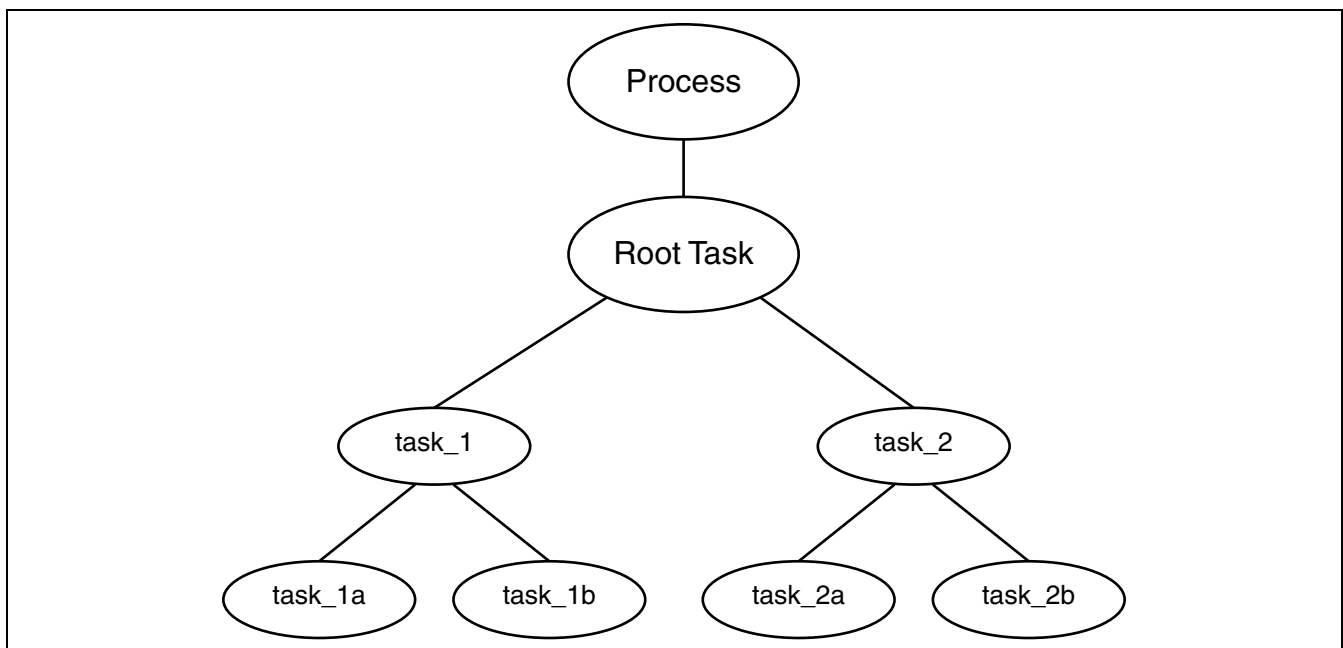
Sample EPM workflow process structure

The root task is the top-level parent task that contains all the other tasks as subtasks. It is the first task run 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 **CN Fast Track** task.
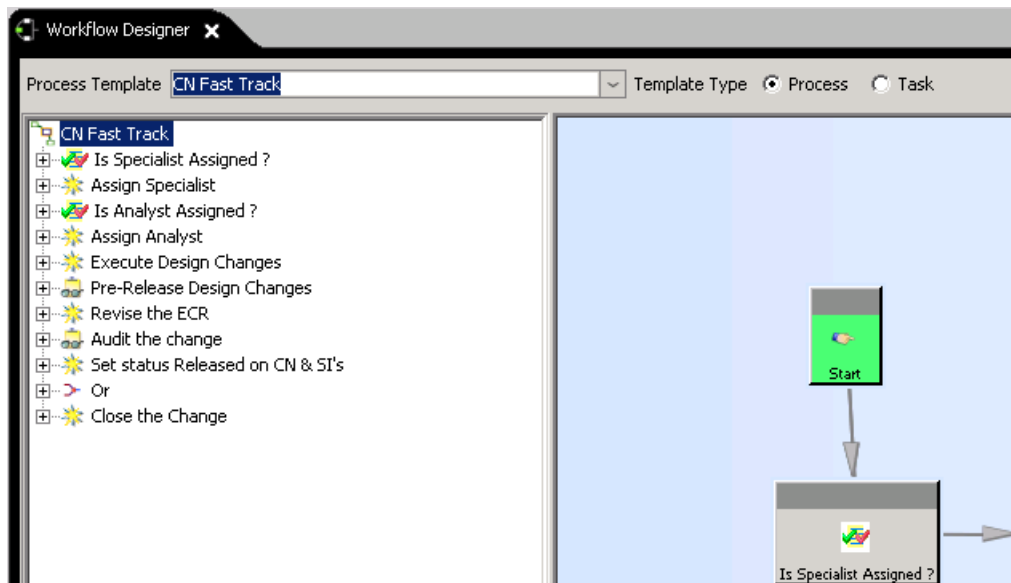
To place handlers on the root task, select the **Start** node and click the **Handlers** button.

> **Note**
>
> A default workflow process template is defined using the **WorkspaceObject_default_workflow_template** preference. This preference does not affect new revisions.
>
> The specified template is automatically selected when submitting existing workspace objects to a *new* workflow process. This preference does not apply to object creation. For new objects, the default workflow template selection is intentionally left blank. Workflow template filters are used to restrict the list of available templates for both existing and new objects.
>
> To set a default workflow template on a new revision, select the process manually when creating the revision.

Example of 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.
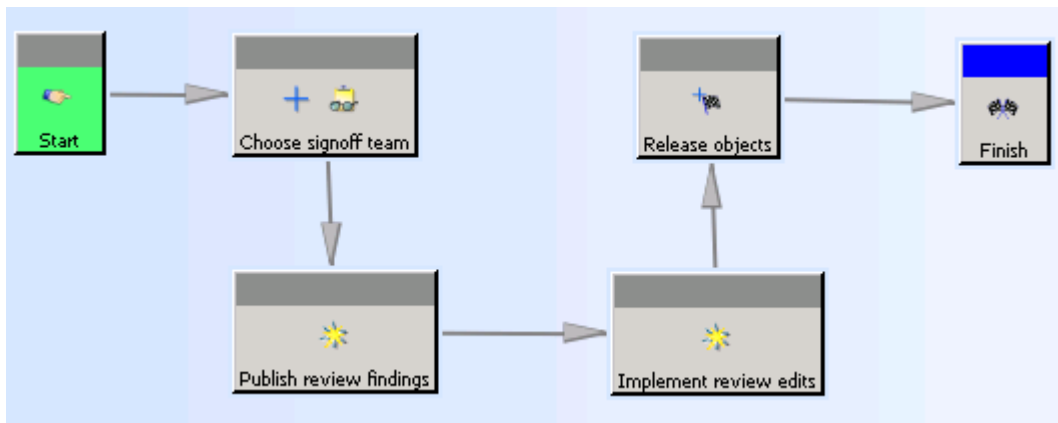
After you finish a new workflow process template, you select the **Set Stage to Available** check box so that the template appears in the **Task Hierarchy** list.

> **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 workflow process templates

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

   The list displays the available workflow process templates.

   When you choose an existing template from the **Based On Root Template** list, the task hierarchy tree and the viewer display workflow process and task information for the selected template. Selecting a task from the tree 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 in the **New Root Template** dialog box; it cannot be modified.

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.

4.   For **Template Type**, select **Process**.

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**

      Configure the workflow task actions and states.

      Configure the explicit and assumed links.

   •   **Task template**

      Configure the attributes and handlers.

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.

Note

Workflow template filters affect:

- The **Process Template** choices displayed by the **New Process Dialog** dialog box.

- The **Process Template List** choices displayed by the **New Item** dialog box **Define Workflow Information** page.

Administrators and customizers can use Business Modeler IDE conditions to configure workflow template filters.

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

1. Choose **File→New Root 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.
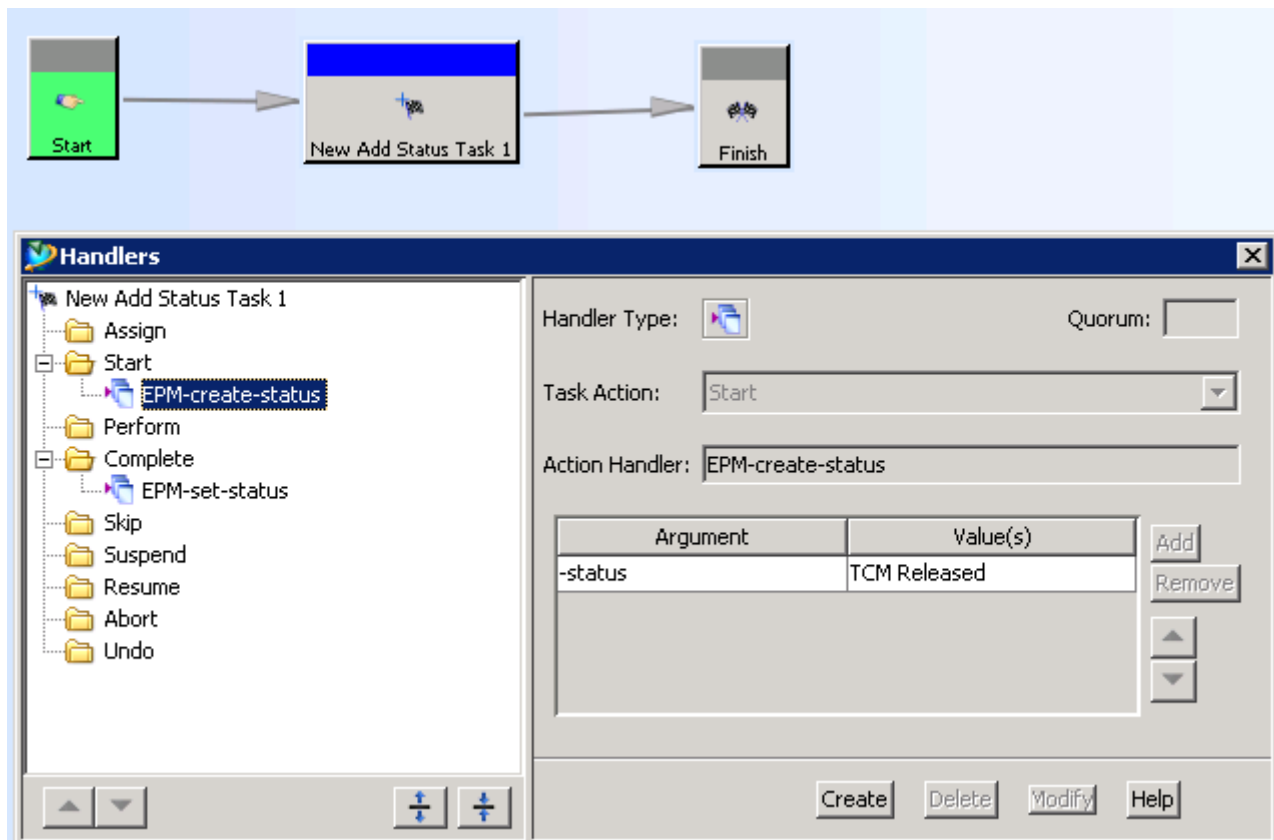
By adding the **Add Status** task, your new quick-release workflow process template contains the required **EPM-create-status** and **EPM-set-status** handlers.

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

Creating Custom Templates

1. Choose **File → New Root Template**.

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

2. In the **New Root Template** dialog box:

    • Select the **Task** option for **Template Type**.

    • Type a name in the **New Root Template Name** box.

    • Select **Empty Template** from the **Based on Root Template** list.

3. Click **OK**.

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

5. Select **Tools → Export**.

6. Select the newly created task and add it to the list of selected tasks.

   Click **OK**.

7. From the desktop, open the exported task template XML file.

   Add the following:

   - `objectType=" EPM<YOUR_CUSTOM_TASK>TaskTemplate "`

   - `iconKey="<YOUR_CUSTOM_TASK_KEY>"`

8. Save the XML file.

9. Select **Tools → Import** to import the template.

   The task is now available.

Creating subprocesses

What are workflow subprocesses?

Subprocesses are child workflow processes of a parent workflow process. You can create subprocesses while performing tasks from your worklist.

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

A typical scenario is one in which you receive a task in your worklist that is dependent upon the completion of an additional workflow process. You decide to create a workflow subprocess to track the work which must be completed before you can complete the task in the parent workflow.

Subprocesses are created in two locations:

Parent workflow templates	Administrators can . For example, a parent workflow template can be configured to automatically launch subprocesses for each target of the parent workflow.

My Worklist   End users can while performing tasks from their worklist or from Workflow Viewer. Generally, any user can create a workflow subprocess from a task within their worklist. This functionality is not limited to privileged users.

When you create a workflow subprocess from an in-process task in your worklist, you create a dependency between the selected task in the parent process and the newly created subprocess. The targets of the active parent workflow process are carried over if you check the **Inherit Targets** box. If a subprocess is created from an in-process task, the task cannot complete until the subprocess completes.

> **Note**
>
> The behavior of the **Inherit Targets** box is determined by the **EPM_multiple_processes_targets** and **EPM_sub_process_target_inheritance** preferences.
>
> Regardless of how these two preferences are set to control the inheritance of target objects from the parent process, users can always manually add or remove targets from subprocesses.

> **Note**
>
> Access to create workflow subprocesses is governed by the Access Manager **Has Class (Task)** rule and the **Task Named** access control list (ACL). The same permissions allowing you to perform the task allows you to create a subprocess from the task.

> **Note**
>
> Workflow subprocesses are not always dependent on parent processes. The **WRKFLW_skip_abort_on_sub_process** preference is honored only for *independent* subprocesses.
>
> Set the **WRKFLW_skip_abort_on_sub_process** preference to **true** to skip abort of subprocess when a parent process is aborted.
>
> If there is a dependency from a parent process to its subprocesses, aborting the parent will abort the *dependent* subprocesses, irrespective of the value of the preference.
>
> The default value is **false** which will abort the subprocesses along with parent process.

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. This would be useful if you want each target to undergo a separate review and signoff 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*.

Argument	Value
-template	**Clinical Trials Phase I**
-from_attach	**ALL**
-to_attach	**ALL**

- 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.

Argument	Value
-template	**Clinical Trials Phase I**
-from_attach	**ALL**
-to_attach	**TARGET**
-exclude_type	**ItemRevision**

- 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.

Argument	Value
-template	**Clinical Trials Phase I**
-from_attach	**ALL**

Argument	Value
-to_attach	**TARGET**
-include_type	**ItemRevision**
-multiple_processes	

Creating subprocesses for multiple targets

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

The most straightforward method to create subprocesses for multiple targets 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_type** 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_type** argument along with the **-multiple_processes** argument to create individual subprocesses for each target except the specified types in the parent process.

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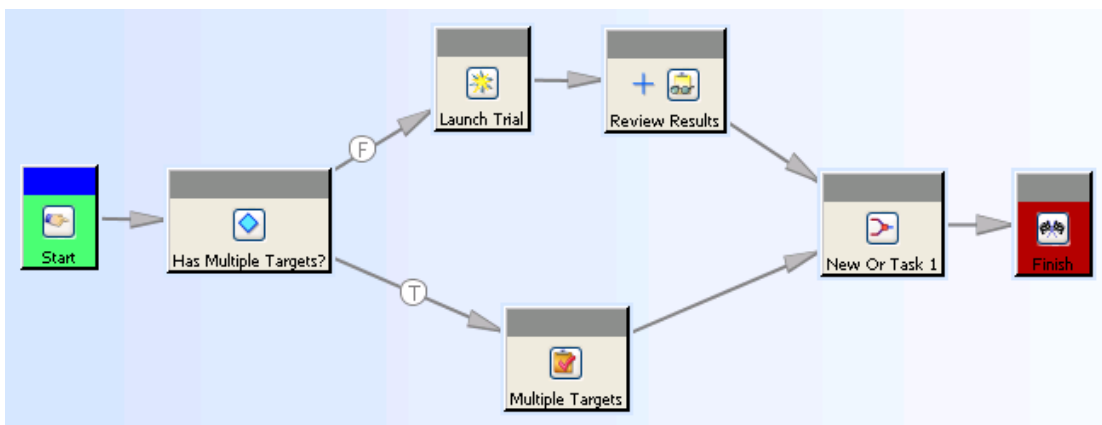
In this example, Pharmaceuticals, Inc., uses such a workflow for its drug trial reviews. The typical trial contains multiple products, but occasionally a trial contains only one product.

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** preference is set to **ON** by choosing **Edit→Options** to launch the **Options** dialog box and locating the preference using the **Filters** link.
>
> If the preference is not created at your site, create the preference and set it to **ON**.

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, select **Empty Template** from the **Based On Root Template** list, 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. 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 , create a new query called **WF - Has Multiple Targets** by completing the query boxes as shown and clicking **Create**.
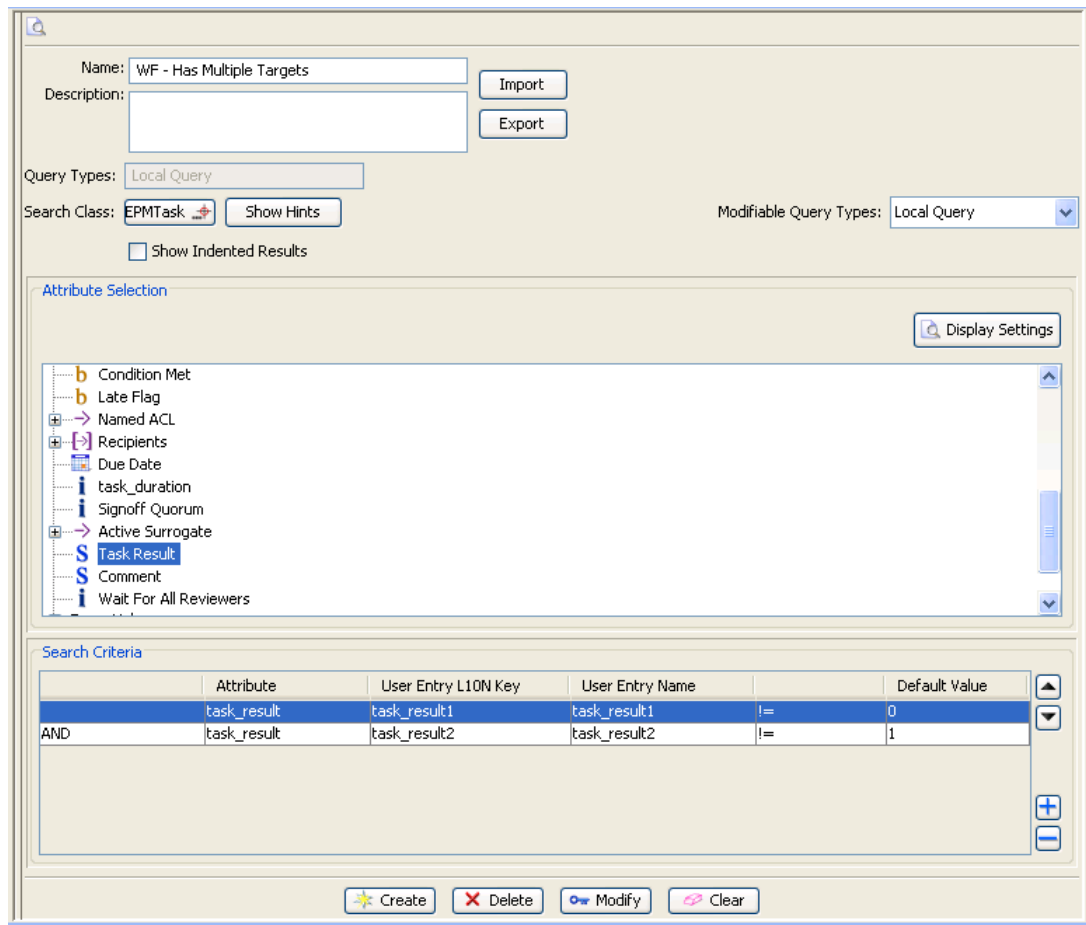
c.   Return to Workflow Designer.

7.   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 bottom of the dialog box.

   d.   Select **Task** as the **Query Against** option.

   e.   Click **OK** to choose the query and exit the dialog box.

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

   f.   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.

8. 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.

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

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

9. When you created the **WF - Has Multiple Targets** query on the **Has Multiple Targets?** task, the **EPM-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.

10. Select the **EPM-set-task-result-to-property** handler in the folder list and click the **Up** button ▲ under the folder list to move it above the **EPM-set-condition** handler in the **Start** action.

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

11. Close the **Handlers** dialog box.

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

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

14. 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 **-attachment** 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.   Select the **EPM-attach-related-objects** handler you just created from the folder list on the left.

    j.   Replace **IMAN_specification** with **PSBOMViewRevision** as the value for the **-relation** argument and click **Create**.

         You should have two **EPM-attach-related-objects** handlers in the **Start** action, one with the **IMAN_specification** relation and one with the **PSBOMViewRevision** relation.

    k.   Close the **Handlers** dialog box.

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

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

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

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

19. 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 **-from_attach** in the **Argument** box and **Target** in the **Value(s)** box.

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

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

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

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

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

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

l. Type **-template** in the **Argument** box and the name for this template that you used in step 2 in the **Value(s)** box.

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

    The system responds with a warning that says `The use of EPM-create-sub-process handler has resulted in a loop`. Teamcenter detected that the **-template** argument referenced the template that you are creating. However, since the subprocesses generated will follow the **False** path, no loop occurs. Click **OK**.

n. Close the **Handlers** dialog box.

20. Create an **Or** task to reconcile the **True** and **False** paths by clicking 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.

21. 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.

22. Draw a flow path from the **Has Multiple Targets?** task to the **Launch Trial** task.

    By default, the path is a **True** path.

23. Change the flow path to a **False** path by right-clicking the line you have just drawn and choosing **Set Path To False Path**.

    The flow path changes to a **False** path.

24. Draw a flow path from the **Has Multiple Targets?** task to the **Multiple Targets** task.

    By default, the path is a **True** path.

25. 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.

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

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

28. Draw a flow path from the **Or** task to the **Finish** node.

29. Select the **Set Stage to Available** check box to put your template online.

The template is now ready to use.

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.

Argument	Behavior
-process_assembly	Searches for assemblies in the target, reference, or all (as specified by the **-from_attach** argument) and creates subprocesses for each component.
-depth	Specifies the depth to which the assembly is traversed.
-rev_rule	Specifies the revision rule applied to the assembly.
-include_related_type	Creates subprocesses only for assembly components of the types specified in this argument.
-exclude_related_type	Does not creates subprocesses for assembly components of the types specified in this argument.

> **Note**
>
> The **-include_related_type** and **-exclude_related_type** arguments can be used in conjunction with each other. If used in conjunction, the **-include_related_type** argument takes precedence; first the objects are processed against **-include_related_type** and then processed against **-exclude_related_type**.

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.

Argument	Behavior
-relation	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.)
-include_related_type	Creates subprocesses only for related objects of the type(s) specified in this argument.
-exclude_related_type	Does not creates subprocesses for related objects of the type(s) specified in this argument.

> **Note**
>
> The **-include_related_type** and **-exclude_related_type** arguments can be used in conjunction with each other. If used in conjunction, the **-include_related_type** argument takes precedence; first the objects are processed against **-include_related_type**, and then **-exclude_related_type**.

Creating ad hoc subprocesses

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

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

Associate templates with a target object type and a user groupSelect a default process template

> **Caution**
>
> This feature is deprecated as of Teamcenter 11.2. By default, conditions written in the Business Modeler IDE control template association.
>
> Siemens PLM Software recommends that you use Business Modeler IDE conditions to associate templates. Conditions offer greater versatility, with criteria such as session group, role, and user; target project and target release status; and custom criteria, both session-specific and target-specific, that a Teamcenter administrator can create.
>
> To continue using the deprecated feature, you must:
>
> • Set the **WRKFLW_use_legacy_template_filter** preference to **true**.
>
> • Set the **CR_allow_alternate_procedures** preference to **Assigned** or **any**.
>
> Also deprecated are some other preferences that this feature uses. These preference names follow the pattern of **TC_**_object-type_**_release_procedure**.

Based on the target's object type and the initiating user's group, you can define which workflow process templates appear in the **Assigned Process Template** list of the **Process Template Filter Dialog** dialog box.

You can select the required filtering criteria when a template is in edit mode. When the **CR_allow_alternate_procedures** preference is set to **none**, additional filtering criteria applies to targets that are added while the workflow is in progress.

> **Note**
>
> - If you associate templates with object types that 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 that has associated templates for that object type. If you explicitly associate templates with a subgroup, the subgroup does not inherit templates from its parent group.

1. Choose **Edit→Template Filter**.

    The **Process Template Filter Dialog** dialog box opens.
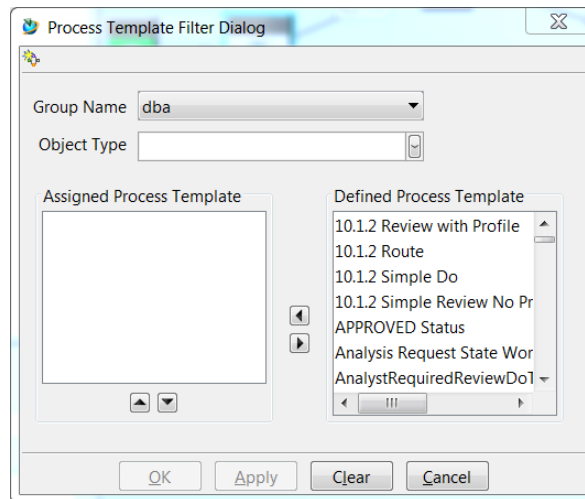
2. From the **Group Name** list, select the group whose workflow process template list you want to filter.

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.

- **Cancel** to close the dialog box without applying the changes.

> **Note**
>
> To remove a workflow process template from the **Assigned Process Template** list, select the template and click the ▶ button. The selected template is moved to the **Defined Process Template** list.

Core templates

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

Template name	Task template definition type	Task type value specified in task template	Executing task's real type	Executing task's task type
Process	EPMTaskDefinition	EPMTask	EPMTask	EPMTask
Review Process	EPMTaskDefinition	EPMTask	EPMTask	EPMTask
Task	EPMTaskDefinition	EPMTask	EPMTask	EPMTask
Review Task	EPMTaskDefinition	EPMReviewTask	EPMTask	EPMReviewTask
Do Task	EPMDoTaskDefinition	EPMDoTask	EPMTask	EPMDoTask
Or Task	EPMTaskDefinition	EPMTask	EPMTask	EPMTask
Add Status Task	EPMTaskDefinition	EPMTask	EPMTask	EPMTask
Change Management Procedure	EPMTaskDefinition	EPMTask	EPMTask	EPMTask
Change Management Item	EPMTaskDefinition	EPMTask	EPMTask	EPMTask

Delete workflow process 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.

Workflow examples

Change Manager workflow example

You can change this example to match your participants, organization, and conditions.

> **Note**
>
> If you are using Aerospace and Defense business objects (for example, **Adc0ChangeRqstRevision**), you can add them to the **-type** and **-include_types** arguments.

When this example is completed, the workflow should look like the following.
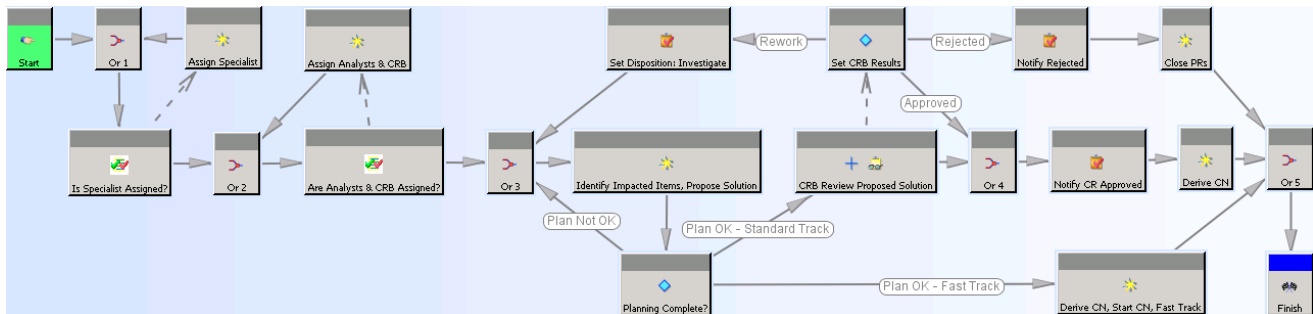
1.   In Workflow Designer, choose **File→New Root Template**, name your template, select **Empty Template** as your root template, and then click **OK**.

> **Note**
>
> Ensure that the **EPM-assign-team-selector** and **EPM-auto-assign-rest** handlers are attached to the **Start** task action.

2.   To the **Start** task action, add the **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	**CMIsFastTrack**
-value	**No**
-to_attach	**TARGET**

Arguments	Values
-include_type	ChangeRequestRevision
-bypass	

These arguments set the ECR **CMIsFastTrack** property to **No**, which ensures that the ECR starts on the standard track.

3. Create an **Or** task named **Or 1**, and draw a path from the **Start** task.

4. Create a **Validate** task named **Is Specialist Assigned?**, to check the ECR for an assigned change specialist.

   • Add the **EPM-check-object-properties** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-include_type	ChangeRequestRevision
-property	ChangeSpecialist
-attachment	target

   • Draw a path from the **Or 1** task.

   A **Validate** task needs tasks at the end of a **Complete** path and **Error** path. If a change specialist is not assigned, you must correct that error.

5. In case a change specialist is not assigned, create a **Do** task named **Assign Specialist**.

   • For the **Do** task, add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	resourcepool:Change Management::Manager

   • Draw an **Error** path from the **Is Specialist Assigned?** task.

   • Draw a **Complete** path to the **Or 1** task.

   This assigns the task to any user who has the Manager role in the Change Management group. The Manager must edit the ECR object to add a change specialist 1 to it. Once that is done, the user can go back to the workflow, click **Complete** on the task, and the workflow moves along the **Complete** path.

   > **Note**
   >
   > By default, the **Do** task has automatically configured **EPM-check-condition**, **EPM-inherit**, and **EPM-hold** handlers. You do not have to alter these.

6. Create an **Or** task named **Or 2**, and draw a **Complete** path from the **Is Specialist Assigned?** task.

7. Create a **Validate** task named **Are Analyst & CRB Assigned?**, to check if an analyst or change review board members are assigned to the ECR.

    •   Add the **EPM-check-object-properties** handler to the **Start** task action of this task with the following arguments and values:

Arguments	Values
-include_type	ChangeRequestRevision
-property	Analyst,ChangeReviewBoard
-attachment	target

    •   Draw a **Complete** path from the **Or 2** task.

8. In case an analyst or change review board members are not assigned, create a **Do** task named **Assign Analyst & CRB**.

    •   For the **Do** task, add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	$CHANGE_SPECIALIST1

    •   Draw an **Error** path from the **Are Analysts & CRB Assigned?** task.

    •   Draw a **Complete** path to the **Or 2** task.

    This assigns the task to the user who has been assigned as the change specialist 1 for the ECR. The change specialist 1 must edit the ECR object to add the missing analyst or change review board members to it. Once that is done, the user can go back to the workflow, click **Complete** on the task, and the workflow moves along the **Complete** path.

    > **Note**
    >
    > By default, the **Do** task has automatically configured **EPM-check-condition**, **EPM-inherit**, and **EPM-hold** handlers. You do not have to alter these.

9. Create an **Or** task named **Or 3**, and draw a **Complete** path from the **Are Analysts & CRB Assigned?** task.

10. Create a **Do** task named **Identify Impacted Items, Propose Solution**.

    •   Add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	$ANALYST

- Draw a **Complete** path from the **Or 3** task.

This assigns the task to the user who has been assigned as the analyst for the ECR. The analyst follows the instructions in the workflow. Once that is done, the analyst can go back to the workflow, click **Complete** on the task, and the workflow moves along the **Complete** path.

> **Note**
>
> By default, the **Do** task has automatically configured **EPM-inherit** and **EPM-hold** handlers. You do not have to alter these.

11. Create a **Condition** task named **Planning Complete?**.

- Add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	$CHANGE_SPECIALIST1

- Draw a **Complete** path from the **Identify Impacted Items, Propose Solution** task.

- Draw a custom path named **Plan Not OK** to the **Or 3** task.

This assigns the task to the user who has been assigned as the change specialist 1 for the ECR. The change specialist 1 follows the instructions in the workflow. Once that is done, the analyst can go back to the workflow and select one of the three paths based on the results. The three paths are added once more tasks further along the workflow are created.

> **Note**
>
> By default, the **Condition** task has automatically configured the **EPM-check-condition** handler. You do not have to alter it.

12. Create a **Review** task named **CRB Review Proposed Solution**.

- Add the **EPM-set-property** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-property	CMMaturity
-value	Reviewing
-to_attach	TARGET
-include_type	ChangeRequestRevision
-bypass	

- Draw a custom path named **Plan OK – Standard Track** from the **Planning Complete?** task.

This sets the ECR's **Maturity** property to **Reviewing**, which notes that the change review board is looking at the proposed change.

> **Note**
>
> By default, the **Review** task has automatically configured the **EPM-inherit**, **EPM-set-rule-base-protection**, and **EPM-execute-follow-up** handlers. You do not have to alter these.

13. Create a **Condition** task named **Set CRB Results**.

   • Add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	$CHANGE_SPECIALIST1

   • Draw an **Error** path from the **CRB Review Proposed Solution** task.

   This assigns the task to the user who has been assigned as the change specialist 1 for the ECR. The change specialist 1 follows the instructions in the workflow. Once that is done, the analyst can go back to the workflow and select one of the three paths based on the results. The three paths are added once more tasks further along the workflow are created.

> **Note**
>
> By default, the **Condition** task has automatically configured the **EPM-check-condition** handler. You do not have to alter it.

14. Create a custom task named **Set Disposition: Investigate**.

   • Add the **EPM-set-property** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-property	CMDisposition
-value	Investigate
-to_attach	TARGET
-include_type	ChangeRequestRevision
-bypass	

   • Draw a custom path named **Rework** from the **Set CRB Results** task.

   • Draw a **Complete** path to the **Or 3** task.

   This sets the ECR's **Disposition** property to **Investigate**, which indicates the analyst needs to do more work on the ECR.

15. Create a custom task named **Notify Rejected**.

   • Add the **EPM-set-property** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-property	CMDisposition
-value	Disapproved
-to_attach	TARGET
-include_type	ChangeRequestRevision
-bypass	

   This sets the ECR's **Disposition** property to **Disapproved**, which indicates no further action is to be taken with the ECR.

   • Add the **EPM-notify** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-recipient	$REQUESTOR,$ANALYST
-subject	CR Rejected
-attachment	$TARGET

   This sends an e-mail to the ECR requestor and analyst notifying them that the ECR has been rejected by the change review board.

   • Draw a custom path named **Rejected** from the **Set CRB Results** task.

16. Create a **Do** task named **Close PRs**.

   • Add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	$CHANGE_SPECIALIST1

This assigns the task to the user who has been assigned as the change specialist 1 for the ECR. The analyst follows the instructions in the workflow. Once that is done, the analyst can go back to the workflow, click **Complete** on the task, and the workflow moves along the **Complete** path.

- To the **Start** task action, add a **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	CMClosure,CMMaturity
-value	Closed,Complete
-to_attach	TARGET
-include_types	ChangeRequestRevision
-bypass	

This sets the ECR's **Closure** and **Maturity** properties to **Closed** and **Complete**, respectively, which closes out the ECR.

- Draw a **Complete** path from the **Notify Rejected** task.

> **Note**
>
> By default, the **Do** task has automatically configured **EPM-inherit** and **EPM-hold** handlers. You do not have to alter these.

17. Create an **Or** task named **Or 4**.

   - Draw a custom path named **Approved** from the **Set CRB Results** task.

   - Draw a **Complete** path from the **CRB Review Proposed Solution** task.

18. Create a custom task named **Notify CR Approved**.

   - Add the **EPM-set-property** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-property	CMDisposition
-value	Approved
-to_attach	TARGET
-include_type	ChangeRequestRevision
-bypass	

This sets the ECR's **Disposition** property to **Approved**, which allows a change notice to be derived from the ECR.

- Add the **EPM-notify** handler to the **Start** task action with the following arguments and values:

Arguments	Values
-recipient	**$REQUESTOR,$ANALYST**
-subject	**CR Approved**
-attachment	**$TARGET**

This sends an e-mail to the ECR requestor and analyst notifying them that the ECR has been approved by the change review board.

- Draw a **Complete** path from the **Or 4** task.

19. Create a **Do** task named **Derive CN**.

- Add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	**$CHANGE_SPECIALIST1**

This assigns the task to the user who has been assigned as the change specialist 1 for the ECR. The analyst follows the instructions in the workflow. Once that is done, the analyst can go back to the workflow, click **Complete** on the task, and the workflow moves along the **Complete** path.

- To the **Start** task action, add a **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	**CMMaturity**
-value	**Reviewing**
-to_attach	**TARGET**
-include_type	**ChangeRequestRevision**
-bypass	

This sets the ECR's **Maturity** property to **Reviewing**, which allows an ECN to be derived.

- To the **Complete** task action, add a **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	**CMMaturity**
-value	**Executing**
-to_attach	**TARGET**
-include_type	**ChangeRequestRevision**

Arguments	Values
-bypass	

This sets the ECR's **Maturity** property to **Executing**, which closes out the ECR after the ECN has been derived.

> **Note**
>
> By default, the **Do** task has automatically configured **EPM-inherit** and **EPM-hold** handlers. You do not have to alter these.

20. Create a **Do** task named **Derive CN, Start CN, Fast Track**.

    • Add the **EPM-auto-assign** handler to the **Start** task action with the following argument and value:

Arguments	Values
-assignee	**$CHANGE_SPECIALIST1**

This assigns the task to the user who has been assigned as the change specialist 1 for the ECR. The analyst follows the instructions in the workflow. Once that is done, the analyst can go back to the workflow, click **Complete** on the task, and the workflow moves along the **Complete** path.

    • To the **Start** task action, add a **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	**CMMaturity,CMDisposition**
-value	**Reviewing,Approved**
-to_attach	**TARGET**
-include_type	**ChangeRequestRevision**
-bypass	

This sets the ECR's **Maturity** and **Disposition** properties to **Reviewing** and **Approved**, respectively, which allows the ECR to be placed on the fast track.

    • To the **Start** task action, add another **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	**CMIsFastTrack**
-value	**Yes**
-to_attach	**TARGET**
-include_type	**ChangeRequestRevision**

Arguments	Values
-bypass	

This sets the ECR's **Is Fast Track?** property to **Yes**, which notes the ECR went through the fast track process.

- To the **Complete** task action, add another **EPM-set-property** handler with the following arguments and values:

Arguments	Values
-property	**CMMaturity**
-value	**Executing**
-to_attach	**TARGET**
-include_type	**ChangeRequestRevision**
-bypass	

This sets the ECR's **CMMaturity** property to **Executing**, which completes the ECR in the change process and allows a change notice to be derived from it.

> **Note**
>
> By default, the **Do** task has automatically configured **inherit** and **EPM-hold** handlers. You do not have to alter these.

21. Create an **Or** task named **Or 5**.

- Draw a **Complete** path from the **Close PRs** task.

- Draw a **Complete** path from the **Derive CN** task.

- Draw a **Complete** path from the **Derive CN, Start CN, Fast Track** task.

- Draw a **Complete** path to the **Finish** task.

You can apply this workflow to any ECR revision object.
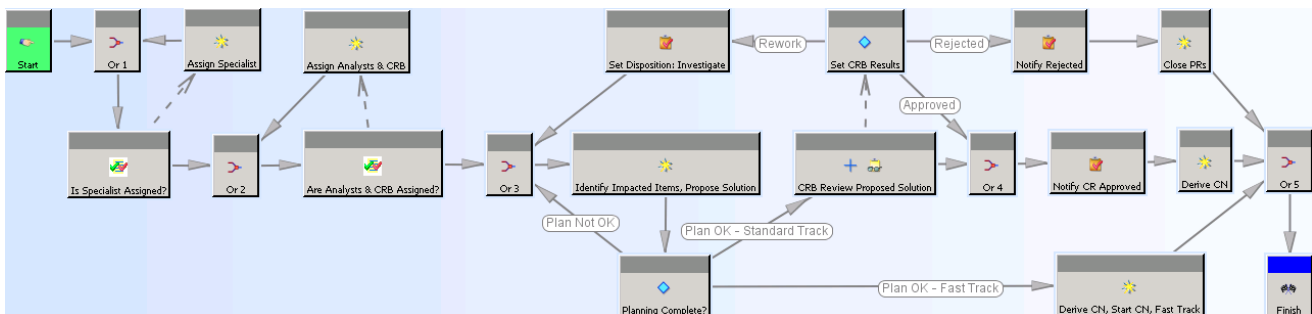
Add Status task example: Replace status of target objects

ACMERP workflow process

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**.

Start task
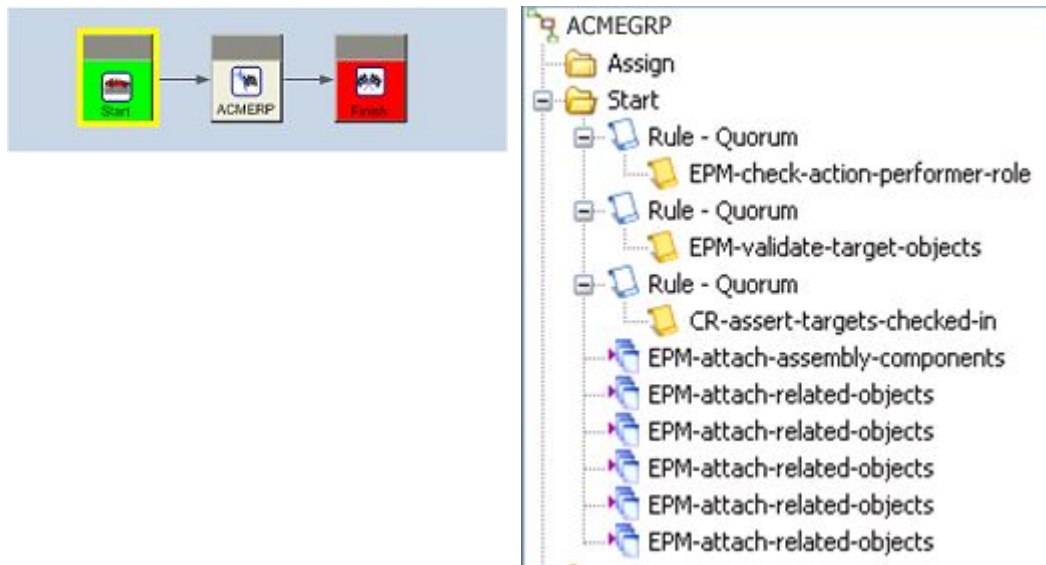
The **Start** node contains all the handlers for the root task. The root task contains all the other tasks within a workflow process. It is the first task to start and the last task to complete. Therefore, the handlers placed on the root task control the beginning and end of the workflow process itself, 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.

> **Note**
>
> Do not place handlers on the **-perform** action of an **Add Status** task, as they are not executed on this task type.

Start action	**Rule handler: EPM-check-action-performer-role**
Arguments:Values	**-responsible:DBA** **-responsible:ME**
Description:	Checks whether a member of the **DBA** or **ME** groups initiated the workflow. If not, the workflow does not proceed.
Start action	**Rule handler: EPM-validate-target-objects**
Arguments:Values	**-include_type:ACMEPartMfgRevision,ACMEMEProcessRevision, ACMEMEOPRevision**
Description:	Restricts the types of objects that can be added as target objects to **ACMEPartMfgRevision**, **ACMEMEProcessRevision** and **ACMEMEOPRevision**.
Start action	**Rule handler: EPM-assert-targets-checked-in**
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.
Start action	**Action handler: PS-attach-assembly-components**
Arguments:Values	**-depth:1** **-exclude_released** **-rev_rule:Working** **-include_related_type: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.

Start action	**Action handler: EPM-attach-related-objects**
Arguments:Values	**-relation:IMAN_METarget**
	-attachment:target
Description:	Attaches all objects with an **IMAN_METarget** relation as targets of the workflow.
Start action	**Action handler: EPM-attach-related-objects**
Arguments:Values	**-relation:IMAN_specification**
	-attachment:target
Description:	Attaches all objects with an **IMAN_specification** relation as targets of the workflow.
Start action	**Action handler: EPM-attach-related-objects**
Arguments:Values	**-relation:IMAN_Rendering**
	-attachment:target
Description:	Attaches all objects with an **IMAN_Rendering** relation as targets of the workflow.
Start action	**Action handler: EPM-attach-related-objects**
Arguments:Values	**-relation:IMAN_Reference**
	-attachment: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**
	-attachment:target
Description:	Attaches all objects with a **PSBOMViewRevision** relation as targets of the workflow.

ACMERP (Add Status task)
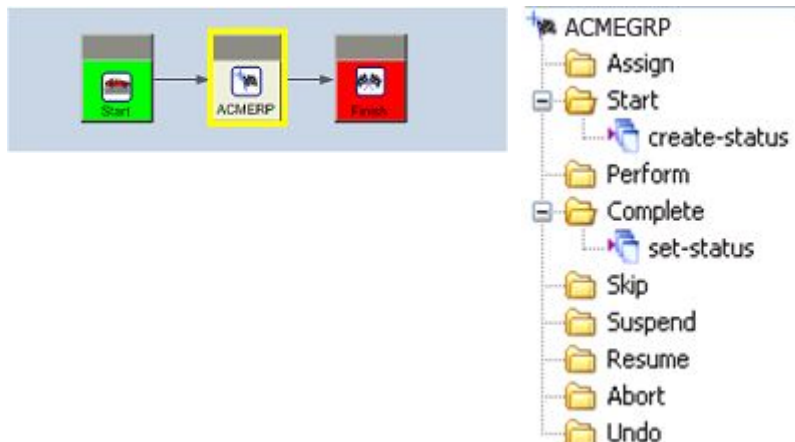
In this workflow example, handlers placed on the **Start** action of the **ACMERP** task:

• 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: EPM-create-status**
Arguments:Values	**-status:ACMERP**
Description:	Attaches the **ACMERP** status to the **ACMERP** task. **Note** The **ACMERP** status should be already defined in the Business Modeler IDE.
Complete action	**Action handler: EPM-set-status**
Arguments:Values	**-action:replace**
Description:	Deletes all existing statuses assigned to any target objects and replaces them with the **ACMERP** status.

Chapter 3: Editing workflow process 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 Mode** button.

Consider the following questions before editing a workflow template.

Editing task	Description
Edit offline or online?	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.
	Online editing allows users to initiate workflows based on the old version of the workflow template, while you edit a copy of the same 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.
Apply edits to running workflow processes?	After editing a workflow template, you can apply the edits to all active processes that are based on the template. When you select the **Set Stage to Available** check box, the **Apply Template Changes** dialog box asks whether to apply the edits to all active workflow processes based on the template.
	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:
	• 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 or path is re-run (using backward branching or loops) or the task is demoted.
	• If the edits in the workflow template impact an active task, the edits are applied after the task completes and take effect only if the task is re-run.

Editing task	Description
	• If the edits delete the currently active task, the next task is started.
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

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.

• Offline editing prevents users from accessing the workflow template while you edit it. Use this option when you do not want the old version of the template available until your edits are complete.

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 appear 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.

• Online editing allows users to initiate workflows based on the existing version of the workflow template while you edit a copy of the same 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 appear in the **Process Template** list in the **New Process** dialog box.

The **Under Construction** ⚠ symbol appears next to the version being edited. You also have the option of not displaying templates under construction by unchecking the **Show Under Construction Templates** checkbox in the **New Process** dialog box.

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

How process template edits are applied to active processes

When you edit a workflow process template, you can apply the edits to active processes that are based on the template. This action modifies all of the template's active processes simultaneously.

> **Note**
>
> - The **EPM_enable_apply_template_changes** preference value must be set to **OPTIONAL** or **AUTOMATIC**.
>
> - You can set up background processing to apply template edits asynchronously, without pausing your Workflow Designer interaction.

If template edits	They are applied to the active process
Occur at a point that the active process has not reached	When you click **OK** in the **Apply Template Changes** dialog box, with the **Apply template changes to all active workflow processes** check box selected.
Occur at a point that the active process has passed	Only if the edited task or path is rerun in a backward-branching path, is rerun in a loop, or if the task is demoted.
	Otherwise, the edits do not take effect until the next new process that is based on the template.
Affect a task that is in progress	After the task is complete.
	Unless the task is rerun in a backward-branching path, is rerun in a loop, or is demoted, the edits do not take effect until the next new process that is based on the template.
	> **Note**
	> If the edits delete a task that is in progress, the next task is started. The deleted task is removed from the worklists of users who log in subsequently.

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

Enable template edits for active processes

Applying workflow template edits to active workflow processes requires editing the **EPM_enable_apply_template_changes** preference value.

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

2. At the bottom left of the dialog box, click the **Filter** tab. Type **EPM_enable_apply_template_changes** in the **Search by preference name** box.

3. Select the **EPM_enable_apply_template_changes** option and set the value to one of the following:

    The following values are available.

Value	Description
NONE	Default value. Suppresses applying all edits to active processes.
OPTIONAL	Applies workflow template edits to active processes based on each selected workflow template.
AUTOMATIC	Automatically applies edits to a workflow template to all active workflow processes based on the edited template.

Edit a workflow process template

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

2. On the main toolbar, click **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; also, add, delete, redraw, and reset flow paths.

5. (Optional) Add, remove, and modify task attributes by clicking the **Task Attributes** button.

> **Note**
>
> **Process Template** attributes also include the **Filter Condition** option.
>
> • Add a filter condition in the **Task Attributes** panel using the dropdown.
>
> • To remove an applied **Filter Condition**, select the condition text in the dialog box and clear the field.
>
> • Optionally, after editing is complete and the updated **Process Template** is available, click **Purge Templates** in the **Tools** menu to remove the template cache.

6. (Optional) Edit task handlers by clicking the **Task Handlers** button.

7. (Optional) Edit perform signoff teams by clicking the **Task Signoff** button.

8. After you finish your edits, select the **Set Stage to Available** check box.

   The **Stage Change** message states that changing the template stage to **Available** makes the template visible to all users, and asks if you want to continue.

9. To make the template available, click **Yes**.

   The **Apply Template Changes** dialog box opens.
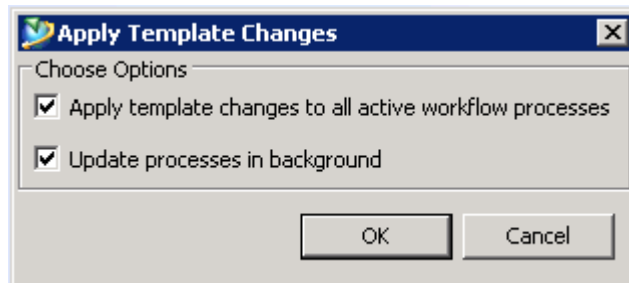
> **Note**
>
> If you click **No** in the **Stage Change** message, the template remains in **Edit** mode for further changes.

10. Click **OK** to save your edits and apply them in the background to the template's active workflow processes.

Caution

- The **EPM_enable_apply_template_changes** preference value must be set to **OPTIONAL** or **AUTOMATIC**.

- Your system must be set up for background processing.

- If you clear the **Apply template changes to all active workflow processes** check box and click **OK**, your edits are saved but are not applied, and the template returns to **Browse** mode.

- The **Update processes in background** check box is selected by default. If you clear the check box, your edits are applied in real time, and Teamcenter may be unavailable until the updates complete. Siemens PLM Software recommends that you leave the check box selected, so that template edits are applied asynchronously, without pausing your Workflow Designer interaction.

Apply process template edits to active processes

After editing a workflow process template, you can make the template available to users and apply the edits to active processes based on the template.

The setting configured in the **EPM_enable_apply_template_changes** option determines how and when the processes are applied. Only the **OPTIONAL** or **AUTOMATIC** values apply edits to active processes.

NONE

Default value that suppresses applying all edits to active processes.

OPTIONAL

Applies workflow template edits to active processes based on each selected workflow template.

It 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.

AUTOMATIC

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

By default, this setting applies the edits in the background. However, background processing requires a four-tier architecture environment. (In a two-tier environment, you can successfully submit requests for asynchronous processing if a four-tier Teamcenter environment is available to accept them.)

Note

Dispatcher must be enabled and configured for asynchronous processing.

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.

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.

   > **Note**
   >
   > Updating the workflow processes in the background is recommended. The **Update processes in background** check box is selected by default.
   >
   > If background processing is not configured and supported at your site, active workflow processes are updated in real time.

   > **Note**
   >
   > If you apply the updates in real time, Teamcenter is unavailable until the updates complete. Although this method is suitable for testing, it is not recommended for updating more than 30 to 50 workflow processes.
   >
   > Update duration depends on the type of edits made to the workflow processes. For example, it takes longer to remove tasks than to add tasks. Edits within tasks (handlers, attributes, etc.) require minimal processing time.

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.

Chapter 4:   Viewing workflow process templates

Viewing templates in the task hierarchy tree or process flow pane

The task hierarchy tree presents a root-level workflow process, along with its tasks and subtasks, in a hierarchical listing.

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.

  > **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 only access.

1. Click **Browse Mode**.

2. Select the task whose attributes you want to view.

3. 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.

4. The Attributes Pane dialog box appears.

   • The **Named ACL** box lists the one assigned to this task.

   • 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.

   • 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 and you are in **Edit** mode.

   • 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 if you are in **Edit** mode.

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

6. If the **Process in Background** check box is selected, the task runs in the background, so the user can continue to work with Teamcenter while the task is executing. If the check box is cleared, the task runs in the foreground, and the user must wait for it to complete.

7. 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.

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 an approval 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 5:   Adding tasks to workflow process templates

Workflow task actions and states

A *task* is a building block in 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 shows how the workflow states and actions interact. States are circled; actions are designated by arrowed lines, indicating the direction the action moves from one state to another.
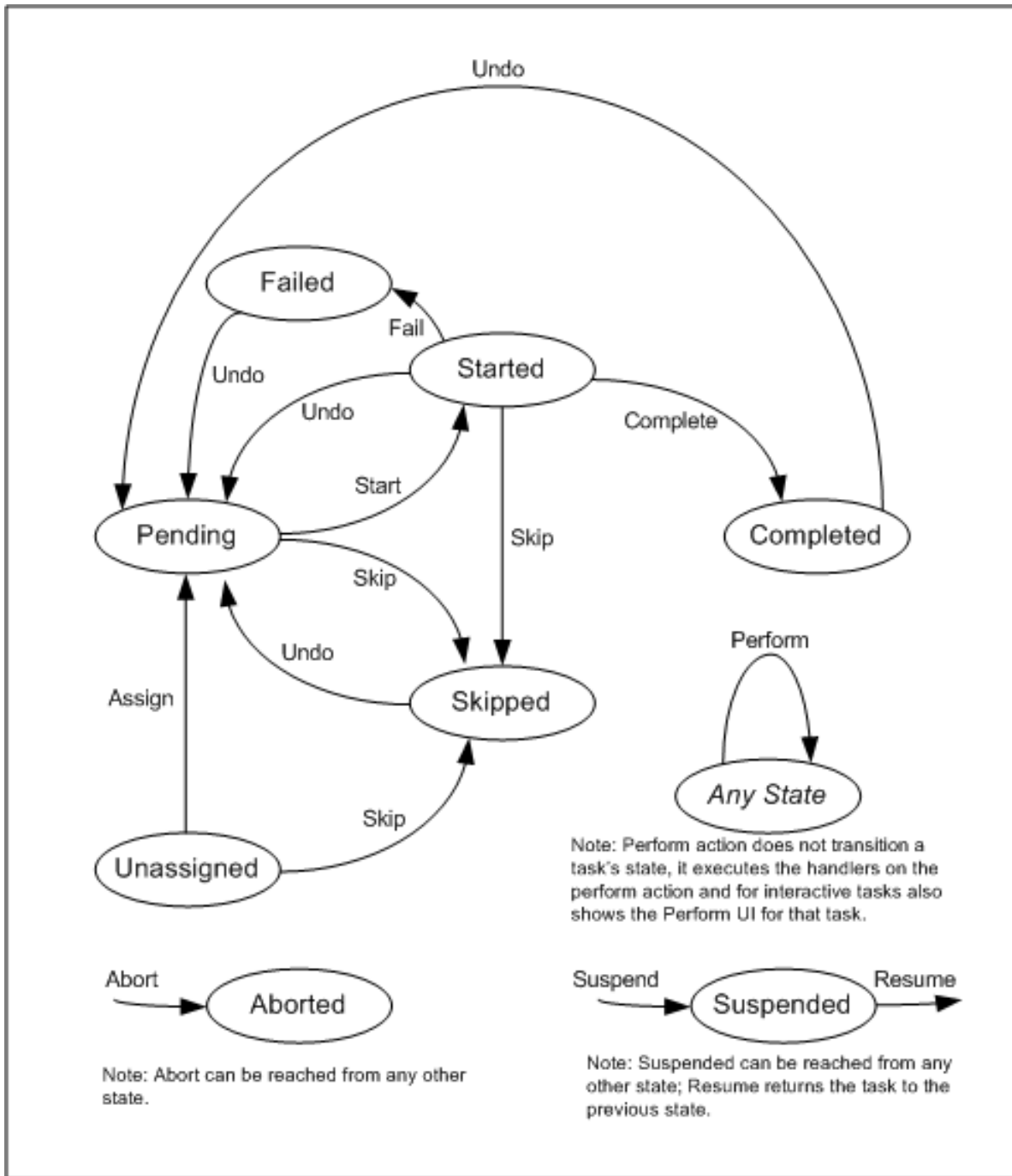
The following table lists the possible beginning states each action can transition from, and the possible ending states each action can transition to:

Action	Beginning state	Ending state	Description
Assign	**Unassigned**	**Pending**	Assigns a task to a responsible party.
Start	**Pending**	**Started**	Starts a task.
Complete	**Started**	**Completed**	Completes a task.

Action	Beginning state	Ending state	Description
Perform	Any state	Any state	Runs any handlers placed on the **Perform** 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.
Suspend	Any state	**Suspended**	Puts a task on hold.
Resume	**Unassigned** **Pending** **Started**	Any state	Resumes a suspended task by returning the task to its previous state.
Skip	**Started** **Completed** **Unassigned** **Pending** **Failed**	**Skipped**	Bypasses the current task and starts the successor task(s).
Undo	**Started** **Completed** **Skipped** **Failed**	**Pending**	Undoes a task by returning the task to the **Pending** state.
Fail	**Started**	**Failed**	Indicates a task configured with a failed path is unsuccessful in fulfilling its requirements.
Abort	Any state	**Aborted**	Cancels a task without attempting to complete it.

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 run 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 run 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.

> **Note**
>
> Use the **INBOX_hide_suspended_tasks** preference to configure how a user's inbox displays tasks that are in either the **Suspend** or **Resume** state.

Task templates

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.

Symbol	Task template	Definition
	Task	The default task template (**EPMTaskTemplate** type). 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.
	Do 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.
		Uses the **EPM-hold** handler, which stops the task from automatically finishing when started.
	Review Task	Uses the **select-signoff-team** and **perform-signoffs** subtasks, each of which has its own dialog box.
		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 finishing and following the appropriate path.
	Add Status Task	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.
	Or 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	Uses the **Acknowledged** and **Not Acknowledged** subtasks, each of which has its own dialog box.
	Condition Task	Branches a workflow according to defined query criteria. Requires that the succeeding task contains an **EPM-check-condition** handler that accepts a Boolean value of either **True** or **False**.
	Route Task	Uses the **Review**, **Acknowledge**, and **Notify** subtasks, each of which has its own dialog box.
	Validate Task	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.

Custom tasks

The **Task** template is the default task template (**EPMTaskTemplate** type). 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.

For example, you may want a task to display a form with fields that users fill in. To set up the custom **Task** template, you use the **Handlers** panel. Your steps can be similar to those in the following example.

- Place the **EPM-create-form** action handler on the **Start** action.
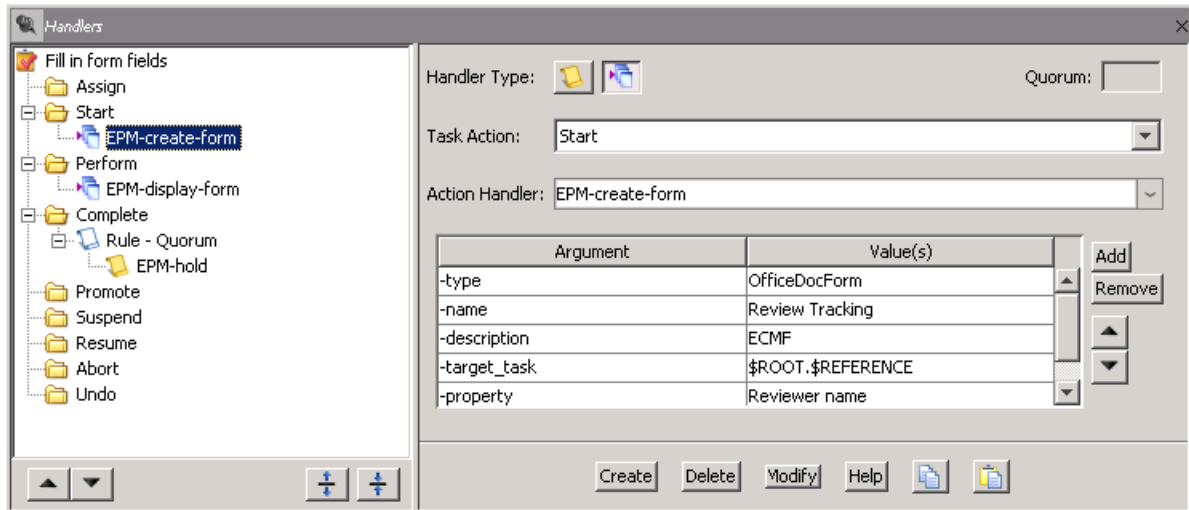
Using the handler arguments, you define the form and attach it to the task. The handler generates the form when the **Start** action is initiated.

- Place the **EPM-display-form** action handler on the **Perform** action.
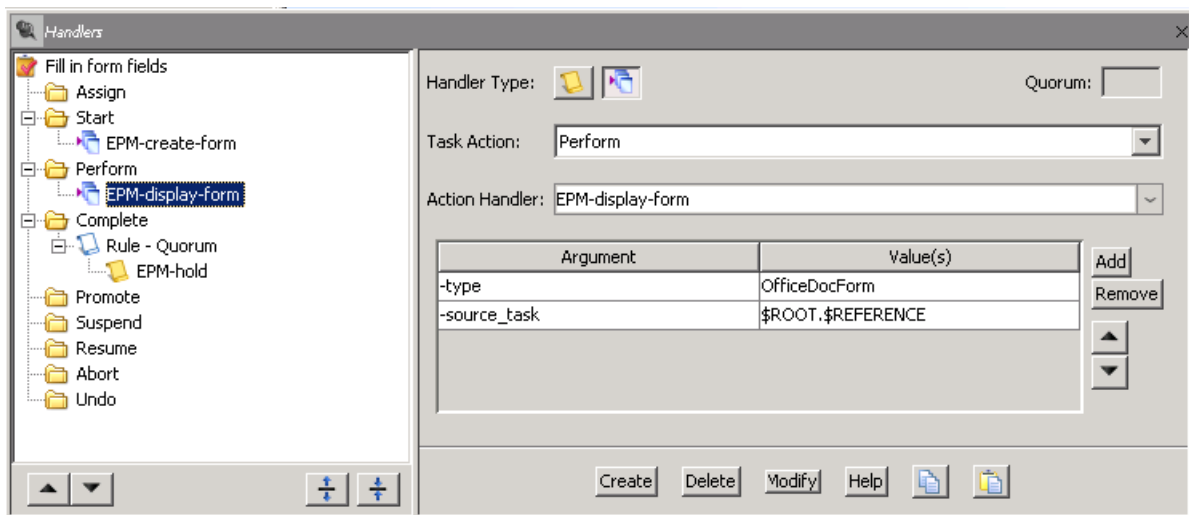
You specify the form type and the task with the handler arguments. The handler displays the form when the **Perform** action is initiated.

- Place the **EPM-hold** rule handler on the **Complete** action.
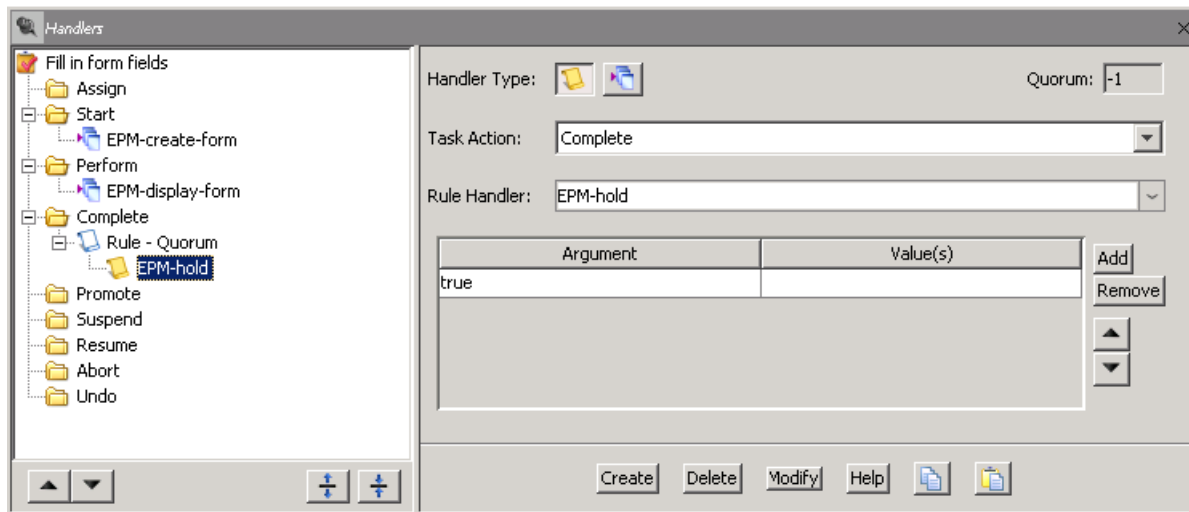
The handler checks the **task_result** property of the task. If that property value is not **Completed**, the handler pauses the task. The user fills in the form fields, and then performs a manual **Complete** action.

> **Tip**
>
> If **Show Task in Process Stage List** is selected in the **Attributes** panel, users can perform the task when the target object is selected or opened in its home location. Click **Display the Task Attributes Panel** at the bottom of the template manager pane to open the **Attributes** panel.

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.

> **Note**
>
> Configure the **WRKFLW_task_complete** preference to use single click functionality for a **Do** task. Setting the value to **true**, the **Complete** value is selected by default for the **Done** button.

If you require user authentication before this **Do** task is performed, add the **EPM-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.

Review tasks

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

> **Note**
>
> Configure the **WRKFLW_task_complete** preference to use single click functionality for a **Select Signoff Team** review task. Setting the value to **true**, the **Ad-hoc Done** check box is selected by default. This also sets the associated **task_result** property to **Complete** by default.

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 selector 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 **EPM-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.

If a user manually promotes a **Review** task that has both an **Approve** path and **Reject** path using the **Actions→Promote** command in My Teamcenter or Workflow Viewer, then they must select which path the workflow process is to follow at that time.

Add Status tasks

Use the **Add Status** task template to create and add a **Release** status to the target objects of the workflow process.

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.

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.

Acknowledge tasks

Use the **Acknowledge** task to define the **Signoff Team** profiles with which a user complies to assign acknowledgment 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.
>
> - Signoff profiles are unavailable for the **Acknowledge** task if it is a subtask within the **Route** task template. The **Route** task does not function properly if signoff profiles are defined for the subtasks. The **Route** task template is designed to be used as an electronic routing sheet, and the workflow process initiator assigns specific signoff members.

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 **No Decision**. 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 **EPM-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.

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.

- 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.

> **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).

- Create a query against the task (automatic only).

- Create a query against subprocesses (automatic only).

  If there are multiple subprocesses, a query runs on the associated subprocesses and the results are used to branch accordingly. The query is typically configured to look at the root task's **result** attribute for all the subprocesses.

  If there is only one subprocess and it is configured to set the result on the **Condition** task, no query is needed, and the workflow follows the branch based on the result.

- Configure the task result from the manual **Condition** task's dialog box.

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.

Task	Description
Automatic **Condition** task	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.

Task	Description
Manual **Condition** task	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.

Caution

To ensure desired results, condition tasks that run queries in workflows should always have at least one target object when a condition query is run against workflow targets.

- When a condition task runs a condition query against workflow targets, the system searches the database for that query class and filters the results based on the workflow target objects.

- Because handlers can move objects between targets and references in a workflow, the workflow may have objects in the references folder, but no objects in the targets folder. The condition query will not search in the database if the workflow does not have any targets. This will set a false path of the condition task.

Route tasks

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

Note

Configure the **WRKFLW_task_complete** preference to use single click functionality for a **Select Signoff Team** route task. Setting the value to **true**, the **Ad-hoc Done** check box is selected by default. This also sets the associated **task_result** property to **Complete** by default.

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 running.
>
> - Signoff profiles are unavailable for the **Acknowledge** subtask within the **Route** task template. The **Route** task does not function properly if a signoff profile is defined for the **Acknowledge** subtask. The **Route** task template is designed to be used as an electronic routing sheet, and the workflow process initiator assigns specific signoff members.

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 **EPM-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.

If a user manually promotes a **Route** task that has both an **Approve** path and **Reject** path using the **Actions→Promote** command in My Teamcenter or Workflow Viewer, then they must select which path the workflow process is to follow at that time.

You can also route or reassign tasks to another user from your inbox, in the same manner as selecting a signoff task. You can select options and designate specific users to notify, acknowledge, or review tasks.

To set up the display of the **Task View** pane, configure the **WORKFLOW_new_route_task_panel** preference. Display choices are **ON** for list box view or **OFF** for option button view.

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).

Adding tasks to a process template

Create your own specific workflow requirements with 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.

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.

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.

Specify user actions with 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.

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.

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.

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.

Require users to look at targets with 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.

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.

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.

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.

     > **Note**
     >
     > You can change the names of the **select-signoff-team** and **perform-signoffs** subtasks. For example, you can rename the subtasks to specify their parent task or the current step in the process (such as **select-design-signoff-team**).

   • 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.

10. Select and enter the number or percentage of reviewers required to satisfy an approval quorum.

    You can designate the number or percentage of reviewers required for the approval 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.

    > **Note**
    >
    > If you set the **WRKFLW_allow_wait_for_undecided_override** preference to *False*, the **Wait for Undecided Reviewers** option is hidden.

11. 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.

Attach a status to targets with 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.

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.

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.

Continue the workflow with 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.

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.

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.

Inform users of a workflow's progress with 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.

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.

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.

9. Define a signoff profile.

> **Warning**
>
> Signoff profiles are unavailable for the **Acknowledge** task if it is a subtask within the **Route** task template. The **Route** task does not function properly if signoff profiles are defined for the subtasks. The **Route** task template is designed to be used as an electronic routing sheet, and the workflow process initiator assigns specific signoff members.

a. Double-click the **Acknowledge** task in the task hierarchy tree.

   The task expands, listing the **select-signoff-team** and **perform-signoffs** subtasks.

   > **Note**
   >
   > You can change the names of the **select-signoff-team** and **perform-signoffs** subtasks. For example, you can rename the subtasks to specify their parent task or the current step in the process (such as **select-design-signoff-team**).

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 an approval quorum.

You can designate the number or percentage of reviewers required for the approval 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.

Branching a workflow with a Condition task

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 **EPM-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 moves to the **Condition** task's false path, proceeding to a **Do** task containing instructions to complete a **Request for Change** form.

You can also query multiple subprocesses, and the results are used to branch accordingly. The query is typically configured to look at the root task's **result** attribute for all the subprocesses.

> **Example**
>
> For example, use a **Condition Task** template to create a conditional task for a change request object that initiates two subprocesses: one that checks to see if a change specialist has been assigned and one that checks if an analyst has been assigned. The task is configured to check if all subprocesses return **true**.
>
> - If the query results are true for both subprocesses, the workflow continues along the **Condition** task's true path, proceeding to a **Do** task containing instructions for the assigned users to identify impacted items and propose solutions.
>
> - If the query results are not true for both subprocesses, the workflow moves to the **Condition** task's false path, proceeding to a **Do** task to assign a user to the change specialist or analyst role.

If there is only one subprocess and it is configured to set the result on the **Condition** task, no query is needed, and the workflow follows the branch based on the result.

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**.

> **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. Right-click the new **Condition** task and choose **Task Properties**.

7. 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 the **Condition Query** button.

        The **Condition Query Dialog** 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.

    c.   Select **Target**, **Task**, or **Sub-Process** to determine if the query is performed on the workflow process attachments, the task to which the query is attached, or the subprocesses that the **Condition** task depends on, respectively.

        When **Target** is selected, the **Include Replica Proposed Targets** is active. Select the **Include Replica Proposed Targets** to include targets on the remote workflow task in the search.

    d.   Select **All**, **Any**, or **None** to determine whether all, any, or none of the target attachments or subprocesses must meet the query criteria to set the **Condition** task's result to **True**. If you clicked **Task**, these buttons are unavailable.

    e.   Click **OK** or **Apply** 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.

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

9.   (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.

10. 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.

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.

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 backward-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**.

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.

Distribute targets to users with 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.

> **Note**
>
> Signoff profiles are unavailable for the **Acknowledge** subtask within the **Route** task template. The **Route** task does not function properly if a signoff profile is defined for the **Acknowledge** subtask. The **Route** task template is designed to be used as an electronic routing sheet, and the workflow process initiator assigns specific signoff members.

6. Explicitly link the task to the predecessor tasks.

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.

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.

9. (Optional) You can change the names of the **Review**, **Acknowledge**, their **select-signoff-team** and **perform-signoffs** subtasks, and **Notify** subtasks. For example, you can rename the subtasks to specify their parent task or the current step in the process (such as **select-design-signoff-team** or **Design Review**).

Check for errors with a Validate task

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*.

   > **Note**
   >
   > To access the *Integration Toolkit Function Reference*, install the Teamcenter developer references when you , or go to the Global Technical Access Center (GTAC):
   >
   >   https://support.industrysoftware.automation.siemens.com/docs/teamcenter/

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, which contain the majority of workflow 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**.

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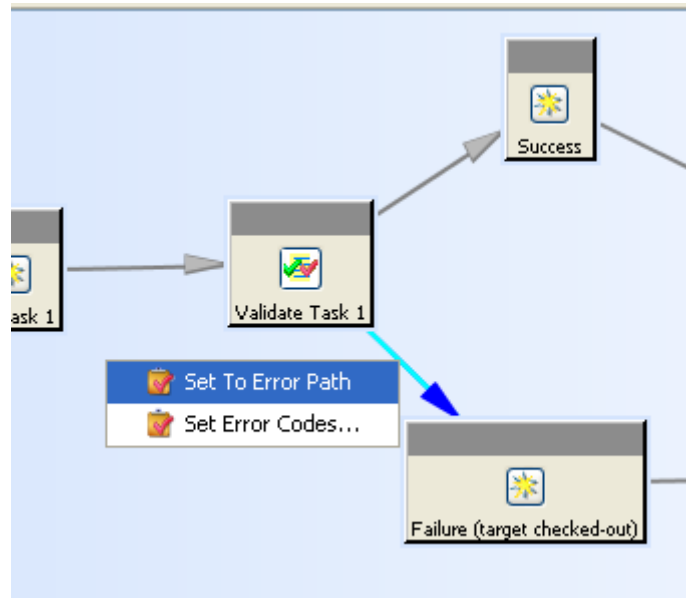
1. Right-click the path you want to configure as a failure path.

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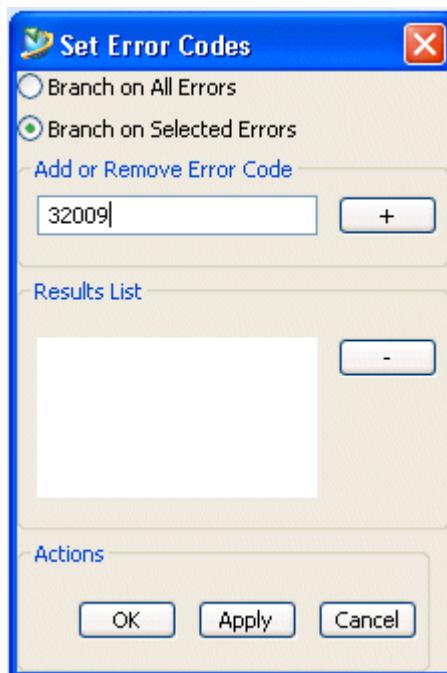
5.   Click **Add** [+] to add this error to the **Results List**.

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.
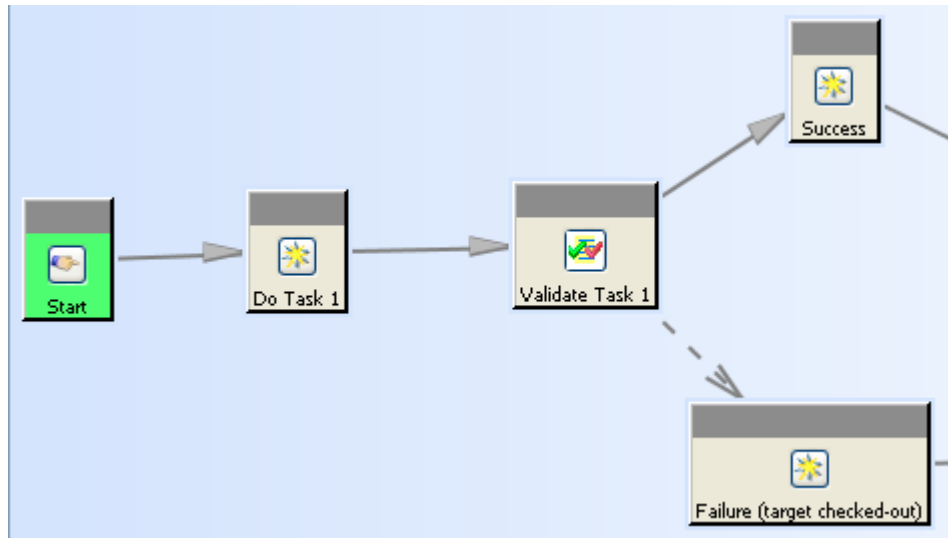
Insert and configure a Validate task

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.

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.

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.

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 **EPM-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.

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 **EPM-assert-targets-checked-in** 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-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.

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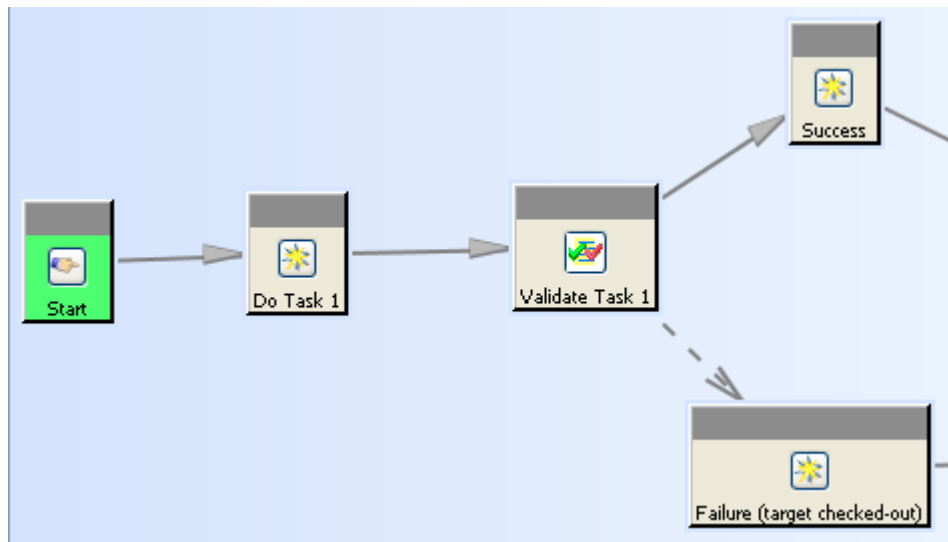
19. Insert another **Do** task after the **Failure (target checked-out)** task.

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.

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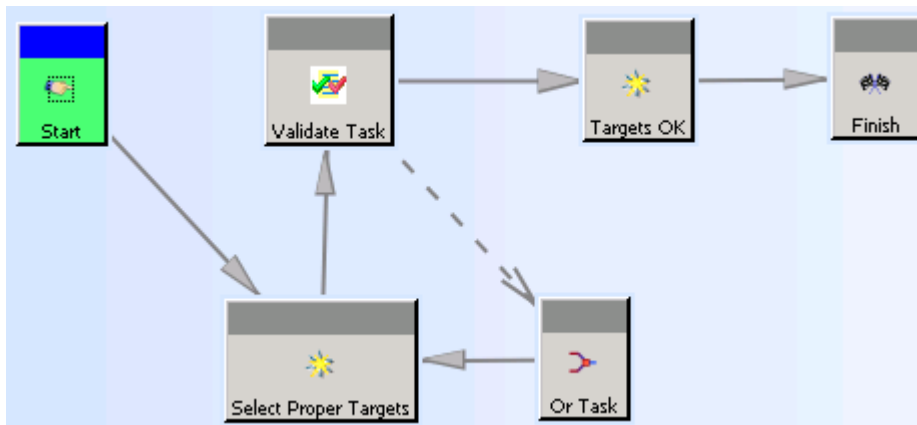
The following procedure illustrates how to configure the workflow in this example:

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 below and 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.

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 **Do** task you create.

9. Insert an **Or** task to the right of the **Select Proper Targets** task.

10. Draw a failure path from the **Validate** task to the **Or** task by placing the cursor in the body of the **Validate** task and dragging it to the body of the **Or** task.

    When proper targets are not selected, the workflow branches backward to the **Or** task and then to the **Select Proper Targets** task, prompting the user to select proper targets.

11. 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.

12. 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.

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

14. 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.

15. Draw a success path from the **Or** task to the **Select Proper Targets** task and another one from there to the **Validate** task.

16. 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.

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

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

19. 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.

20. Draw a success path from 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.
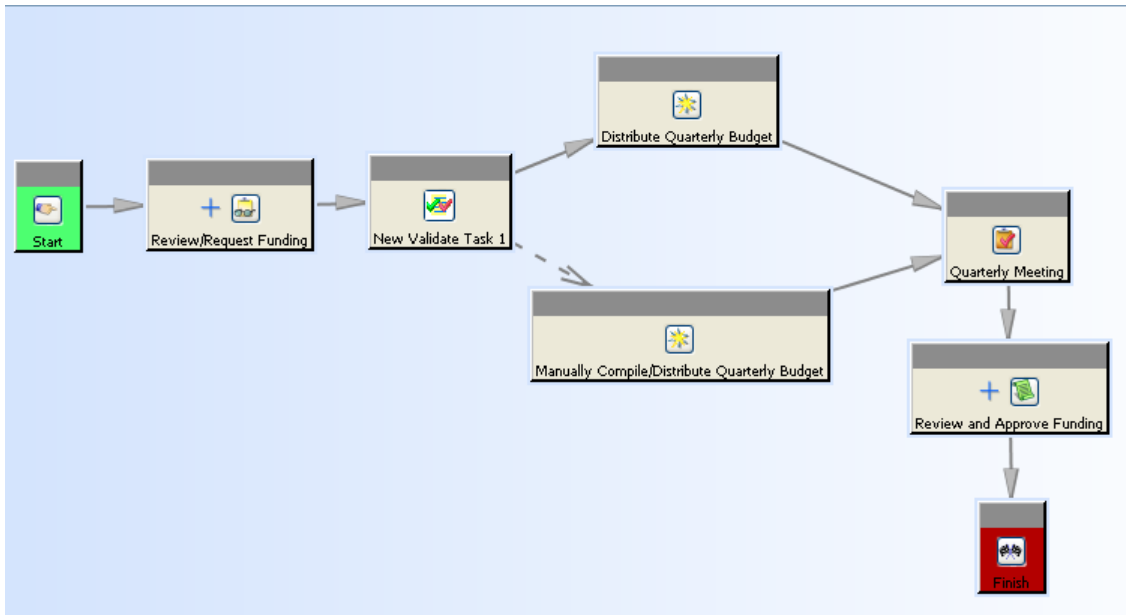
The following procedure illustrates how to configure the workflow in this example:

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.

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 ![handlers icon].

       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** ![action handler icon] 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 ![do task icon] above and to the right of the **Validate** task. This is the first of the two successor tasks uses 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 ![do task icon] 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**).

19. Click **Add** ![+ button] to add this error 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.
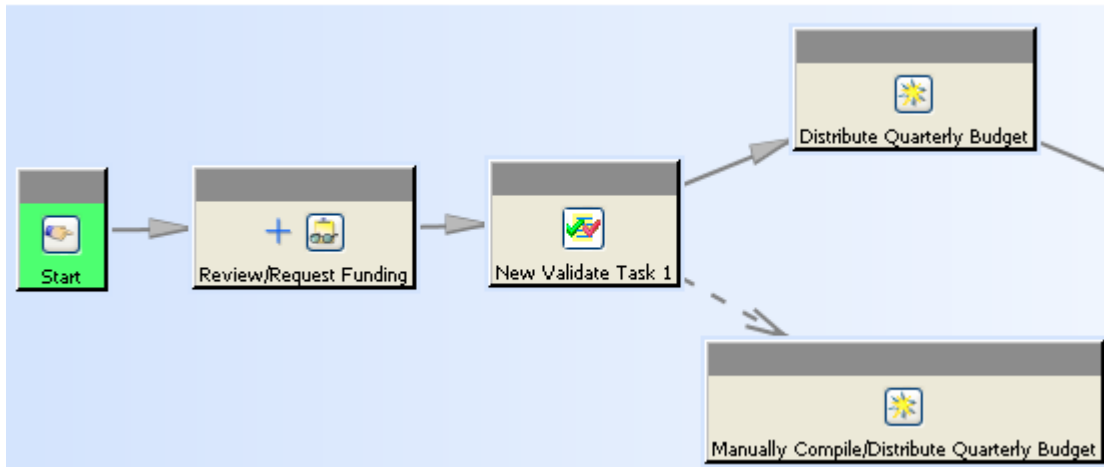
21. Reconcile the success and failure paths by inserting a generic task and linking it to the **Distribute Quarterly Budget** task (on the success path) and the **Manually Compile/Distribute Quarterly Budget** task (on the failure path).

    • 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.

Failure criteria you specified	Error thrown (if any)	Task behavior
Fail if any error	Any error	Failure path is followed.
Fail if error on error list occurs	Error on error list	Failure path is followed.
Fail if error on error list occurs	Error *not* on error list	Workflow process halts. Task remains in **Started** state and an error appears.
No failure path configured	Any error	Workflow process stops. Task remains in **Started** state and an error appears.
Regardless of whether failure path was configured, and whether errors occurred	No errors occur	Success path followed. If no success path was configured, workflow process stops.

Automatically reassign tasks for inactive users

Workflow tasks can be redirected around inactive users, for example, users who are out of the office. There are two preferences that can be set: **WRKFLW_admin_for_inactive_user** and **WRKFLW_error_on_invalid_dynamic_participant**.

When a task or signoff is assigned to a user, Workflow checks to see if the user has an out-of-office turned on. If the user is out of the office or otherwise inactive, Workflow reassigns the task to the resource pool corresponding to the user's group and role.

In the case of an adhoc signoff, the signoff task is reassigned to the user's group resource pool.

When a task is reassigned from an inactive participant, an email indicating the task reassignment is sent to a mailing list defined in the **WRKFLW_admin_for_inactive_user** preference. If the preference is not set, the email is sent to the process owner.

Insert a task into a template

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

2. On the toolbar, click one of the task buttons.

Button	Task	Definition
	Do 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.
		Uses the **EPM-hold** handler, which stops the task from automatically completing when started.
	Acknowledge Task	Uses the **Acknowledged** and **Not Acknowledged** subtasks, each of which has its own dialog box.
	Review Task	Uses the **select-signoff-team** and **perform-signoffs** subtasks, each of which has its own dialog box.
		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.
	Route Task	Uses the **Review**, **Acknowledge**, and **Notify** subtasks, each of which has its own dialog box.
	Task	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 **EPMTask** template.
	Condition Task	Branches a workflow according to defined query criteria. Requires that the succeeding task contains a **EPM-check-condition** handler that accepts a Boolean value of either **True** or **False**.
	Validate Task	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.
	Add Status Task	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.

Button	Task	Definition
	Or 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.

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

    A new task appears with the default name of **New** *task_type***Task** **#**, where *task_type* is the kind of task you selected and # 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 users must perform for this task.

6. Explicitly link the task to the predecessor tasks.

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.

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.

9. Follow the additional steps listed based on the task you inserted.

Task	Additional steps
Do Task	None.
Acknowledge Task or **Review Task**	For more information about completing the insertion process, see step 10.
Route Task	None. **Warning** The **Route** task is designed to be used as an electronic routing sheet. The workflow process initiator assigns specific signoff members. Signoff profiles for the **Review** subtask should not be defined within this task. Signoff profiles are unavailable for the **Acknowledge** subtask. The task does not function properly if signoff profiles are defined at this stage.

Task	Additional steps
Task	None.
Condition Task	Additional steps are required for the **Condition** task.
Validate Task	Additional configuration steps are required for the **Validate** task.
Add Status Task	None.
Or Task	None.

10. For an **Acknowledge Task** or **Review Task** :

a. Define a signoff profile.

    A. Double-click the 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 then 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.

    D. 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.

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

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

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

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

b. 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.

c. 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.

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.

   > **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.

Localize task names

To localize workflow task names for a workflow process:

1. In the Business Modeler IDE, set the **Localizable** constant to **true** on the **EPMTaskTemplate** business object **template_name** property.

2. In Workflow Designer in the rich client:

    a. Select a process template.

    b. Display the **Properties** dialog.

    c. Provide the localized value for the **template_name** property.

3. Do this for each task in the workflow template.

4. Create the workflow process using the workflow template.

Chapter 6: Linking tasks in a workflow process template

Explicit and assumed links

A link establishes the sequence by which peer-level tasks are run, 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.

Link tasks manually

Each workflow requires an execution sequence. Arrows represent paths between tasks, whether assumed or explicit. The arrow identifies the sequence from a starting task to an ending task. Tasks must be completed in sequence.

Creating a link manually produces an *explicit task* and should be linked immediately after inserting tasks. Saving the workflow process, or switching away from Workflow Designer before manually linking tasks, prompts Teamcenter to automatically insert *assumed links*.

> **Tip**
>
> Always explicitly link your tasks to ensure predictable results.

Each link consists of a predecessor task and a successor task.

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

2. Insert tasks.

3. Click a predecessor task node.

> **Note**
>
> Do not click the title bar of the task node. Clicking the title bar begins a drag process.

4. Hold the mouse button and drag the cursor to a successor task.

    A link arrow follows the cursor. When the cursor moves over a task node, the node is highlighted.

5. Release the mouse button.

    A link arrow connects the predecessor and successor nodes creating an explicit task.

Delete 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.

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.

Task	Failure option
Do	**Set to Unable to Complete**
Review	**Set to Reject**
Route	**Set to Reject**
Condition	**Set to Unable to Complete**
Validate	**Set to Error Path**
EPM	**Set to Unable to Complete**

This example shows the options for an existing **Condition** task failure path.
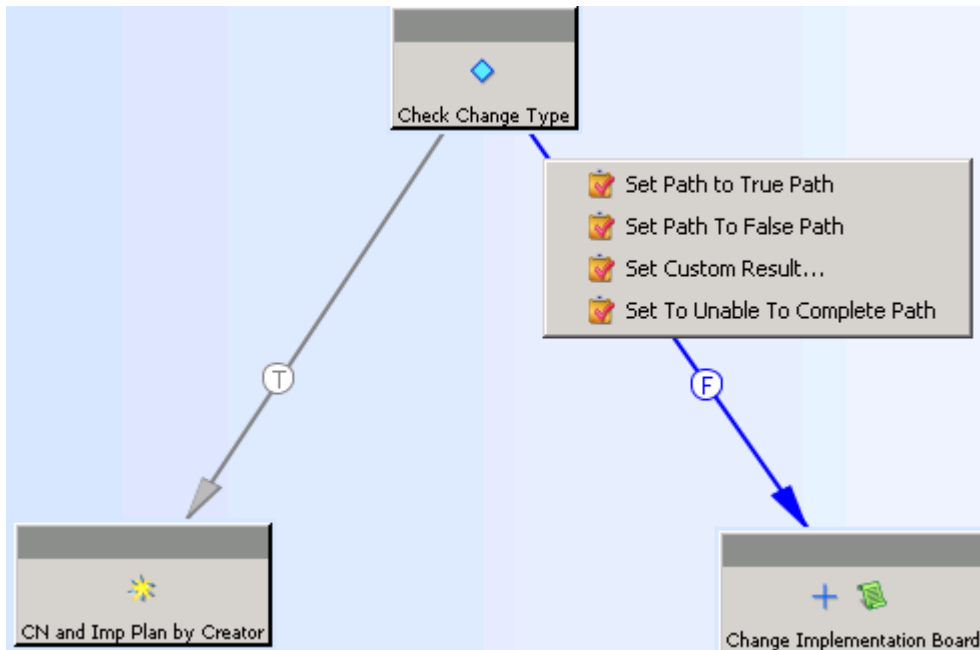
Developing workflow process templates with backwards branches

You might need to construct a workflow process template that branches backwards, in other words, one that links directly or indirectly to a task earlier in the flow that has already been performed.

In this example, **Task 3** branches backwards to **Task 1**, which was already performed.
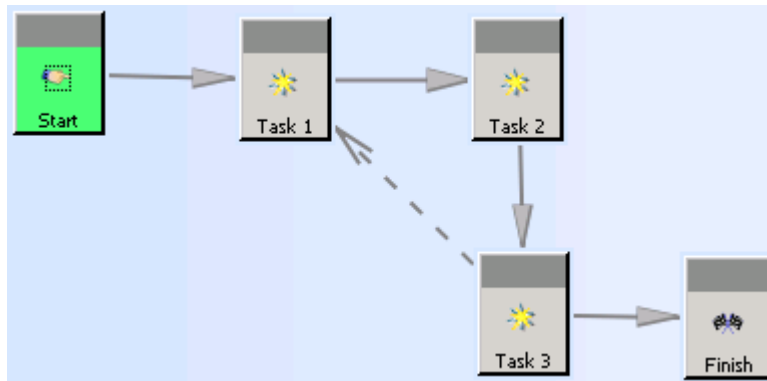
The way Teamcenter processes tasks repeated in the backwards-branching loop depends on the version of Teamcenter you are using.

• In the legacy versions (Teamcenter versions 9.1 and earlier, 8.3.3.2 and earlier), the workflow automatically determines if the repeated task could be restarted.

• In later versions (Teamcenter versions 10.1 and later, 9.1.1 and later, 8.3.3.3 and later), you must design the workflow with **Or** tasks or custom tasks that act as **And** tasks to determine the behavior of the repeated tasks shown in the following example.
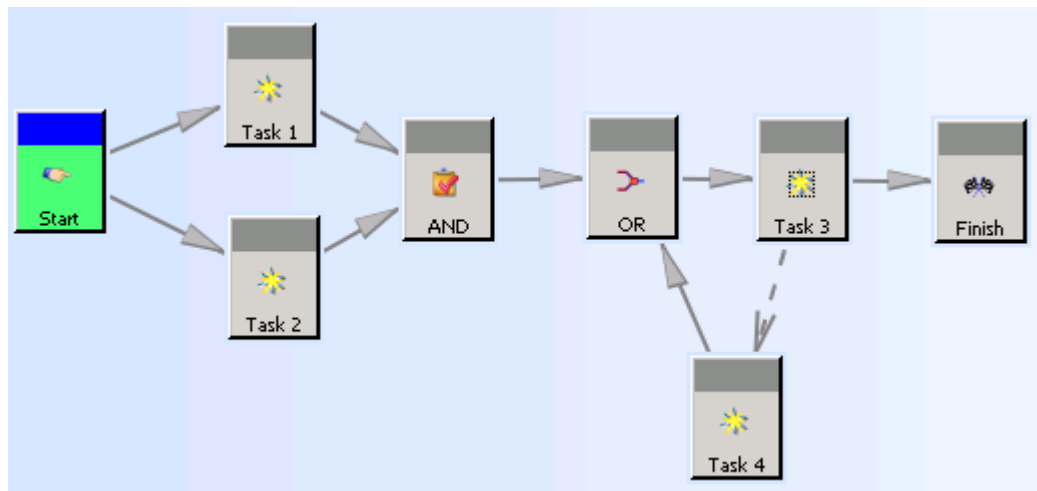
In this case, because of the custom **And** task, both **Task 1** and **Task 2** must complete before the workflow moves to **Task 3**. If **Task 3** is rejected, the workflow moves to **Task 4** and then with the **Or** task moves back to **Task 3** again.

Newly created templates have the new behavior even if based on a legacy template that has the legacy behavior.

Converting legacy backwards branching templates to the new behavior

If the workflow process template was created in a legacy version, it retains the legacy behavior unless the user converts it to the new behavior.

Note

If the user converts the template, it might need to be redesigned to produce the required workflow correctly. For example, if the legacy template looks like the following:
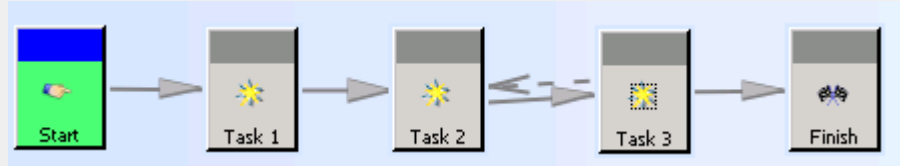
And the user converts this template to the new style, any process based on this template stalls because **Task 2** is waiting for **Task 3** to complete and **Task 3** is waiting for **Task 2** to complete. To complete successfully using the new style, the template should be redesigned to look like the following:
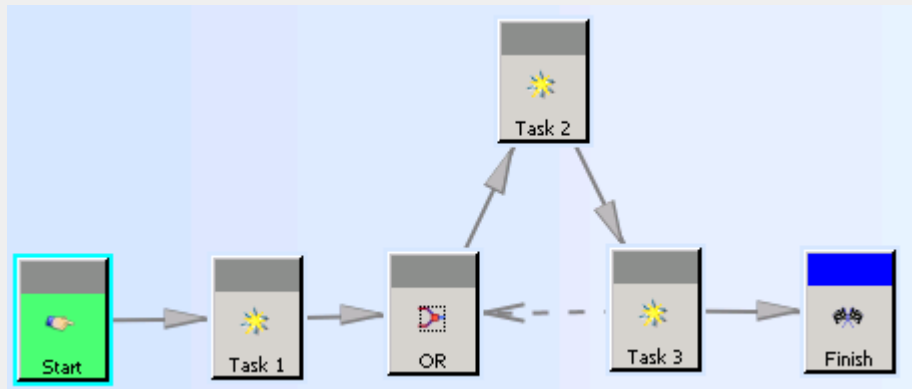
The introduction of the **Or** task allows the process to complete because **Or** tasks do not require all predecessor tasks to complete.

To convert to the new behavior, the **WRKFLW_convert_backward_path_representation** preference must be set to **true** so the option to convert is displayed when the legacy template is taken offline.

When the legacy template is taken offline, the user can select the **Convert Backward Branches to New Style** check box to convert the template or clear the check box to keep it in the legacy style.

Note

- If the user converts the template to the new style, it cannot be converted back to the legacy style.

- An imported legacy template retains its legacy behavior until it is taken offline and converted by a user.

Siemens PLM Software encourages you to convert your templates. A future version will automatically convert the templates for you.

Moving to a previous task after Review or Route task is rejected

When designing a workflow process that moves back to a previous task if a **Review** or **Route** task is rejected, you must determine if the workflow should be demoted to a previous task or follow a failure path.

> **Note**
>
> Do not use a failure path together with the **EPM-demote-on-reject** and **EPM-demote** handlers—use either the failure path or the handlers.

- **EPM-demote-on-reject** and **EPM-demote** handlers

  Use these handlers to move the workflow backwards to a previous task.
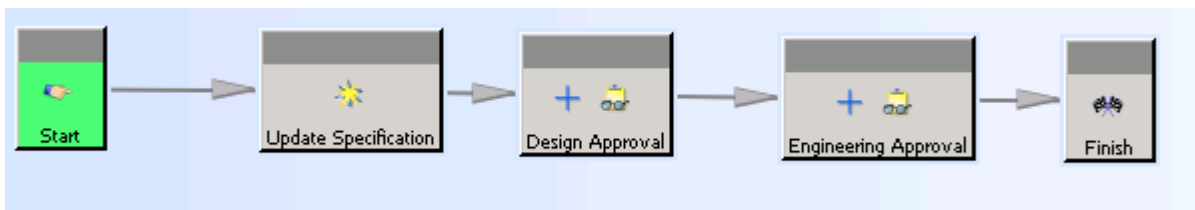
  o If the **Design Approval Review** task is rejected, the **EPM-demote-on-reject** handler with the **-num_rejections=-1** argument placed on the **Perform** action of its **perform-signoffs** subtask demotes the task when a quorum cannot be reached. The **EPM-demote** handler with no arguments on the **Undo** action of the **Review** task demotes the workflow back to the **Update Specification Do** task. The **Do** task must be completed again before moving the workflow forward again.
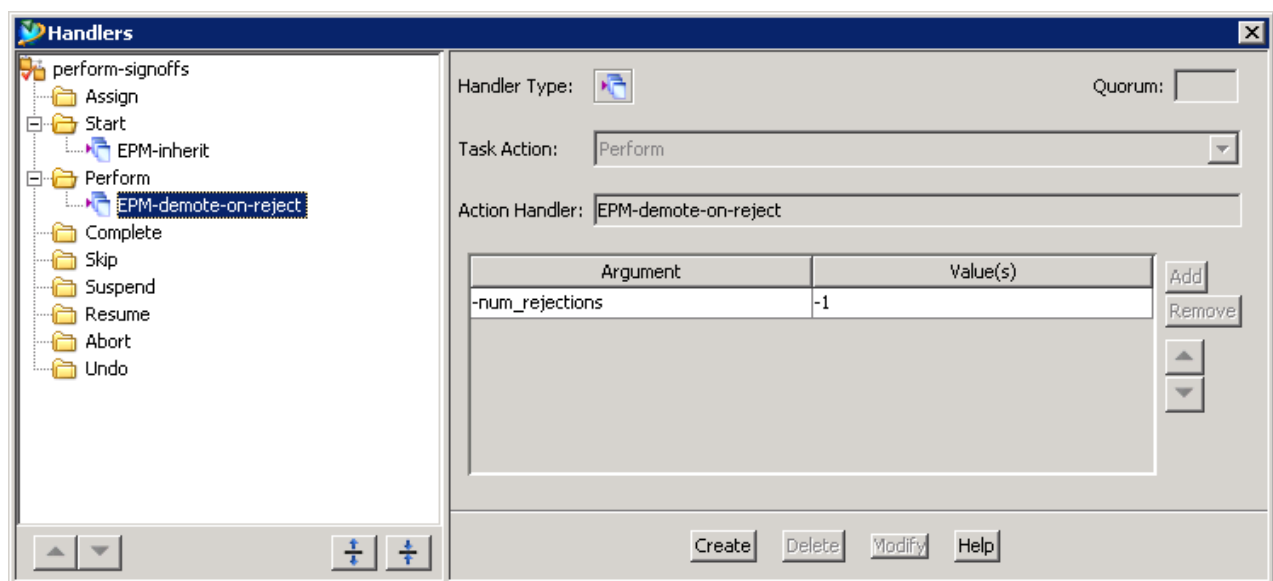
  o If the **Engineering Approval Review** task is rejected, the **EPM-demote-on-reject** handler with the **-num_rejections=-1** argument placed on the **Perform** action of its **perform-signoffs** subtask demotes the task when a quorum cannot be reached. The **EPM-demote** handler with the **-task_target=Update Specification** argument on the **Undo** action of the **Review**

task demotes the workflow back to the **Update Specification Do** task. The **Do** task must be completed again before moving the workflow forward again.
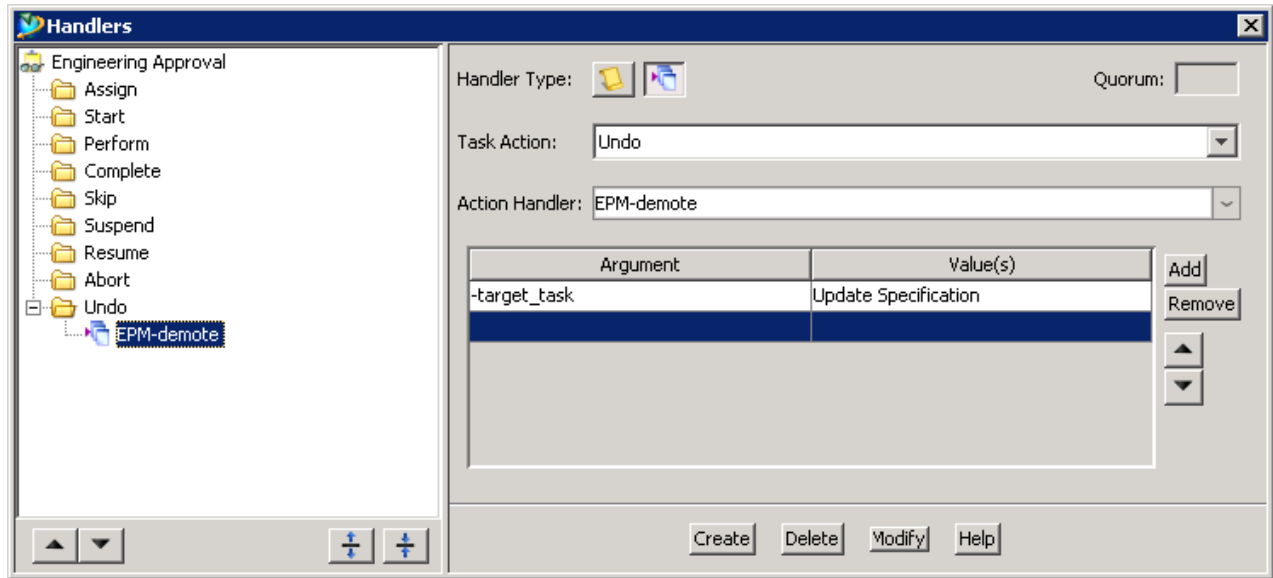
o   Once both **Review** tasks are approved, the workflow completes.

- Failure path

  You can design a workflow process so that if any of several **Review** tasks are rejected, the workflow is sent back to the same point. However, you must insert an **Or** task to receive the failure path from the rejected tasks. For example:
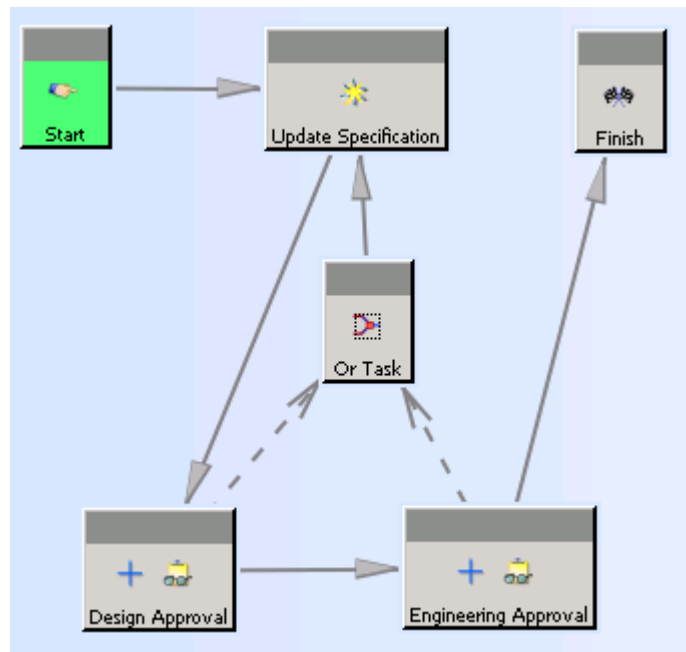
In the following *invalid* configuration without an **Or** task, the **Update Specification Do** task awaits rejection from all **Review** tasks, which stalls the workflow.
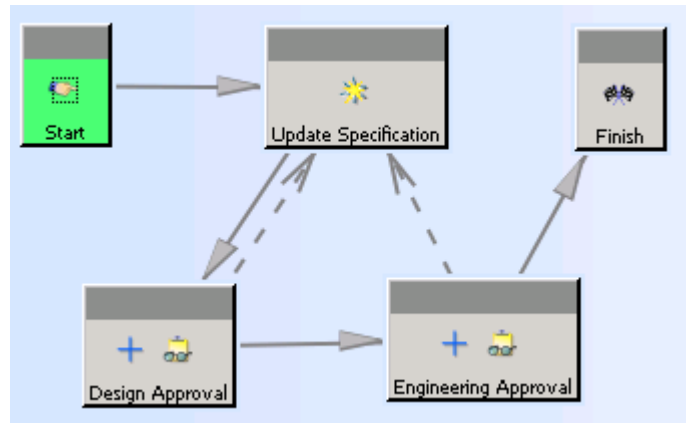
If a **Review** or **Route** task is rejected and there is no failure path or **EPM-demote-on-reject** handler attached, the task remains in the **Started** state and the workflow does not move forward until a user intervenes.

Chapter 7:   Modifying task behavior

Using attributes and handlers to modify tasks

Modify task behavior within a workflow process template using attributes and handlers.

- **Attributes**

    Allows you to set requirements and/or restrictions on a task. Possible task attributes are:

    o   Named ACL
    o   Template name
    o   Signoff quorum
    o   Release status
    o   Icons

- **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:

    o   Set protections
    o   Assign reviewers
    o   Demote a task
    o   Perform a signoff
    o   Change a status

    There are two kinds of handlers:

    o   **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.

    o   **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.

For more information on handlers refer to What are workflow handlers.

> **Caution**
>
> You cannot modify shipped task templates by changing the structure of the subtasks. Modifying the original task templates shipped with Teamcenter in this manner affects all subsequent workflows. This may affect the database during a future upgrade when the upgrade script attempts to update the task templates.

Instead of modifying templates, create custom tasks to perform the desired actions. You can extend shipped task templates using attributes and handlers.

> **Example**
>
> You want to add a task to your process that notifies users of a deadline inherent to the process. You know that the **Route** task shipped with Teamcenter contains within it a **Notify** subtask. You would not strip out the **Acknowledge** and **Review** subtasks within the **Route** task. Rather, you would create a new task, for example, **NotifyDeadline**, and add the **EPM-notify** action handler to the task.

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 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.

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.

   •   for the Condition task.

      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.

      > **Note**
      >
      > The **Task Manager** daemon must be installed to see color-coding relating to task completion.

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.
        >
        > o   The **Show Task in Process Stage List** displays the task in the **Process Stage List** property for the target object.
        >
        > o   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 **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. Select **Require Task Confirmation on Complete** to require users to confirm a selected interactive task is completed in Active Workspace.

    Selecting a root task requires completion confirmation on all child tasks.

    > **Note**
    >
    > The confirmation of completion dialog displays with a task-specific message in Active Workspace.

11. Select **Process in Background** to run the task in the background so the user can continue to work with Teamcenter while the task is executing.

    Clear **Process in Background** to run the task in the foreground. The user must wait for it to complete.

12. 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.

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**.

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 approval 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.

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. Select a handler from the **Action Handler** or **Rule Handler** list.

3. 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**.

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. 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. 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**.

3. 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.

4. 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.

5. 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.

6.   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.

Configuring rule quorums

You can include one or more rule handlers under a **Rule** container.
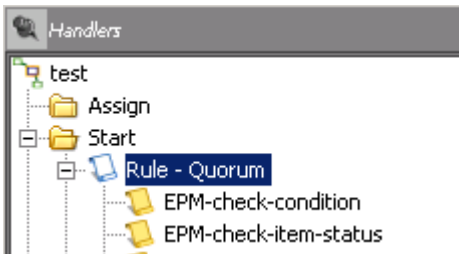
You can then set the **Rule** container **Quorum** value to specify whether one rule, all rules, or a number of rules must be satisfied for the task to progress.
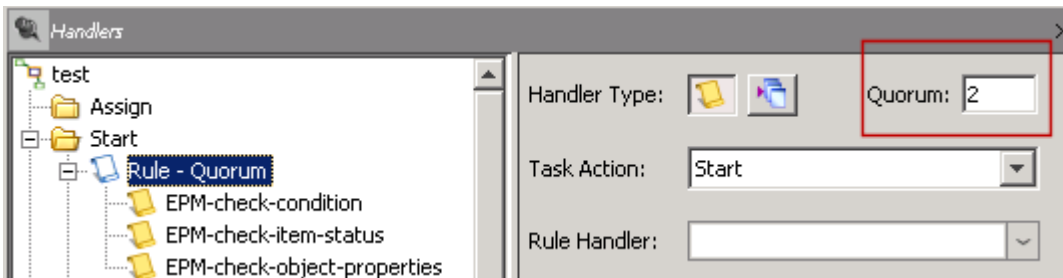
For example, if a **Rule** container has five rule handlers, but you only require two of them to pass, you can set the rule handler quorum value to **2**.

The **Rule** container label changes automatically based on the number of handlers in the container and the **Quorum** value, which specifies the number of handlers that must be satisfied for the task to proceed.

The **Rule** container label can be:

•   **Rule - Quorum**:displayed when the **Quorum** value is set to:

    o   **-1**, which is equivalent to **All**. In this case, every rule must pass to meet the quorum.

    o   A number greater than **1**, but less than the number of rules in the **Rule** container.

•   **Rule - OR**:displayed when the **Quorum** value is set to **1** and there are two or more rules in the **Rule** container.

- **Rule - AND**: displayed when the quorum number is equal to the number of rules in the **Rule** container.

> **Note**
>
> If you set the **WRKFLW_allow_quorum_override** value to *False* the user cannot modify the quorum value. This will remove the **Numeric** or **Percent** options, as well.

Examples

- Condition to satisfy: Every rule in the **Rule** container must be satisfied for the workflow to continue.

  o **Quorum** value: **-1**, which is equivalent to All. In this case, every rule must be satisfied to meet the quorum.

  o **Rule** container label: **Rule - Quorum**.

In this example, there are three rule handlers in the **Rule** container and the **Quorum** value is **-1**, so all rules must be satisfied.
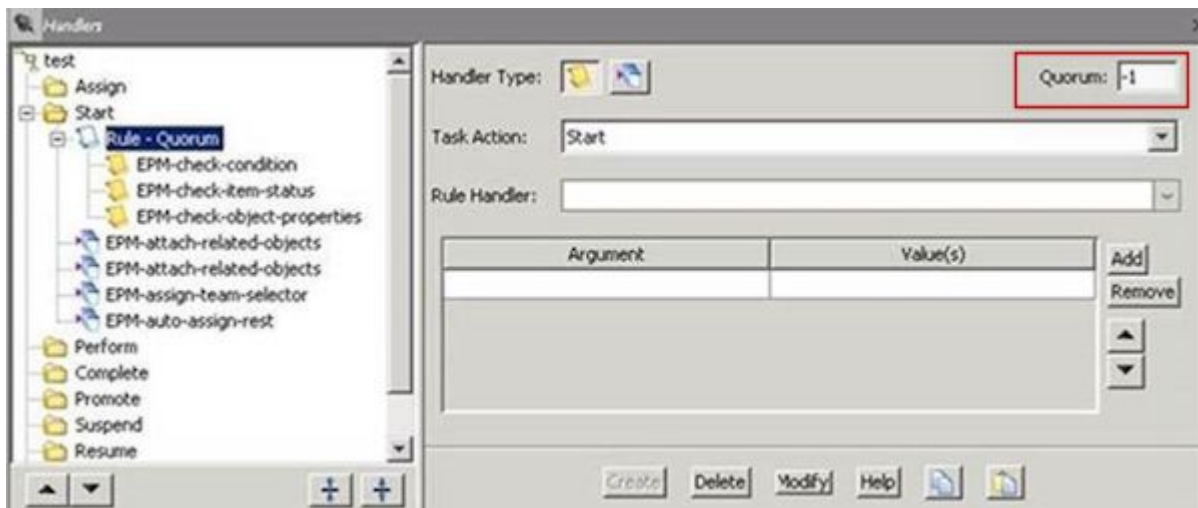
- Condition to satisfy: More than one rule but less than the number of rules in the **Rule** container must be satisfied for the workflow to continue.

  o **Quorum** value: Greater than one but less than the total number of rules.

  o **Rule** container label: **Rule - Quorum**.

In this example, the **Quorum** value is **2**, so if any two of the three rules is satisfied, the workflow can continue.
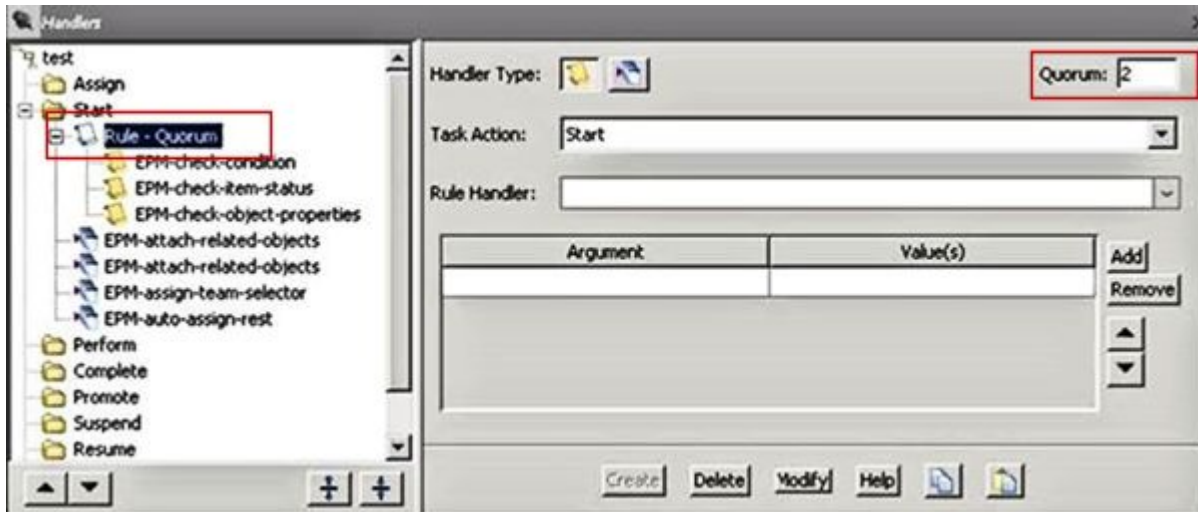
- Condition to satisfy: if any of the rules is satisfied, the workflow can continue.

    o   **Quorum** value: **1**, and there are several rules in the **Rule** container.

    o   **Rule** container label: **Rule - OR**.

In this example, the **Quorum** value is **1**, so if any of the three rules is satisfied, the workflow can continue.
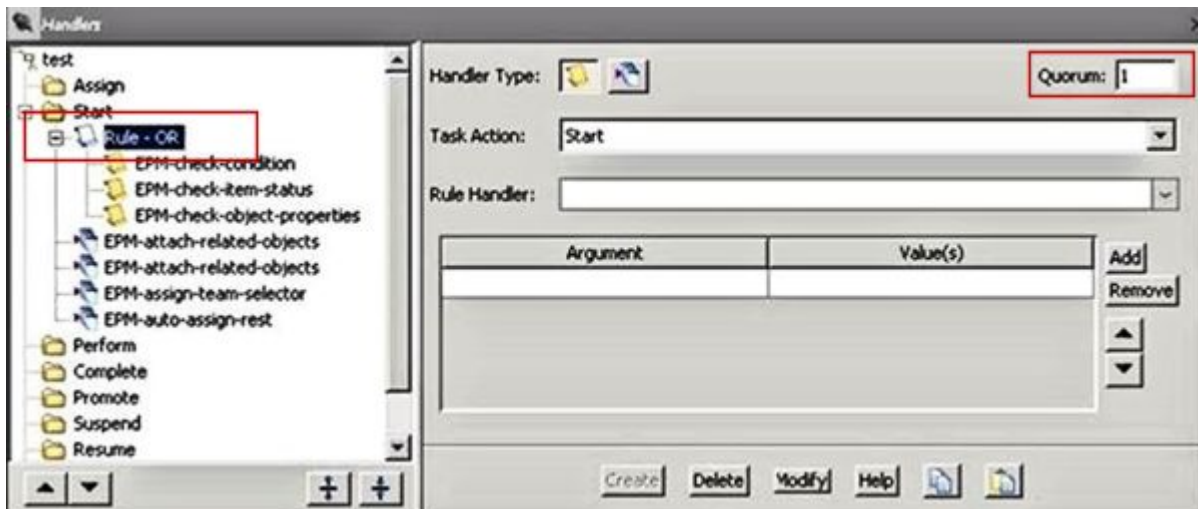
- Condition to satisfy: All rules must be satisfied for the workflow to continue.

    o   **Quorum** value: Equal to the number of rules in the **Rule** container.

    o   **Rule** container label: **Rule - AND**.

In this example, the **Quorum** value is **3**, which matches the number of rule handlers.
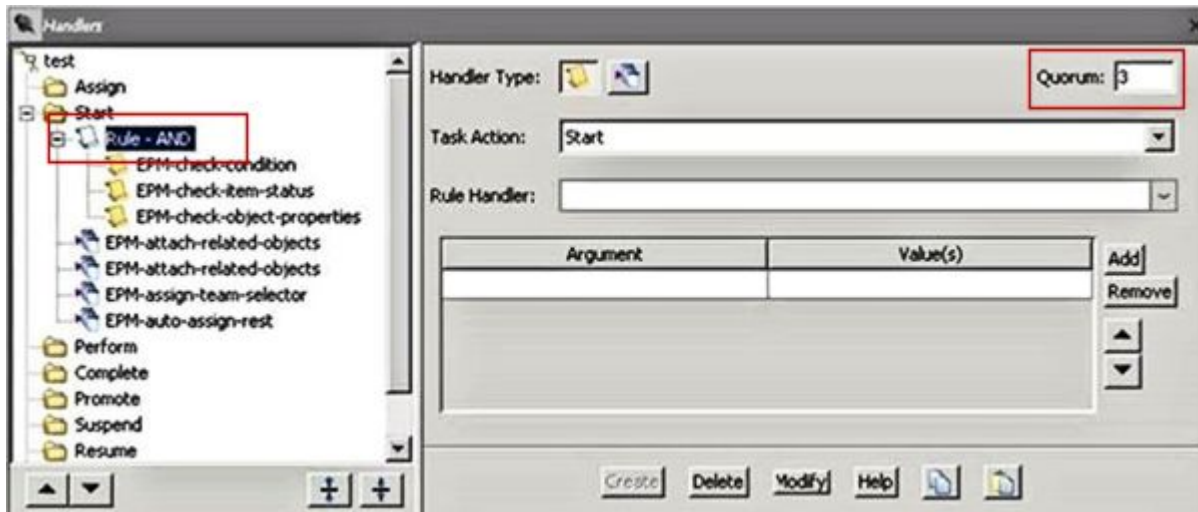
Delete task handlers

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.

Create an ACL and recipients for a task

1. On the 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. Click the **Attributes Panel** tab.

   The system displays the **Attributes Panel** dialog box.

4. Click **Named ACL** to add permissions for the task and target objects.

   a. 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.

   b. (Optional) To verify the assignment, view the **Task Handler** panel.

5. 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.

6. 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.

7. 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.
   >
   > o   The **Show Task in Process Stage List** displays the task in the **Process Stage List** property for the target object.
   >
   > o   Tasks in the **Process Stage List** are used to determine the ACL for the target objects.

8. 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.

9. Click **Close** to save the changes to the database and close the dialog box.

Requiring a PKI digital signature during a workflow

If you wish users to apply their PKI digital signature to objects in workflow, place the **EPM-apply-digital-signature** handler on an interactive workflow task. Where the handler is placed depends upon when you want the user to apply the digital signature.

Application of the digital signature	Place the handler as follows	Results
User applies signature to the workflow targets.	On the **Complete** action of a **Do**, **select-signoff-team**, **perform-signoffs**, **Condition**, or form task.	If the PKI authentication passed, their digital signature is applied to the workflow targets. If a schedule task is attached as a schedule attachment to the workflow, the digital signature is also applied to it.
User needs to be authenticated while selecting signoff members during the routing of the task.	On the **Complete** action of the **select-signoff-team** subtask of the **Review** task under the **Route** task.	
Multiple reviewers usually need to sign off the task.	On the **Perform** action of the **perform-signoffs** task.	Every user signing off the task is prompted for authentication. The digital signature from each user is applied when that user signs off.

If you want to check for valid digital signatures during the workflow, place the **EPM-verify-digital-signature** handler on a workflow task. You can use this handler on a **Validate** task and configure a failure path if the minimum number of valid signatures is not present or if there are void signatures, depending on the arguments used in the handler.

> **Note**
>
> - You can configure which attributes of an object cannot be changed after a digital signature is applied.
>
> - Do not design the workflow to modify the configured attributes of the object using other handlers on the same or a subsequent task in the workflow, including final approval. Modifications to configured attributes should be performed in tasks previous to applying the digital signature.
>
> - If a schedule task is attached to workflow with a schedule task attachment, do not configure the **State**, **Actual Finish Date**, and **Percent Complete** attributes because they are updated when workflow completes after the digital signature is applied.
>
> - For change management objects, do not configure the change states (**Closure**, **Maturity**, and **Disposition**) because they are updated following a digital signature.

Digital signatures are PKI authentication attempts and are logged as an audit event.

Requiring PKI authentication to perform a workflow task

If you want users to authenticate themselves before they can complete a workflow task, place the handler on an interactive workflow task. The task is completed only after the user provides valid PKI authentication, but does not apply a digital signature on any object.

PKI authentication attempts are logged as an audit event.

Adding schedule tasks and attachments to a workflow process

You can locate the schedule tasks attached to the workflow and add their related change objects as target or reference objects to the workflow as well as the schedule task itself.

Use the **EPM-attach-related-objects** handler with the **-from_attch** argument on a task in the workflow process to add them.

Chapter 8: Manage signoff behavior

Signoff profile creation

Signoff profiles are created by administrators based on group or role, making it easier to assign approvers to a task. Defined in the process template, signoff profiles are particularly useful in enforcing groups and roles in a signoff. 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 a particular **Review** task, the administrator creates three group profiles: a **Marketing/manager** profile, an **Engineering/manager** profile, and an **Engineering/engineer** profile.

To enhance project-based user assignments, administrators can use the **WRKFLW_show_user_assignment_options** preference to determine which tab in the signoff tree is active by default: the **Organization** tab or the **Project Teams** tab. By default, **Organization** is selected. You can also choose to show only the **Organization** tab and hide the **Project Teams** tab, or vice-versa. Users are filtered using group or role membership criteria.

Once an administrator has defined a signoff profile, you, as a member of the signoff profile, can choose to approve tasks by one of the following methods:

- Quorum format: approval based on a specified minimum number of approvers

- Percentage: approval based on specified percent of approvers

- All: approval based on return of all review and comments

Quorum and required signoff behavior

Use quorums for task signoffs to indicate the number (percentage) of users who must approve the task in order for it to complete.

- Use a quorum when a signoff task should proceed without waiting for undecided reviewers.

- Use a quorum to reduce the number of decisions required for the task to proceed.

Use required reviewers to ensure that the key reviewers have provided their decision.

> **Note**
>
> You can use required signoffs with quorums to prevent the task from completing until all required reviewers provide a signoff decision.
>
> For example:
>
> • When a task has five reviewers, but none are required, and quorum is set to **2**; the task proceeds when two reviewers provide their decisions.
>
> • If, however, one of the five reviewers is marked as required; the task does not proceed until the required reviewer provides a decision, even if the quorum is met.

You can make a reviewer required using one of the following methods:

• Use the **Assign All Tasks** tab when a workflow is created.

   The **EPM_valid_user_to_apply_assingment_list** preference determines which users are authorized for assigning resources.

• Assign when selecting a signoff team.

• Use the **EPM-adhoc-signoffs** or **EPM-fill-in-reviewers** handlers with the **-required** setting.

Use the **SIGNOFF_adhoc_quorum** preference to configure constraints on the quorum value during team selection. When ad hoc signoff is enabled, you can set quorum value limits or no constraints.

Workflow task assignment options

There are two categories of workflow task assignment options:

Interactive task assignment	Interactive tasks can use individual users or resource pools, but requires user input to complete. It includes manual assignment of tasks and the creation and application of process assignment lists (PALs).
Automated task assignment	Automated task assignment can use individual users, resource pools, or dynamic participants. Four action handlers perform automated assignment:

   • **EPM-auto-assign**

   • **EPM-auto-assign-rest**

   • **EPM-adhoc-signoffs**

   • **EPM-fill-in-reviewers**

> **Note**
>
> Use the **WRKFLW_allow_signoff_assignment_to_OOO_user** preference to control task assignment of a signoff when the delegate is a member of the signoff team.
>
> **True** assigns the signoff to the out-of-office user, while **False** assigns it to the resource pool of the out-of-office user.

Create a signoff profile

1. Double-click the **Review** task in the task hierarchy tree.

   The task expands, listing the **select-signoff-team** subtasks.

   > **Note**
   >
   > You can change the names of the **select-signoff-team** and **perform-signoffs** subtasks. For example, you can rename the subtasks to specify their parent task or the current step in the process (such as **select-design-signoff-team**).

2. 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.

3. 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.

4. Type the number or percentage of reviewers required for this particular group/role signoff profile.

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

6. Click **Create** to add this profile to the **Signoff Profiles** list. To change an existing profile in the **Signoff Profiles** list, click **Modify**. To delete an existing profile in the **Signoff Profiles** list, click **Delete**.

Define a surrogate for another user (requires administrative privileges)

1. Click **My Worklist** in the navigation pane.

    The system displays your inbox.

2. Choose **Tools→Workflow Surrogate**.

    The system displays the **Workflow Surrogate** dialog box.

3. Select the group, role, and user for whom you are defining surrogates.

    The dialog box displays surrogates for the selected user in the **Current Surrogate User(s)** list.

    > **Note**
    >
    > You can choose all roles within a group by selecting the asterisk (*) rather than selecting a specific role.

4. Select the group, role, and user to be a surrogate.

5. Set the **Surrogate Effective Dates** effectivity start date for the surrogate user as follows:

    a. Click the calendar button in the **From** box to open the popup calendar.

    b. Select the month in which the surrogate user becomes effective. Click the back arrow to scroll to the previous month or click the forward arrow to scroll to the next month.

    c. Type the year in which the surrogate user becomes effective.

        Click the back arrow to scroll to the previous month or click the forward arrow to scroll to the next month.

    d. Select the day the surrogate user becomes effective by clicking the appropriate square on the calendar.

    e. Type the hour, minute, and second at which the surrogate user's effectivity begins in the **h**, **m**, and **s** boxes.

        Use the 24-hour clock format; for example, type 1:30 p.m. as **13 h**, **30 m**, and **00 s**.

        If you do not specify another time or clear the boxes, the current time is entered.

    f. Click **OK** to accept the effectivity start date and time and close the calendar.

6. Set the **Surrogate Effective Dates** effectivity end date for the surrogate user:

    a. Click the calendar button in the **To** box to open the popup calendar.

    b. Select the month in which the surrogate user's effectivity ends.

        Click the back arrow to scroll to the previous month or click the forward arrow to scroll to the next month.

    c. Select the year in which the surrogate user's effectivity ends.

      Click the back arrow to scroll to the previous year or click the forward arrow to scroll to the next year.

    d. Select the day the surrogate user's effectivity ends by clicking the appropriate square on the calendar.

    e. Type the hour, minute, and second at which the surrogate user's effectivity ends in the **h**, **m**, and **s** boxes.

      Use the 24-hour clock format; for example, type 1:30 p.m. as **13 h**, **30 m**, and **00 s**.

      If you do not specify another time or clear the boxes, the current time is entered.

    f. Click **OK** to accept the effectivity end date and time and close the calendar.

> **Tip**
>
> To allow the surrogate user to be effective indefinitely, leave the end date unset. To reset the effectivity dates, click **Reset**.

7. Click **Add**.

   The system displays the surrogate user in the **Current Surrogate Users** list, the surrogate user is notified via email, and a link is created in the surrogate user's inbox.

   The link in the surrogate user's inbox allows the surrogate user to access the inbox of the user for whom they are acting surrogate.

> **Note**
>
> Configure the **WRKFLW_mail_surrogates** to send email notifications in the Workflow application. **True** sends email notifications to all active surrogate users. To prevent sending email notifications to all active surrogate users set the value to **false**.

Chapter 9: Using workflows to manage security and project data

Managing security and project data using custom forms

Developers create custom forms in the Business Modeler IDE. The workflow administrator can use those forms to customize workflow task templates for the following purposes:

- **Assigning members to projects**

  In the Business Modeler IDE, the developer creates a custom form with properties for members (privileged and nonprivileged) and a property for projects. Attached to each property, a dynamic List of Values (LOV) gathers all of the available members or projects.

- **Assigning and removing projects on workflow targets**

  In the Business Modeler IDE, the developer creates a custom form with properties for assigning and removing projects. Attached to each property, a dynamic List of Values (LOV) gathers all of the available projects.

- **Setting security classifications on workflow targets**

  In the Business Modeler IDE, the developer creates a custom form with properties for government classification, intellectual-property (IP) classification, or both classifications. A classification property contains a List of Values (LOV) from which the responsible party can select the classification to set on the target.

Assign members to projects using workflow arguments

The workflow initiator can access the form properties and modify the target. Then, the workflow administrator configures the task template to assign the members to the project, using the **PROJ-assign-members** handler.

In the task, the responsible party first creates an instance of the form. The responsible party then selects the appropriate value from each list of values (LOV).

You can add project members by using form properties attached to the workflow template.

1. From the Workflow Designer main screen, select a **Process Template** from the drop-down list. Ask your Business Modeler IDE administrator if you are unsure of the template name.

   Alternately, you can create a new workflow process. Go to **File→New→Workflow Process**.

2. In the workflow, click the task (for example, **Assign Project Members**). The **Handlers** dialog box displays.

3. The handler displays under the **Complete** folder. Enter the arguments and values. For example:

   - The projects to receive members are named Proj1 and Proj2.

- The user named John is to be added to both projects as a nonprivileged member. This user has the **Designer** role in the**Engineering** group.

- The user named Jane is to be added to both projects as a privileged member. This user has the **Manager** role in the **Engineering** group.

Argument	Values
-projects	**Proj1,Proj2**
-members	**Engineering/Designer/john**
-privileged_members	**Engineering/Manager/jane**

4. Click **Create**. The members you entered into the argument are displayed in the **Member Selection** list for the project.

5. Go to **My Worklist** and select the task from the **Task to Perform** folder.

6. Select **Action** and then **Perform** from the menu at the top of the screen.

7. From the LOV on the **Assign Member** form, select the name of the user you want to add.

8. Select **Complete** and click **OK**. The selected members are added to the project.

Assign a project to workflow targets

The responsible party can access the form properties and modify the target. Then, the workflow administrator configures the task template to:

- Create a form instance and relate it to the task, using the **EPM-create-form** handler.

- Display the form, using the **EPM-display-form** handler.

- Copy the values from the form to the target, using the **PROJ-update-assigned-projects** handler.

1. For the item you want to assign, select the **Item Revision**.

2. Select the **Process Template** from the drop-down list. Ask your Business Modeler IDE administrator if you are unsure of the template name.

   Alternately, you can select the **Item Revision** and go to **File→New→Workflow Process** to create a new process.

3. To display the form, click the name of the form (for instance, Create ProjMemberForm). The **Handlers** dialog box displays.

4. Select the handler under the **Perform** folder. Enter the arguments and values.

5. Click **Create**.

6. Go to **My Worklist** and select the item revision. From the **New Process Dialog**, select **Approve Project Updates** from the **Complete** folder.

7. Select the handler under the **Perform** folder. Enter the arguments and values.

8. Click **Create**.

9. Select **Action**, and then **Perform** from the menu at the top of the screen.

10. Select **Complete** and click **OK** in the **Perform Do Task** dialog box.

Setting the security classification on a workflow target

Once a form is created in Business Modeler IDE, the workflow administrator can perform any of the following:

- Create a process template and task templates that display the custom form to the user.

- Allow the user to read and write to the objects involved with the handler.

- Set the classification on the target. Security classifications are set using the **EPM-set-property** handler.

Chapter 10: Using workflow templates at multiple Teamcenter sites

Configuring remote workflows

> **Caution**
>
> If you are using Global Change Management, both **WRKFLW_allow_replica_targets** and **TC_disallow_release_status_on_replica** must both be set to **false** to allow application of release status to replicas that are not remotely checked out during a remote workflow activity.

To enable remote workflows, install or configure the following components:

- Teamcenter 4 tier installation with these templates:

    o   Change Management

    o   Change and Schedule Management Interface

- Dispatcher/Asynchronous server

- Application Registry

- Teamcenter Integration Framework

- Teamcenter SSO (single sign-on; proxy link mechanism to communicate with the remote site)

- CMS-RPC (connect multisite remote-procedure call; set up outside Teamcenter SSO)

Distributing workflow templates using Multi-Site Collaboration

Replicate a workflow template

You can distribute your workflow templates to different Teamcenter sites by replicating templates using Multi-Site Collaboration. 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.

1.   If necessary, create the template you want to replicate.

2.   Run the 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.

   > **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**.

2. Run the utility with the following arguments:

   > **data_sync** **-u=**_user-id_ **-p=**_password_ **-g=**_group_ **-f=sync**
   > **-site=**_remote-site-name1_ **-class=**_class-name_ **-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):

   > **data_sync -f=sync -site=teamcentersite2 -class=EPMTaskTemplate -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.

Distributing workflow templates using Workflow Designer

Importing and exporting workflow templates

You can distribute your workflow templates to different Teamcenter sites by importing and exporting workflow process and task templates from the Teamcenter database in an XML format.

*   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.

Note

You can import and export workflow templates using the Workflow Designer **Tools** menu, or you can use the and utilities for these tasks.

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.

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.

    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 ignore the origin ID of the template you are importing by selecting the **Ignore origin ID check** check box.

    Select this option if you get the following error when attempting to import workflow templates:

    ```
    The importing template(s) do not match with the existing template(s).
    The import of template(s) in overwrite mode failed.
    ```

6. (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.

When you import templates from a Teamcenter version prior to 10.1, do not select the **Apply template changes to all active workflow processes** check box. If you do, Teamcenter does not successfully import the template.

Updates are applied as described in *How process template edits are applied to active processes*.

> **Note**
>
> • This check box is visible only if the **EPM_enable_apply_template_changes** preference is set to **OPTIONAL**.
>
> • This check box is not available if you selected the **Ignore origin ID check** check box.

7. (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.

8. 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

1. Choose **Tools→Export**.

   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.

Chapter 11: Working with remote inboxes

Sending schedule tasks through workflows at remote sites

A workflow for a schedule task is created as a remote workflow when the schedule task's privileged user or, in the absence of a privileged user, the workflow owner, is a remote user. When the system creates the remote workflow, it links the schedule task to that workflow and attaches all task attachments using GRM (Generic Relationship Management) relations. It also replicates the attachments and either checks them out remotely or transfers ownership to the remote site.

The following conditions must be met to create a workflow for a schedule task at a remote site:

- Schedule task's privileged user or, in the absence of a privileged user, the workflow owner, is a remote user

- Schedule task's privileged user and the workflow owner are from the same remote site

- Schedule task has a workflow process template required to create the workflow on the remote site

- Only one user is assigned to the schedule task

- The **WRKFLW_create_remote_workflow** preference is configured to enable remote workflow creation

Enabling remote inboxes

Remote inboxes are created when you subscribe to your account inbox at a remote site. This action creates a link in your local site worklist. When you click the link, a new Teamcenter client session is started that runs against the remote site. You can then see and perform tasks in your worklist on the remote site. There is an associated **Remote Checkout** command that lets you place data on your local site for work.

To enable remote inbox functionality, your site must be configured to use the application registry and interoperability linking. Information about configuring this functionality is currently available on the Global Technical Access Center (GTAC) Web site. Teamcenter administrators with valid WebKey accounts can access the *Teamcenter Interoperability* guide at the following location:

http://support.ugs.com/docs/tc_eng/8/en/tss00004.pdf
Chapter 1, *Introduction*, see *Understanding Components Required for Interoperability*, *Using Application Registry*.
Chapter 2, *Installing Application Registry*.
In chapter 3, *Configuring for Basic Linking*, see *Linking Teamcenter Engineering To Remote Engineering*.

In chapter 4, *Configuring Advanced Linking*, see the following topics:

•   *Understanding Teamcenter for engineering process management*

•   *Advanced Linking Terms*

•   *System Requirements*

•   *Configuration Overview*

•   *Deploying Teamcenter Application Registry*

•   *Configuring Teamcenter for engineering process management*

•   *Setting and Verifying Advanced Linking Properties*

Chapter 6, *Troubleshooting*

Remote inboxes let you interact with workflow tasks that originated at remote sites.

•   When you have a user account at a remote site, you can subscribe to that site to access your inbox, called your *remote inbox*, and access tasks assigned to you at the remote site.

•   After you subscribe to your inbox at a remote site, your local site worklist displays a remote site link you can use to launch a client to let you access the remote site inbox.

> **Note**
>
> The remote site link also shows the number of tasks in your worklist at the remote site. However, this number is not refreshed automatically.
>
> Click on the remote inbox link to launch the login to the remote site. The remote site then displays the accurate task list.

•   Remote site links in the local site worklist cannot be expanded in the local tree display.

•   When you click the link to the remote site, Teamcenter launches a full, separate Teamcenter session to display the remote inbox.

> **Note**
>
> The remote site link launches the client specified by the **TC_external_default_launch_ui** setting.
>
> o **thin**
>
>   Displays the remote site in the thin client.
>
>   This is the default setting.
>
> o **rich**
>
>   Displays the remote site in the rich client.

o **dynamic**

Displays the remote site in the same type of client used to access the link to the remote site.

- Remote inboxes contain **Tasks to Perform** and **Tasks to Track** folders.

Working with task data in remote inboxes

You can use **Remote Checkout** and **Remote Export** commands to access to data for tasks in your remote inboxes.

- **Remote Checkout** lets you access modifiable replicas of the target data associated with the tasks assigned to you.

    o When a workflow task requires you to modify data located at a remote site, use **Remote Checkout** to check out and send an editable copy of the data to your local **Home** location.

    This checks out the data at the remote site and puts the data on the local site in the checked-out state.

    o When you have completed the data modification, use the standard **Check-In** option at the local site to undo the checkout at the remote site, move the modified data to the remote site.

    This checks in the data at the remote site.

- **Remote Export** lets you access read-only replicas of data. If necessary, you can also use this command to transfer site ownership of the data required to perform your tasks.

Subscribe to a remote inbox

1. Choose **Tools→Remote Inbox Subscription**.

    The system displays the **Remote Inbox Subscription Dialog** dialog box.

    - Sites with remote inboxes to which you are already subscribed are listed as **Selected Inboxes**.

    - Sites with remote inboxes to which you are not already subscribed are listed as **Available Inboxes**.

2. To subscribe to an available inboxes, select the site in the **Available Inboxes** list and click **Add**(**+**).

    To unsubscribe from any of your subscribed inboxes, select the relevant inboxes in the **Selected Inboxes** list and click **Remove** (**–**).

3. When the subscriptions are listed correctly, click **OK** or **Apply**, and the system displays the **Subscribe Remote Inbox** dialog box. This dialog box shows the progress of each subscription request.

> **Note**
>
> Configure the **TC_subscription_available_subscription_handlers** preference to display the handlers available during the create or modify subscription. The **IMAN_Smpt_Mail_notify** handler is the default value and cannot be deleted.

Check out data to your local site from a remote site

1. Start a client that accesses a remote site.

   > **Note**
   >
   > When you subscribe to a remote inbox, your worklist displays a link that lets you launch a client that accesses the remote site.

2. Select the object to check out from the remote site, and choose **Tools→Multi-Site Collaboration→Send→Remote Checkout**.

   The system displays the **Remote Checkout** dialog box.

3. Type the following information in the dialog box:

   - **Change ID**

     Type the change number associated with the checkout request.

   - **Comments**

     Type the reason for the checkout request.

   - **Target Site**

     From the list of available sites, choose the site to which the object should be sent.

     Click **Home** on the right side of the **Target Site** box to choose sites from the list.

   - **OK to remote checkout?**

     Displays the status of objects being remotely checked out.

4. Click **Yes**.

   The system displays the current options in the **Remote Checkout Options Settings** dialog box.

5. Click **Yes** to continue or **No** to cancel the checkout operation.

   The system displays the status of the checkout operation. The checked-out data goes to your home folder on the target site.

Export data to your local site from a remote site

1. Start a client that accesses a remote site.

> **Note**
>
> When you subscribe to a remote inbox, your worklist displays a link that lets you launch a client that accesses the remote site.

2. In the remote site client, select the object to export as a read-only replica.

3. Choose **Tools→Multi-Site Collaboration→Send→Remote Export**.

   The system displays the **Remote Export** dialog box.

4. Enter the following information in the dialog box:

   - **Reason**

     Type the reason for the data export.

   - **Target Sites**

     From the list of available sites, select the site to which the object should be sent.

     Click **Home** to the right of the **Target Site** box to select sites from the list.

   - **OK to remote export?**

     Displays the status of objects being remotely exported.

5. Click **Yes**.

   The system displays the current options in the **Remote Export Options Settings** dialog box.

6. Click **Yes** to continue or **No** to cancel the export operation.

   The system displays the status of the export operation. The exported data goes to your home folder on the target site.

Chapter 12: Workflow handlers

What are 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 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.

- Rule handlers integrate workflow business rules into EPM workflow processes at the task level. They attach conditions to an action. 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. If there are multiple targets for a single rule handler, all targets must satisfy the rule for **EPM_go** to be returned (**AND** condition).

  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 **EPM_go** for the action to complete.

Action and rule handlers in the **Handlers** panel can be copied:

- From one action to another action in a task.

- From one task to another task in the same template.

- From a task in one template to a task in another template.

For the selection in the action tree, click **Performs a Copy action** or **Performs a Paste action** as desired.

> **Note**
>
> For **Performs a Paste action**, the process template must be in **Edit** mode.
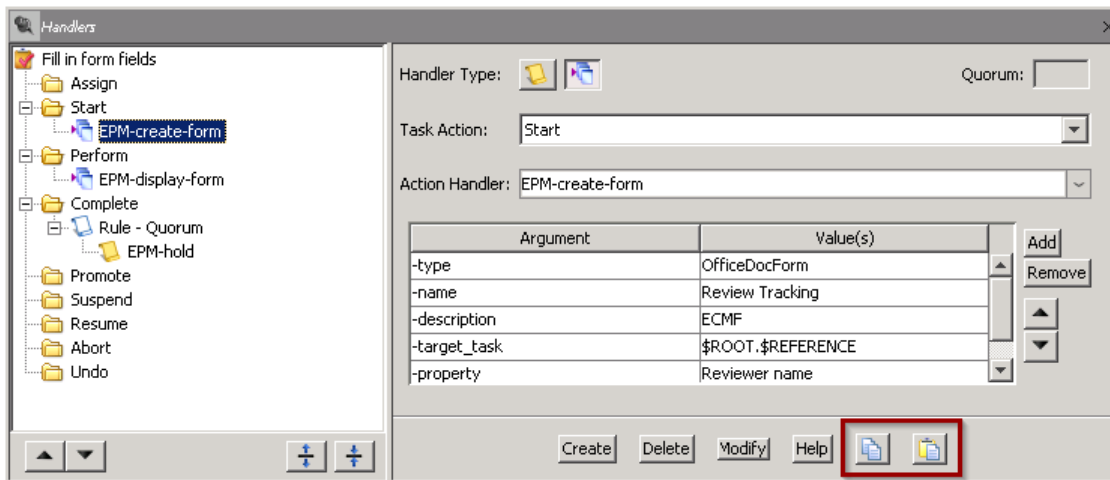
- To paste on another task in the same template, select the target task in the task hierarchy tree.

- To paste on a task in another template, select the target template from the **Process Template** list.

Updating your task templates to use the new handler and argument names

Starting with Teamcenter version 10.1, many of the workflow handlers, their arguments, and accepted argument values were changed to make them more consistent. The effect of the renaming depends on your situation:

- If you did not have an installation of Teamcenter prior to version 10.1, the renaming has no effect for you.

- If your installation was upgraded from a Teamcenter version prior to 10.1 to the current version, the utility was run during the upgrade and the handlers and arguments provided by Teamcenter were automatically renamed.

- If you are importing templates from a Teamcenter version prior to 10.1 to the current version, you must run the utility after importing the templates to rename the handlers and arguments.

    When you import templates from a Teamcenter version prior to 10.1, do not select the **Apply template changes to all active workflow processes** check box in the rich client or use the **-apply_template** argument in the **plmxml_import** utility. If you do, Teamcenter does not successfully import the template.

- If you have custom handlers, you can use the utility and a custom mapping file to rename your custom handlers and arguments.

Renaming your custom handlers and arguments

You can use a custom XML mapping file and the utility to rename your custom handlers and arguments to make them consistent with the Teamcenter handlers and arguments. The elements of the mapping file are:

Element	Attributes	Usage
`<Mapping>`	None.	The `<Mapping>` element is the root level element in the XML file.
`<Remove>`	None.	Removes a handler or handler argument depending on where it is placed and its child elements.
		If `<Remove>` is the top level element, it may only have a `<Handler>` element as a child.
		If `<Remove>` is the child of a `<Handler>` element, it may only have one or more `<Argument>` elements as children. If an Argument value is specified, the Argument is removed only if the Argument value in the mapping file is a subset of the actual Argument Value in the system. If an Argument value is not specified, the Argument is removed, ignoring whether it has any value or not.
`<Replace>`	None.	Replaces a handler with more than one handler as specified by subsequent `<Add>` elements. The `<Handler>` child element of the `<Replace>` names the handler to be replaced. The arguments that need to be copied over to the new handlers (for example, see arg3 below) should be explicitly identified. If an argument from the old handler is not explicitly defined to be copied over, it is not added to a new handler, unlike the update/rename handler case.
		For replacing one handler with another single handler, use the `<Update>` element.
`<Update>`	None.	Changes a handler's name and/or arguments.
`<Argument>`	• `name` (optional) The current name of an argument. • `value` (optional) The current value of an argument. • `newName` (optional) The new name to be given to an argument. • `newValue` (optional)	Specifies the current and possibly new names and values for arguments of a handler.

Element	Attributes	Usage
	The new value to be given to an argument.	
	• `index` (optional) Position of the argument in the handler. The `index` and `name` attributes are mutually exclusive.	
`<Handler>`	• `name` The current name of a handler.	Specifies the current and possibly new name of a handler.
	• `newName` (optional) The new name to be given to a handler.	
	• `transformAssignees=`*`to-be-argname`* (optional) Use this attribute when your existing handler has any number of users, groups, roles, address lists, and/or resource pools as arguments where they are not already specified in the form of a *-argname=argvalue* pair (such as **-participant=Smith**).	
`<Criteria>`	• `match` (optional) `false`—the result of the criteria should be negated. `true`—default value.	Specifies restrictions on the `<Handler>` element in which it is embedded. The action specified by the `<Handler>` element is only applied if the criteria evaluate to true. `<Criteria>` may have two child elements: `<Template name="`*`template-name`*`" />`, and `<Argument name="`*`arg-name`*`" value="`*`arg-value`*`" />` that may be specified alone or together. The *template-name* is compared to the name of template containing the handler. The *arg-name* and *arg-value* are compared to the list of handler arguments. If both `<Template>` and `<Argument>`

Element	Attributes	Usage
		are specified, a handler must match both of the respective attributes.
`<Template>`	• `name=template-name` Compared to the name of the template containing the handler.	Restricts the `<Criteria>` element in which it is embedded to the specified template. If both `<Template>` and `<Argument>` are specified, a handler must match both of the respective attributes.
`<Argument>`	• `name=arg-name` Compared to the list of handler argument names. • `value=arg-value` Compared to the list of handler argument values.	Restricts the `<Criteria>` element in which it is embedded to the specified argument name and value. If both `<Template>` and `<Argument>` are specified, a handler must match both of the respective attributes.
`<Add>`	None.	Adds a handler or handler argument. Unlike the `<Remove>` element, `<Add>` is never a top level element, but is always a child of a `<Handler>` element.
`<Modify>`	None.	Modifies a handler argument.
`<Split>`	• `name` An argument name. • `newName` An argument value. • `Delimiter` (optional) For splitting two delimited values existing only in the handler name field. For example, values delimited by two colons (**::**).	Splits any handler argument *old-name=old-value* pair into separate arguments *name1=old-name* and *name2=old-value*. A wildcard may be used for the name to match *old-name*. For example, `<Split name="*" newName="-source,-decision" />` splits **Cond1=Checked** and **Cond2=true** into **-source=Cond1**, **-decision=Checked**, and **-source=Cond2**, **-decision=true**. Because handler arguments with the same name are combined into a single argument, this finally results in **-source=Cond1,Cond2** and **-decision=Checked,true**.

Note

For any handler matched and processed by the **migrate_wf_handlers** utility, arguments having the same name are combined into a single argument with a resulting value composed of a comma-separated list.

Here is a full example of a mapping file:

```
<Mapping>
   <Remove>
      <!-- Remove all instances of Handler -->
      <Handler name="old-handler-name">
      </Handler>
   </Remove>

   <Update>
      <Handler name="old-handler-name" newName="new-handler-name">
         <Remove>
            <!-- if value is specified, remove the argument only if arg4
            has value val4 -->
            <Argument name="arg4" value="val4"/>

            <!-- if value is not specified, remove argument irrespective
            of its value -->
            <Argument name="arg5" />
         </Remove>
      </Handler>

   <Replace>
    <Handler name="old-handler-name">
      <Add>
         <Handler name="new-handler1">
            <!-- copy value from arg1 to new-arg1 -->
            <Argument name="arg1" newName="new-arg1" />

            <!-- if arg2 has val2 (substring match) on old handler,
            add new argument new-arg2, copy over the value and
            replace the substring to new-sub-value2  -->
            <Argument name="arg2" value="sub-val2-1" newName="new-arg2"
            newValue="new-sub-val2-1" />

            <!-- The same argument can be repeated multiple times for different
            substring value -->
            <Argument name="arg2" value="sub-val2-2" newName="new-arg2"
            newValue="new-sub-val2-2" />

            <!-- if arg3 is defined on old handler, add it to new handler
            and copy its value from old handler -->
            <Argument name="arg3" />

            <!-- add new argument with new value -->
            <Argument newName="new-arg6" newValue="new-val6"/>

         </Handler>

         <Handler name="new-handler2">
            <Argument newName="new-arg5" newValue="new-val5"/>

            <!-- copy value from arg1 to new-arg1 -->
            <Argument name="arg1" newName="new-arg1"/>
         </Handler>
      </Add>
```

```
</Replace>

<Update>
   <!-- Rename the old handler, as well as removing, adding and modifying
   its arguments. -->
   <!-- If any handler argument names are not mentioned in remove/modify
   sections, they are copied over to new handler. -->
   <Handler name="old-handler-name" newName="new-handler-name">
      <Remove>
         <!-- if value is specified, remove the argument only if arg4 has
         val4 -->
         <Argument name="arg4" value="val4"/>

         <!-- if value is not specified, remove argument irrespective of its
         value -->
         <Argument name="arg5"/>
      </Remove>

      <Add>
         <Argument name="new-arg6" value="new-val6"/>

         <!-- if value is not specified or is empty, set the argument value
         to empty -->
         <Argument name="new-arg7" value=""/>
      </Add>

      <Modify>
         <Argument name="arg1" value="val1" newName="new-arg1"
         newValue="new-val1"/>

         <!-- if newValue is not specified, copy the old argument value to
         new argument -->
         <Argument name="arg2" value="val2" newName="new-arg2" />
      </Modify>
   </Handler>
</Update>

<Remove>
   <!-- Remove Handler if the criteria matches (arg1 exists with value
   val1 and arg2 exists) -->
   <Handler name="old-handler-name">
      <Criteria>
         <Argument name="arg1" value="val1"/>
         <Argument name="arg2"/>
      </Criteria>
   </Handler>
</Remove>

<Update>
   <!-- Rename Handler if arg3 does not exist on the handler -->
   <Handler name="old-handler-name" newName="new-handler-name">
      <!-- If match set to 'false', the result of the criteria should be
      negated. (!)  -->
      <Criteria match="false">
         <Argument name="arg3"/>
      </Criteria>
```

```
            </Handler>
        </Update>

        <Update>
            <!-- Add one or more handler arguments -->
            <Handler name="old-handler-name">
                <Add>
                    <Argument name="new-arg1" value="new-val1"/>
                    <Argument name="new-arg2" value="new-val2"/>
                </Add>
            </Handler>
        </Update>

        <Update>
            <Handler name="old-handler-name">
                <Modify>
                <Argument name="arg1" value="val1" newName="new-arg1"
                newValue="new-val1"/>

                <!-- if newValue is not specified, copy over the old argument value to
                new argument -->
                <Argument name="arg2" value="val2" newName="new-arg2"/>

                <!-- if newValue is empty, clear the value for new argument. If val3 is
                a substring of original value, special care should be taken in
                removing ',' -->
                <Argument name="arg3" value="val3" newName="new-arg3" newValue=""/>

                <!-- if new argument name is not specified, do not rename the argument,
                but modify the argument value -->
                <Argument name="arg8" value="val8" newValue="new-val8" />

                <!-- Rename Handler Argument, keeping/copying-over the value -->
                <Argument name="arg9" newName="new-arg9" />

                <!-- Irrespective of the name of the argument, rename it to new-arg1
                and copy the argument name as value of the new argument. If the new
                argument name is already defined/added on the handler, append the
                value to existing value of that argument with delimiter set in the
                preference. -->
                <Argument name="*" newName="new-arg11" newValue="$ARGNAME"/>

                <!-- Replace the argument value by another value which includes the
            original value. If value is a comma separated list,
                                the new value will be a comma separated list
                                with the static string (user:) added
            to each value in the list. -->
                <Argument name="user" newName="-assignee" newValue="user:$ARGVALUE"/>

                <!-- index attribute will mention the arguments sequence in the
            handler. name and index are mutually exclusive. -->
                <Argument index="1" newName="year" newValue="$ARGNAME"/>
                <Argument index="2" newName="week" newValue="$ARGNAME"/>
                </Modify>
            </Handler>
        </Update>
```

```
<Update>
    <!-- Rename Handler example. Rename "old-handler-name" handler to
"new-handler-name" for all instances of "old-handler-name" handler -->
    <Handler name="old-handler-name" newName="new-handler-name">
</Update>

</Mapping>
```

Renaming of Teamcenter handlers, arguments, values, and keywords

Starting with Teamcenter version 10.1, many of the workflow handlers, their arguments, and accepted argument values were changed to make them more consistent. The following handler names have been changed as indicated:

Handler name prior to Teamcenter 10.1	New handler name beginning in Teamcenter 10.1
add-status	EPM-set-status
adhoc-signoffs	EPM-adhoc-signoffs
approve-service-structure	SERVICEPROCESSING-approve-service-structure
ASB-attach-physical-components	ASBUILT-attach-physical-components
ASM-attach-physical-components	ASMAINTAINED-attach-physical-components
assert-signoffs-target-read-access	EPM-assert-signoffs-target-read-access
auto-assign	EPM-auto-assign
auto-assign-rest	EPM-auto-assign-rest
auto-relocate-file	SMP-auto-relocate-file
change-all-started-to-pending	EPM-change-all-started-to-pending
check-condition	EPM-check-condition
check-responsible-party	EPM-check-responsible-party
check-signoff	EPM-check-signoff
CR-assert-targets-checked-in	EPM-assert-targets-checked-in
CR-assign-team-selector	EPM-assign-team-selector
CR-change-group-owner	EPM-change-group-owner
CR-change-target-group	EPM-change-target-group
CR-change-target-group-owner	EPM-change-target-group-owner
CR-check-item-status	EPM-check-item-status
create-status	EPM-create-status
CR-fill-in-reviewers	EPM-fill-in-reviewers
CR-notify	EPM-notify-report
debug	EPM-debug
debug-rule	EPM-debug-rule
demote	EPM-demote
demote-on-reject	EPM-demote-on-reject
disallow-adding-targets	EPM-disallow-adding-targets
disallow-removing-targets	EPM-disallow-removing-targets

Handler name prior to Teamcenter 10.1	New handler name beginning in Teamcenter 10.1
EPM-add-released-parts-queue	RDV-add-released-parts-queue
EPM-assert-target-classified	ICS-assert-target-classified
EPM-attach-assembly-components	PS-attach-assembly-components
EPM-attach-mgcitemrev-targets	MGC-attach-mgcitemrev-targets
EPM-attach-targets-AH	ERP-attach-related-targets-AH
EPM-check-assembly-status-progression	PS-check-assembly-status-progression
EPM-check-occ-notes	PS-check-occ-notes
EPM-check-validation-result	VAL-check-validation-result
EPM-check-validation-result-with-rules	VAL-check-validation-result-with-rules
EPM-delete-ugcgm-markup	RDV-delete-ugcgm-markup
EPM-export-AI-AH	AI-export-AH
EPM-export-to-plmxmlfile	PIE-export-to-plmxmlfile
EPM-generate-image	RDV-generate-image
EPM-generate-ugcgm-drawing	RDV-generate-ugcgm-drawing
EPM-make-mature-design-primary	PS-make-mature-design-primary
EPM-mark-archive	AR-mark-archive
EPM-perform-offline-export	BC-perform-export
EPM-publish-target-objects	PUBR-publish-target-objects
EPM-send-target-objects	OBJIO-send-target-objects
EPM-set-condition-by-check-validation-result	VAL-set-condition-by-check-validation-result
EPM-set-form-value-AH	ERP-set-form-value-AH
EPM-tessellation-handler	RDV-tessellation-handler
EPM-unpublish-target-objects	PUBR-unpublish-target-objects
execute-follow-up	EPM-execute-follow-up
inherit	EPM-inherit
invoke-system-action	EPM-invoke-system-action
invoke-system-rule	EPM-invoke-system-rule
late-notification	EPM-late-notification
notify	EPM-notify
notify-signoffs	EPM-notify-signoffs
release-asbuilt-structure	ASBUILT-release-asbuilt-structure
release-asmaintained-structure	ASMAINTAINED-release-asmaintained-structure
require-authentication	EPM-require-authentication
schmgt-approve-timesheetentries	SCHMGT-approve-timesheetentries
schmgt-revise-timesheetentries	SCHMGT-revise-timesheetentries
schmgt-reject-timesheetentries	SCHMGT-reject-timesheetentries
set-condition	EPM-set-condition
set-duration	EPM-set-duration
set-parent-result	EPM-set-parent-result
set-status	EPM-set-status
suspend-on-reject	EPM-suspend-on-reject

Handler name prior to Teamcenter 10.1	New handler name beginning in Teamcenter 10.1
system	EPM-system
TCX-auto-approve-first-step	TCRS-auto-approve-first-step
TCX-check-approver	TCRS-check-approver
TCX-check-bomchild-statuslist	TCRS-check-bomchild-statuslist
TCX-check-bom-precise	TCRS-check-bom-precise
TCX-check-comps-against-pattern	TCRS-check-comps-against-pattern
TCX-check-datasets	TCRS-check-datasets
TCX-check-itemrev-status	TCRS-check-itemrev-status
TCX-check-jobowner	TCRS-check-jobowner
TCX-check-prev-itemrev-status	TCRS-check-prev-itemrev-status
TCX-check-signoff	TCRS-check-signoff
TCX-check-status	TCRS-check-status
TCX-has-target-drawing	TCRS-has-target-drawing
TCX-create-form	TCRS-create-form
TCX-create-snapshot	TCRS-create-snapshot
TCX-Create-Translation-Request	TCRS-Create-Translation-Request
TCX-delete-dataset	TCRS-delete-dataset
TCX-delete-log-datasets	TCRS-delete-log-datasets
TCX-export-signoff-data	TCRS-export-signoff-data
TCX-IRM-cleanfields	TCRS-IRM-cleanfields
TCX-purge-dataset	TCRS-purge-dataset
TCX-release-previous-itemrevs	TCRS-release-previous-itemrevs
TCX-remove-targets-with-status	TCRS-remove-targets-with-status
TCX-set-bom-precise	TCRS-set-bom-precise
TCX-store-cr-data	TCRS-store-review-data
TCX-trigger-approve-first-step	TCRS-trigger-approve-first-step
trigger-action	EPM-trigger-action
trigger-action-on-related-process-task	EPM-trigger-action-on-related-process-task
validate-for-checkedout-asmaintained -physicalpartrevision	ASMAINTAINED-validate-for-checkedout- physicalpartrevision
validate-for-checkedout-physicalpartrevision	ASBUILT-validate-for-checkedout-physicalpartrevision
validate-for-class	MROCORE-validate-for-class
validate-for-latest-asmphysicalpartrevision	ASMAINTAINED-validate-for-latest-asmphysicalpartrevision
validate-for-physicalpartrevision	ASBUILT-validate-for-physicalpartrevision
validate-for-unserviceable -physicalpartrevision	ASMAINTAINED-validate-for-unserviceable-physicalpartrevision
validate-missing-asmaintained-structure	ASMAINTAINED-validate-missing-asmaintained-structure
validate-missing-structure	ASBUILT-validate-missing-structure

The following handler argument names have been changed or removed as indicated:

Handler name beginning in Teamcenter 10.1	Argument name prior to Teamcenter 10.1	New argument name beginning in Teamcenter 10.1
EPM-adhoc-signoffs	**AUTO_COMPLETE**	**-auto_complete**
	-auto_incomplete	Argument removed.
	-conventional-execution	**-ce**
EPM-assign-team-selector	**-owner**	**-assignee=$PROCESS_OWNER**
	-person	**-assignee=person:***person-name*
	-resourcepool	**-assignee=resourcepool:***pool-name*
	-user	**-assignee=user:***user-name*
EPM-attach-related-objects	**-status_allow**	**-allowed_status**
	-status_disallow	**-disallowed_status**
	-att_type	**-attachment**
	-exclude_type	**-exclude_related_types**
	-type	**-include_related_types**
EPM-auto-assign	*resource pool*	Argument removed.
	-owner	**-assignee=$PROCESS_OWNER**
	-person	**-assignee=person:***person-name*
	-resourcepool	**-assignee=resourcepool:***pool-name*
	-user	**-assignee=user:***user-name*
	subtasks	**-subtasks**
EPM-auto-assign-rest	*list-of-users* (legacy syntax)	**-assignee**
EPM-auto-check-in-out	**-include_type**	**-include_related_type**
	-user	**-assignee**
		The values can be **$REVIEWERS** or **$RESPONSIBLE_PARTY**.
EPM-change-ownership	**-owner**	**-assignee**
		The values can be **$REVIEWERS** or **$RESPONSIBLE_PARTY**.
EPM-check-action-performer-role	*user-values*	**-responsible**
EPM-check-condition	*task-name*	**-source_task**
	None.	**-decision**
EPM-check-item-status	**-status**	**-allowed_status**
	-type	**-include_related_type**
EPM-check-object-properties	**-att_type**	**-attachment**
		The values are now lowercase.
	-props	**-property**
	-type	**-include_type**
	-values	**-value**
EPM-check-related-objects	**-status**	**-allowed_status**
	-target_type	**-include_type**
EPM-check-signoff	**-QUORUM**	**-quorum**
EPM-check-signoff-comments	**-decision**	**-decision**
		The values are now lowercase.

Handler name beginning in Teamcenter 10.1	Argument name prior to Teamcenter 10.1	New argument name beginning in Teamcenter 10.1
EPM-check-target-attachments	**-att_type**	**-include_related_type**
	-attachtype	**-include_related_type**
	-status	**-allowed_status**
	-target	**-include_type**
	-type	**-include_type**
EPM-check-target-object	**-status_allow**	**-allowed_status**
	-status_disallow	**-disallowed_status**
	-status	**-disallowed_status**
EPM-create-form	**-location**	**-target_task**
	-default	**-property** and **-value**
EPM-create-relation	**-primary**	**-primary_attachment**
	-secondary	**-secondary_attachment**
EPM-create-status	*status*	**-status**
EPM-create-sub-process	**-exclude_related_types**	**-exclude_related_type**
	-exclude_types	**-exclude_type**
	-include_related_types	**-include_related_type**
	-include_types	**-include_type**
	-process_desc	**-description**
EPM-debug	*comment*	**-comment**
EPM-debug-rule	*comment*	**-comment**
EPM-demote	**-level**	**-target_task**
EPM-display-form	**-form**	**-source_task**
EPM-execute-follow-up	*argument*	**-command**
EPM-fill-in-reviewers	**-level** (legacy syntax)	**-target_task**
	-review_task_name	**-target_task**
	-reviewer	**-assignee**
EPM-hold	**{true | false }**	Argument removed.
EPM-inherit	**PREVIOUS | CALLER | ROOT**	**-task**
		The values are now **$PREVIOUS | $CALLER | $ROOT**.
	TARGET | REFERENCE | SIGNOFFS	**-attachment**
		The values are now lowercase. Multiple values are separated by commas.
EPM-invoke-system-action	**-signoff**	**-comment**
	-system	**-command**
EPM-invoke-system-rule	**debug**	**-debug**
	-signoff	**-comment**
	-system	**-command**
EPM-late-notification	*user*	**-recipient**
EPM-notify	**-comments**	**-comment**

Handler name beginning in Teamcenter 10.1	Argument name prior to Teamcenter 10.1	New argument name beginning in Teamcenter 10.1
	-attachment	-attachment
		The values are now lowercase and the **$** has been removed.
EPM-notify-report	-comments	-comment
EPM-notify-signoffs	-comments	-comment
	log	-log
	-attachment	-attachment
		The values are now lowercase and the **$** has been removed.
EPM-set-condition	$Query	-query
	All | Any | None	-check_targets
		The values are now lowercase.
	-query_type	-query_type
		The values are now lowercase and **sub-process** is now **sub_process**.
EPM-set-duration	day	-day
	hour	-hour
	minute	-minute
	week	-week
	year	-year
EPM-set-parent-result	true | false	-value
EPM-set-property	-exclude_types	-exclude_type
	-from_att_type	-from_attach
	-include_types	-include_type
	-props	-property
	-to_att_type	-to_attach
	-values	-value
EPM-set-rule-based-protection	named-ACL	-acl
EPM-set-status	APPEND | REPLACE | DELETE | RENAME	-action
		The values are now lowercase.
	-f	-status
	RETAIN_RELEASE_DATE	-retain_release_date
	SET_EFFECTIVITY	-set_effectivity
	-t	-new_status
EPM-set-task-result-to-property	-target_type	-include_type
	-task_name	-source_task
	-target	-target_task
EPM-system	argument	-command
EPM-trigger-action	action	-action

Handler name beginning in Teamcenter 10.1	Argument name prior to Teamcenter 10.1	New argument name beginning in Teamcenter 10.1
	comment	-comment
EPM-trigger-action-on-related -process-task	-process_template	-template
EPM-validate-target-objects	-allowed_type	-exclude_type
	-disallowed_type	-include_type
PS-attach-assembly-components	-exclude_types	-exclude_related_type
	-include_types	-include_related_type
TCRS-check-approver	-A_level	-a_task
	-A_user	-a_user
	-A_jobowner	-a_jobowner
	-B_level	-b_task
	-B_user	-b_user
	-B_jobowner	-b_jobowner
TCRS-check-signoff	-level	-task
TCRS-create-snapshot	-RevRule	-revision_rule
	-SnapshotName	-name
	-SnapshotDescription	-description
TCRS-set-bom-precise	-RevRule	-revision_rule

The following keyword names have been changed as indicated:

Keyword name prior to Teamcenter 10.1	New keyword name beginning in Teamcenter 10.1
$PROCESSGROUP	**$PROCESS_GROUP**
$PROCESSOWNER	**$PROCESS_OWNER**
$ROLEINGROUP	**$ROLE_IN_GROUP**
	For more information, see note below.
$ROOTTask	**$ROOT**
ROOT	**$ROOT**
$TARGETGROUP	**$TARGET_GROUP**
$TARGETOWNER	**$TARGET_OWNER**

> **Note**
>
> For the , , and handlers, use **resourcepool:$GROUP::$ROLE** instead of **$ROLE_IN_GROUP**.
>
> For the and handlers, use **allmembers:$GROUP::$ROLE** instead of **$ROLE_IN_GROUP**.

Handler argument values

Syntax for handler arguments and values

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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. For example:

Argument	Values
-relation	**IMAN_specification**
-type	**UGMASTER, UGPART**
-att_type	**target**

> **Note**
>
> - Handler values are case sensitive and must be accurate to the letter.
>
> - If an argument calls for the name of an object, attribute, or property defined in the Business Modeler IDE, it must use the actual name, not its display name.
>
> - If an argument value has a comma in its name, you must use the **EPM_ARG_target_user_group_list_separator** preference to specify another separator for multiple values.
>
>   For example, if you have an **Engineering, Home Office** group and use it as an argument value, you must change the preference to use a different separator character, such as a quotation mark (**"**).

Keywords as argument values

What are 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.

Keyword	Description
$USER	Extracts the user ID of the current user.
$GROUP	Extracts the group ID of the current user.
$ROLE	Extracts the role of the current user.

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.

Keyword	Description
$PROCESS_OWNER	Extracts the user ID of the owner of the current workflow process.
$PROCESS_GROUP	Extracts the group ID of the owner of the current workflow process.
$TARGET_OWNER[[(Class)|Type]]	Extracts the user ID of the owner of the current workflow process's target.
	You can define an optional type or bracketed class in square brackets to specify the type or class of target object from which to extract the owner ID. If you do not define a class or type, the system uses the class of **ItemRevision** by default.
	If the system finds more than one object, it returns the owner ID from the first object.
	For example, **$TARGET_OWNER[(Dataset)]** extracts the owning user ID from the first dataset target found, and **$TARGET_OWNER[UGMASTER]** extracts the owning user ID from the first **UGMASTER** target found.
$TARGET_GROUP[[(Class)|Type]]	Extracts the group ID of the owner of the current workflow process's target. Only the first owner is returned.
	As with **$TARGET_OWNER**, 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.
$TARGET_OWNERS[[(Class)|Type1[,Type2,…]]]	Extracts the user IDs of the owners of the current workflow process's targets. Only the first owner is returned.
	This keyword works the same as **$TARGET_OWNER**, except that it returns a unique comma-separated list of the different owning user IDs from all specified target types.

Keyword	Description
$TARGET_GROUPS[[(Class)|Type1[,Type2,…]]]	Extracts the group IDs of the owners of the current workflow process's targets.
	This keyword works the same as **$TARGET_OWNERS**, except it returns group IDs.
$ROLE_IN_GROUP	Extracts the user's current logged-on group ID and role in the format of a resource string, for example, *group*:*role*.

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.

Keyword	Handlers
$ANALYST	**EPM-adhoc-signoffs**
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$CHANGE_IMPLEMENTATION_BOARD	**EPM-adhoc-signoffs**
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$CHANGE_REVIEW_BOARD	**EPM-adhoc-signoffs**
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify

Keyword	Handlers
$CHANGE_SPECIALIST1	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$CHANGE_SPECIALIST2	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$CHANGE_SPECIALIST3	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$CURRENT_DATE	EPM-set-property
$OWNER	EPM-check-action-performer-role
	EPM-late-notification
$PROCESS	EPM-notify
	EPM-notify-signoffs

Keyword	Handlers
$PROJECT_ADMINISTRATOR	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$PROJECT_AUTHOR	EPM-adhoc-signoffs
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$PROJECT_MEMBER	EPM-adhoc-signoffs
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$PROJECT_TEAM_ADMINISTRATOR	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$PROPOSED_RESPONSIBLE_PARTY	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify

Keyword	Handlers
$PROPOSED_REVIEWERS	EPM-adhoc-signoffs
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$REFERENCE	EPM-attach-related-objects
	EPM-create-form
	EPM-create-relation
	EPM-display-form
	EPM-remove-objects
	EPM-set-property
	EPM-notify
	EPM-notify-signoffs
$RELEASE_STATUS	EPM-create-form
	EPM-create-relation
	EPM-display-form
$RESPONSIBLE_PARTY	EPM-notify-report
	EPM-check-action-performer-role
	EPM-late-notification
	EPM-notify
$REQUESTOR	EPM-adhoc-signoffs
	EPM-auto-assign
	EPM-auto-assign-rest
	EPM-assign-team-selector
	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-notify
$REVIEWERS	EPM-fill-in-reviewers
	EPM-notify-report
	EPM-late-notification
	EPM-notify

Keyword	Handlers
$SIGNOFF	**EPM-create-form**
	EPM-create-relation
	EPM-display-form
$TARGET	**EPM-attach-related-objects**
	EPM-check-target-attachments
	EPM-create-form
	EPM-create-relation
	EPM-display-form
	EPM-remove-objects
	EPM-set-property
	EPM-notify
	EPM-notify-signoffs
$UNDECIDED	**EPM-notify-report**
	EPM-late-notification
	EPM-notify

Use keywords to implement dynamic participants in handlers

You can use the following keywords to invoke dynamic participants:

$ANALYST	**$PROJECT_ADMINISTRATOR**
$CHANGE_SPECIALIST1	**$PROJECT_TEAM_ADMINISTRATOR**
$CHANGE_SPECIALIST2	**$PROJECT_AUTHOR**
$CHANGE_SPECIALIST3	**$PROJECT_MEMBER**
$CHANGE_REVIEW_BOARD	**$REQUESTOR**
$CHANGE_IMPLEMENTATION_BOARD	

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 and users with the specified role.

Configuring assigning participants automatically

You can configure your workflow to automatically assign participants with a set of Business Modeler IDE constants that have conditions as values. You can also use assign participants by adding workflow handlers that use properties that have participants as values.

Workflow constants

A set of constants is provided in the form:

> *<prefix><participant-name>***AssignableCondition**

The variable *<prefix>* is the Business Modeler IDE template prefix and *<participant-name>* is an existing participant name.

> **Note**
>
> If the participant name also has a template prefix, the prefix appears twice.

For example, if the prefix is **Fnd0** and the participant name is **PROPOSED RESPONSIBLE PARTY**, the constant is **Fnd0ProposedResponsiblePartyAssignableCondition**.

The constants are for item revisions and change item revisions.

Workflow conditions

The values of the constants are conditions in the form:

> **is***<participant-name>***Assignable**

For example, if the participant name is **PROPOSED RESPONSIBLE PARTY**, the condition is **isProposedResponsiblePartyAssignable**.

This is used while assigning dynamic participants. Teamcenter gets the value of the *<prefix><participant-name>***AssignableCondition** constant to get the condition name to evaluate before assigning the participant.

Search for condition names

You can search for the constant name given an object type and participant type using pattern matching.

For example, to find a constant associated with an item revision and the **Fnd0MyNewParticipant** participant, search for a constant that ends with **Fnd0MyNewParticipantAssignableCondition**. The actual constant name is **Fnd0Fnd0MyNewParticipantAssignableCondition**.

If there are multiple matches, choose the one which has the same prefix as the prefix of the participant name.

Creating constants and conditions

If you have your own participant types, you must create your own constants and conditions for them.

For example, if your template prefix is **CUS1** and the new participant name is **MyParticipant**:

1. Create a participant named **CUS1MyParticipant**.

2.   Create a constant named **CUS1CUS1MyParticipantAssignableCondition** with a value of **isMyParticipantAssignable**.

The participant creation code looks up the constant and corresponding condition and evaluates it.

Assigning participants with workflow handlers

You can use the following workflow handlers when automatically assigning participants:

- **EPM-assign-responsible-party-dynamic-participant**

- **EPM-assign-signoff-dynamic-participant**

The following handlers can be used to get assignees from a property value:

- **EPM-adhoc-signoffs**

- **EPM-assign-team-selector**

- **EPM-auto-assign**

- **EPM-auto-assign-rest**

- **EPM-fill-in-reviewers**

You can use the **user:PROP::***property_name*, **resourcepool:PROP::***property_name*, or **allmembers:PROP::***property_name* values for the **-assignee** argument to get the name of the assignee from a property of the target, reference, or schedule task.

You can find the object type with the **-include_related_type**, **-exclude_related_type**, **-include_type**, **-exclude_type**, **-from_relation**, and **-from_attach** arguments.

For more information, see the full handler description.

Lists of values as argument values

Using lists of values (LOVs) in 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 Business Modeler IDE; 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.

LOV syntax

Any handler using an LOV accepts the **-lov=**_lov-name_ argument, which specifies the LOV to be used.

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.

> **Note**
>
> The name of an LOV used with a handler can be anything, but the Business Modeler IDE may enforce a particular naming convention, for example, an **M4_** prefix. You can add the handler name as a suffix to help identify LOVs used by handlers.

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.

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).CMHasSolutionItem.(ItemRevision)
.Reference.Affected Item Form
```

The previous example uses three relation pairs, as follows:

Pair	Description
$TARGET.(ItemRevision)	Finds objects of the class **ItemRevision** attached as workflow process targets.
CMHasSolutionItem.(ItemRevision)	For each of the revisions found by the first pair, the system searches the **CMHasSolutionItem** relation to find objects of the class **ItemRevision**.
Reference.Affected Item Form	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**.

The individual handler documentation indicates which handlers accept this syntax.

LOV syntax example

This LOV example can be used with the **EPM-attach-related-objects** handler. Each line is a separate value in the LOV.

Argument	Values
-lov	**M4_EPM_attach_objects**

The **M4_EPM_attach_objects** LOV contains this data:

Value	Description
$TARGET.(ItemRevision).Specification.*	**Attach all objects in target revision Specification relation**
$TARGET.(ItemRevision).Specification.(Dataset). Form.(Form)!UGPartAttr	**Attach all forms attached to datasets in target revision Specification relation**
$TARGET.(ItemRevision).PSBOMViewRevision.*	**Attach all BOM View Revisions in target revision**
$TARGET.(ItemRevision).Manifestation.(Form)	**Attach all forms in target revision Manifestation relation**

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.

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.

Class	Relation	Description
Item	Any GRM relation	Identifies any GRM-related objects attached to items.
		For example: **(Item).IMAN_reference**
	Revisions	Identifies all revisions from items.
		For example, to find all the datasets in the **IMAN_specification** relation of all revisions in any items found:
		(Item).Revisions.*.IMAN_specification. (Dataset)
		Note
		The type of revision is not relevant as there is only one type of revision in any item; therefore, an asterisk (*) is used to specify any type.
	PSBOMView or **BV**	Identifies all BOM views from items.
		For example, to select all BOM views:
		(Item). PSBOMView
		Select only the *view* BOM views:
		(Item).BV.BOMView Revision

Class	Relation	Description
Revision	Any GRM relation	Identifies any GRM-related objects attached to revisions.
		For example, to identify all reference objects from revisions:
		(ItemRevision).IMAN_reference
		Identifies all specification objects in document revisions that are attached as requirements to design revisions:
		Design Revision.IMAN_requirement.Document Revision.IMAN_specification. *
	PSBOMViewRevision or **BVR**	Identifies all BOM view revisions from revisions.
Dataset	Any GRM relation	Identifies any GRM-related objects attached to datasets.
		For example:
		(Dataset).IMAN_Rendering
	Any reference	Identifies any objects attached as references to datasets, such as **UGPART-ATTR** forms attached to **UGMASTER** and **UGPART** datasets.
		For example:
		(Dataset).UGPART-ATTR
Folder	*	Identifies objects in folders.
		For example, to identify all revisions in a folder:
		(Folder).*.(ItemRevision)
Job	**$TARGET** or **Targets**	Identifies targets attached to a job.
		For example:
		(Job).$TARGET
	$REFERENCE or **References**	Identifies targets attached to a job.
		For example:
		(Job).$REFERENCE

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.

Action Handlers

AI-export-AH

DESCRIPTION

This handler has two modes of operation, depending on whether the required **type** argument is used with or without additional arguments.

- When **type** is the only argument:

  o When there already is an **AIObject** in the reference attachments, this handler does nothing.

  o When there is initially no **AIObject** in the reference attachments, this handler creates a new **AIObject** of the specified type and a new **CCObject** of type **CCObject** and name **ERPObject**. The handler creates a **StructureContext** for each **ItemRevison** found in the target attachments. The **Latest Working** revision rule is used in the **StructureContext** that is attached to the **CCObject**.

- When **type** is specified with at least one of the available optional arguments:

  o Exports the objects found in target attachments to one or more **AIObjects**, based on the settings of the optional arguments.

  o Searches the reference attachments for an **AIObject** of the type specified by the **type** argument.

  ▪ When an **AIObject** is found, it is used. Otherwise this handler creates an **AIObject** of the specified type.

  The objects attached to the targets attachments can be filtered by the list of types specified by **targetTypes** argument.

  The types listed must be one of the following supported types:

  ◊ **ItemRevision**
  ◊ **Item**
  ◊ **PSBOMView**
  ◊ **PSBOMViewRevision**
  ◊ **CCObject**
  ◊ **AppearanceGroup**

  If a **targetTypes** value is not provided, then all types are included.

  o If the **multipleAI** value is equal to **1**, the handler creates an **AIObject** for each object in the target attachments.

  o If the **multipleAI** value is equal to **0** and **createRequests** is equal to **1**, the handler creates a single **AIObject** with a new **RequestObject** for each object in the target attachments.

  o If **createCC** is equal to **1**, the handler creates a **CCObject** of the type specified by the **ccType** argument for non CC/SC objects in the target attachments, and exports the **CCObject**.

SYNTAX

AI-export-AH -type=*ai-object-type* [**-multipleAI=** *0* |*1*] [**-createCC=** *0* |*1*]

[**-ccType=** *cc-object-type*] [**-createRequests=** *0 |1*]
   [**-targetTypes=** *delimited list of object types by which to filter target attachments*]

ARGUMENTS

-type

The type of **AIObject** to search for in the reference attachments or, if none are found, the type of **AIObject** to be created. The created **AIObject** is attached to the root task.

-multipleAI

If equal to **0**, creates a single **AIObject**. This is the default value.

If equal to **1**, creates an **AIObject** for each object found in the target attachments.

-createCC

If set equal to **1**, creates a **CCObject** with the type specified in the **-ccType** argument. The default value is **0**.

-ccType

The type of **CCObject** to be created.

-createRequests

If **-multipleAI** is equal to **0** and **-createRequests** is equal to **1**, this handler creates a single **AIObject** with a new **RequestObject** for each object in target attachments. The default value is **0**.

-targetTypes

Uses a delimited list of object types for filtering target attachments. The types listed must be of the following supported types: **ItemRevision**, **Item** | **PSBOMView** | **PSBOMViewRevision** | **CCObject** | **AppearanceGroup**.

The delimiter can be a colon (**:**) or a comma (**,**).

If no types are provided, all types are considered without filtering.

PLACEMENT

This handler can be placed on any task.

RESTRICTIONS

None.

EXAMPLES

Select an **ItemRevision** and submit to a workflow with this handler. This handler creates and exports the **AIObject**, and then attaches it to the root task.

Argument	Values
-type	NX_AI
-createCC	1
-ccType	CCObject

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.

VAL-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 **VAL-check-validation-result-with-rules** rule handler.

SYNTAX

VAL-set-condition-by-check-validation-result
-rule_item_revision=item-revision-id* **[-current_event=**value**]**
[-pass_item_revision_only] [-ref_log]

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.

-ref_log

If this argument is present and the validation fails, the validation results log is created and the log is attached, but no warning is displayed.

If this argument is not present and the validation fails, the validation results log is created, the log is *not* attached, and no warning message is displayed.

If the validation passes, the validation results log is not created and no message is displayed.

PLACEMENT

Place under the **Complete** action.

RESTRICTIONS

- **-rule_item_revision** cannot be NULL.

- You cannot customize the path names that branch from the **Condition** task. They must be either **T** or **F**.

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-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.

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.
>
> The target of the handler must be an item revision. The handler traverses the item revision to look for the dataset that was specified in the handler definition.

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.

TCRS-trigger-approve-first-step

DESCRIPTION

Initiates 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).

SYNTAX

TCRS-trigger-approve-first-step

ARGUMENTS

None.

PLACEMENT

Place on the **Start** action of the root task.

RESTRICTIONS

None.

TCRS-store-review-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

TCRS-store-review-data -name=*attribute-name* **-date=***attribute-name-date*
[-mode=signoff | owner | modifier | delete] [-dest=IRM | UGPartAttr] [-person]

ARGUMENTS

Parameter	Description	Required
-name	Stores the approver's logon name. This attribute should be of type string and should have sufficient length.	Yes
-date	Stores the approval date of the task. This attribute should be of type string and should have sufficient length.	Yes
-mode	Valid values are:	No
	• **signoff**: Approver and date approved of the current level. This is the default value.	
	• **owner**: Owners and date created.	
	• **modifier**: Last modified user and modification date.	
	• **delete**: Previous attribute contents will be deleted.	
-dest	Defines the destination form type. Valid values are **IRM** (item revision master form) and **UGPartAttr** (UGPartAttribute form). **IRM** is the default value.	No
-person	If this parameter is used, the actual person name of the signoff person is used instead of the user ID. The values for the argument are any of the person's attributes, such as address, city, state, zip code, or country.	No

PLACEMENT

Set in the **Complete** action. If **-mode=signoff**, set in the **Complete** action of the **perform-signoffs** task.

Set in the **Undo** or **Start** action. If **-mode=delete** , set in the **Undo**or **Start** action of the **perform-signoffs** task. Placement on the **Undo** action is done on a **Review** task.

> **Note**
>
> The workflow can be designed so when a task is demoted on the **Reject** action, this handler is called. This can be achieved by placing **EPM-demote-on-reject** handler on the **Perform** action of the **perform-signoffs** task. A **Reject** action causes a demotion to the previous task, invoking the **Undo** action.

RESTRICTIONS

All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

This example shows how to store the workflow approver's information into the item revision master form.

Argument	Values
-name	**user_data_1**
-date	**user_data_3**
-mode	**signoff**

TCRS-setstatus-EngOrder-folder

DESCRIPTION

For Teamcenter Rapid Start, 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.

For information for applying a status to other objects, see the **EPM-set-status** handler.

SYNTAX

TCRS-setstatus-EO-folder [-eo_folder=relation-name**] -status=**status-name **-type=**EO-revision-type

ARGUMENTS

Parameter	Description	Default	Required
-eo_folder	Relation name of the engineering order pseudofolder. For example, the relation name of the **New Parts** folder is **TCX_New_Parts**.	**TCX_Obsolete_Parts**	No
-status	Status for the engineering order.	**90**	Yes
-type	Type of the engineering order revision (for example, **Eng_Order Revisions**).	**Revision**	Yes

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.

Argument	Values
-eo_folder	**TCX_Obsolete_Parts**
-status	**90**
-ype	**Eng_Order Revisions**

TCRS-set-bom-precise

DESCRIPTION

Switches all target BOM view revisions to precise or imprecise.

SYNTAX

TCRS-set-bom-precise [-revision_rule=config-rule**] [-precise=true|false]**

ARGUMENTS

Parameter	Description	Default	Required
-revision_rule	Name of the configuration rule. **Examples:** • Latest Released • Latest by Creation Date • Precise; Working	Default configuration rule of the user.	No
-precise	Set to true for precise BOM view revisions or false for imprecise BOM view revisions	true	No

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.

EXAMPLES

This example sets the target BOM view revisions to **-precise** with a revision rule of **Latest Released**.

Argument	Values
-revision_rule	**Latest Released**
-precise	**true**

TCRS-remove-targets-with-status

DESCRIPTION

Allows you to remove target objects with specified status from the workflow process.

SYNTAX

TCRS-remove-targets-with-status -status= *status-name*

ARGUMENTS

Parameter	Value	Required
-status	Status of objects to remove.	Yes

PLACEMENT

Place on the **Start** action of the root task.

RESTRICTIONS

None.

EXAMPLES

This example removes all objects with a status of 60 from the workflow process.

Argument	Values
-status	**60**

TCRS-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

TCRS-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

Parameter	Description	Default	Required
-status	Assigns a release status. If this parameter is set to **$NONE**, you can start a workflow on the previous revision without assigning a status.	None	Yes
-rev_status	Use commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference 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.	None	No
-latest	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.	None	No
-proc_name	Name of the workflow that will start according to the item revisions.	None	No
-job_name	Job name for this workflow.	*job_timestamp*	No
-job_desc	Job description for this workflow.	Empty string	No

PLACEMENT

Place before the **ADD** status (preferably in the **Start** action of the **Add Status** task).

RESTRICTIONS

None

EXAMPLES

This example releases all previous item revision with a status of 60.

Argument	Values
-status	**60**

This example releases the previous item revision which has a latest status of 30, with a status of 60.

Argument	Values
-status	**60**
-rev_status	**30**
-latest	

This example releases the previous item revision which has a latest status of 30, with a status of 60.

Argument	Values
-status	**60**
-proc_name	**New Process 1**

TCRS-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

TCRS-purge-dataset

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

All datasets that require purging must be a target to the EPM task.

TCRS-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 **EXPRESS_IRM_cleanfields***release*, where *release* is the value defined in the **-block** parameter. For example, define the **EXPRESS_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 run, the values stored in the **Release No** and **Release text** fields of the item revision master form are deleted.

SYNTAX

TCRS-IRM-cleanfields -block=*blockname*

ARGUMENTS

Parameter	Value	Default	Required
-block	Any value.		Yes

PLACEMENT

Requires no specific placement.

RESTRICTIONS

All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

Argument	Values
-block	**release**

TCRS-export-to-tcxmlfile

DESCRIPTION

Exports targets and references to a TC XML file.

SYNTAX

TCRS-export-to-tcxmlfile -option_set=*export-option-set* **-filename=***export-file-name* **[-attach= target | reference | both]**

ARGUMENTS

Parameter	Description	Required
-option_set	Specify the name of the transfer option set when when exporting the objects.	Yes
-filename	TC XML file name for the exported objects.	Yes
-attach	Object attachments to be exported from the process target and/or reference folder. **target | reference | both**	No

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EXAMPLES

This example shows how to export targets of the workflow to TC XML file with name of **"ExportFileName"**, using the **TIEConfiguredExportDefault** transfer option set.

Argument	Values
-option_set	**TIEConfiguredExportDefault**
-filename	**ExportFileName**
-attach	**target**

TCRS-export-signoff-data

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 Solid Edge draft files. This handler is dependent on a Solid Edge conversion utility, which can be downloaded from GTAC at the following location:
>
> *https://download.industrysoftware.automation.siemens.com/solid_edge/ SEEC_Workflow_PDF_Generation/*
>
> Use the short path convention in the values of **SE_TO_PDF_WORKING_DIR** and **SE_TO_PDF_EXECUTABLE_DIR**.
>
> This workflow handler can be used in both Teamcenter and Teamcenter Rapid Start workflows.

SYNTAX

TCRS-export-signoff-data [-person] [-tif] [-replace]

ARGUMENTS

Parameter	Description	Required
-person	Prints the person name on the PDF file instead of the user name.	No
-tif	Generates a TIF dataset instead of a PDF dataset.	No
-replace	Replaces any existing PDF dataset that may have been created by a previous execution of this handler.	No

PLACEMENT

This handler must be placed after a **Release** task template. The item revision must be released before this handler can be run. 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 run.

• Create the following preferences before using this handler:

  o **SE_TO_PDF_WORKING_DIR**

  Specifies the staging location where the **PDF-generation** utility is run.

For example:

**SE_TO_PDF_EXECUTABLE_DIR=C:\Progra~1\Solide~1\Program\
SEEC_WorkFlow_PDF_Generation.exe**

o   **SE_TO_PDF_EXECUTABLE_DIR**

Specifies the location of the SEEC_WorkFlow_PDF_Generation conversion
utility.

For example:

**SE_TO_PDF_EXECUTABLE_DIR=C:\Progra~1\Solide~1\Program\
SEEC_WorkFlow_PDF_Generation.exe**

> **Note**
>
> Use the short path convention in the
> values of SE_TO_PDF_WORKING_DIR and
> SE_TO_PDF_EXECUTABLE_DIR.

o   **SE_PDF_GEN_WAITING_PERIOD**

Specifies the number of seconds the handler should wait for the PDF file
to be generated.

For example:

SE_PDF_GEN_WAITING_PERIOD=20

•   Set the following preferences to the indicated values:

o   **TC_audit_manager=ON**

o   **TC_audit_manager_version=3**

EXAMPLES

In following example `-person` and `-replace` arguments are specified. This converts the
attached 2D draft dataset into PDF. It also retrieves the workflow signoff information,
such as the signoff user name, signoff date, and so on. The signoff user name is
replaced by the person name of signoff user and the generated PDF has the person
name in the title block.

Argument
-person
-replace

TCRS-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

TCRS-delete-log-datasets [-name=*name***] [-desc=***description***]**

ARGUMENTS

Parameter	Value	Required
-name	Name of the dataset.	Yes, if **-desc** is not provided.
-desc	Description of the dataset.	Yes, if **-name** is not provided.

PLACEMENT

Place the handler on an action of the root task that occurs after the datasets are created.

RESTRICTIONS

None.

EXAMPLES

Syntax	Description
TCRS-delete-log-datasets -name=CheckBomChildStatus	This example shows how to delete all log datasets with the name **CheckBomChildStatus**.
TCRS-delete-log-datasets -desc= HANDLER_LOG	This example shows how to delete all log datasets with the description **HANDLER_LOG**.
TCRS-delete-log-datasets -name=CheckBomChildStatus -desc= HANDLER_LOG	This example shows how to delete all log datasets with the name **CheckBomChildStatus** and the description **HANDLER_LOG**.

TCRS-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

TCRS
-delete-dataset -dataset_type=*dataset-type*
-relation=*relation-type*
[-reference=*named-reference-type***]**

ARGUMENTS

Parameter	Value	Required
-dataset_type	Defines dataset type to be deleted.	Yes
-relation	Defines relation between target object and specified dataset type.	Yes
-reference	Defines named reference type of the dataset to be deleted.	No

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.

> **Note**
>
> If you delete a dataset, it should be added as a workflow target.
>
> If you delete a named reference, the dataset containing the reference should be added as a workflow target.

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

This example shows how to delete Text named reference of Text dataset with IMAN_reference relation to target object.

Argument	Values
-dataset_type	**Text**
-relation	**IMAN_reference**
-reference	**Text**

TCRS-Create-Translation-Request

DESCRIPTION

For Teamcenter Rapid Start, 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

TCRS-Create-Translation-Request -pr [-tr=ONDEMAND | CHECKIN | IMPORT] [-MS=msoffice-translation**] [-UG=**nx-translation**] [-SE=**solid-edge-translation**]**

ARGUMENTS

Parameter	Value	Default	Required
-pr	The value can be 1 to 5. The greater the number, the higher the priority in the translation schedule.	**3**	No
-tr [ONDEMAND | CHECKIN | IMPORT]	Categorizes the reason for the translation request.	**ONDEMAND**	No
-MS	Defines Microsoft Office translations. For example, to convert Microsoft Office to PDF, the value should be **pdf**.	**pdf**	No
-UG	Defines NX translations. For example, to convert NX to PDF, the value should be **pdf**. Valid values are: • **cgm** • **hpg** • **jt** • **pdf** • **tif**	**hpg**	No

Parameter	Value	Default	Required
-SE	Used to define Solid Edge translations. For example, to convert Solid Edge to PDF, the value should be **pdf**. Valid values are:	**jt**	No

- **bmp**

- **dwg**

- **dxf**

- **emf**

- **igs**

- **jpg**

- **jt**

- **pdf**

- **plmxml**

- **sat**

- **step**

- **stl**

- **tif**

- **xgl**

- **xt**

PLACEMENT

Place on the **Start** action of the root task.

RESTRICTIONS

None.

EXAMPLES

This example shows how to create a translation request for all the Solid Edge datasets attached to the item revision to PDF file.

Argument	Values
-SE	pdf

TCRS-create-snapshot

DESCRIPTION

Creates a snapshot of the target BOM view revision. It adds a folder as a reference under the job and as a reference under the item revision.

SYNTAX

TCRS-create-snapshot -revision_rule=*rule-name* **-name=***snapshot-name* **[-description=***snapshot-description***]**

ARGUMENTS

Parameter	Value	Default	Required
-revision_rule	Defines the name of the revision rule to be applied for BOM traversal.		Value
-name	Defines the name of snapshot.	"Snapshot"+Object name	Yes
-description	Defines the description of snapshot.		No

PLACEMENT

Requires no specific placement.

RESTRICTIONS

All item revisions must have write privileges at the level that the handler is used.

EXAMPLES

This example shows how to create snapshot with name **'MySnapshot'**, using **Precise** or **Latest Working** revision rule.

Argument	Values
-revision_rule	**Precise; Latest Working**
-name	**MySnapshot**

TCRS-Create-Print-Requests

DESCRIPTION

For Teamcenter Rapid Start, 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

TCRS-Create-Print-Requests [-printername] [-watermark]

ARGUMENTS

Parameter	Value	Default	Required
-printername	Defines the name and path to the printer.	The default printer name from the **TcX_Server_ Printers** preference.	No
-watermark	Specifies the watermark text for the printed output.		No

PLACEMENT

Place on the **Complete** action.

RESTRICTIONS

None.

EXAMPLES

This example shows how to print the datasets that are attached to target object of workflow with the watermark "My water mark."

Argument	Values
-printername	My printer
-watermark	My water mark

TCRS-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

TCRS-create-form -form_type=form [-rev_type=item_rev_type] [-description=description] [-name=name] [-relation] [-separator=separator]

ARGUMENTS

Parameter	Description	Default	Required
-form_type	Valid type of form.	None	Yes
-rev_type	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.	**ItemRevision**	No
-description	Description of the form.	Empty string	No
-name	Name of the form to be created. If this parameter is not specified, the default form name is used.	**Item_id** + separator + **Rev_ID**	No
-relation	Relation used to attach the form to the item revision. This must be a valid relation type between a form and a revision.	**IMAN_reference**	No
-separator	Separator between the item ID and revision ID if the parameter name was not indicated.	Minus sign (-)	No

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.

Only one form of a **form_type** can be created and attached to the target revision by this handler. If the item revision already a form of the specified **form_type** attached, a new form of that **form_type** cannot be added.

EXAMPLE

This example shows how to create a form of My Form Type, form name MyForm, and attach the form to target item revision using EPM_reference relation.

Argument	Values
-form_type	**My Form Type**
-name	**MyForm**

Argument	Values
-rev_type	**ItemRevision**
-relation	**IMAN_reference**

TCRS-auto-approve-first-step

DESCRIPTION

Automatically approves the first task with this handler attached. Use this handler only when the **TCRS-trigger-approve-first-step** handler is placed on the root task. This handler only works the first time. If the task starts again, Teamcenter Rapid Start does not auto approve the first task.

SYNTAX

TCRS-auto-approve-first-step

ARGUMENTS

None.

PLACEMENT

Place on the **Start** action of the **perform-signoffs** task.

RESTRICTIONS

None.

SMP-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*.

SERVICEPROCESSING-approve-service-structure

DESCRIPTION

Runs an approval process for SLM service structures.

SYNTAX

SERVICEPROCESSING-approve-service-structure

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

Use only for approval of SLM service structures inheriting from a transaction element.

SERVICEFORECASTING-approve-ma-extension

DESCRIPTION

Approves a change in a maintenance action due date in Service Scheduler.

SYNTAX

**SERVICEFORECASTING-approve-ma-extension
-prop=ssf0ExtensionApproval -value=Approved**

ARGUMENTS

-prop

Specifies the property to be updated. The only valid property for this handler is
ssf0ExtensionApproval.

-value

Specifies the value for the property. The only valid value for this handler is **Approved**.

PLACEMENT

Place on the **Start** action of a task that follows the approval path of a **Review** task.

RESTRICTIONS

None.

EXAMPLES

- Approves the request to change a maintenance action due date.

Argument	Values
-prop	**ssf0ExtensionApproval**
-value	**Approved**

SCHMGT-sync-schedule-attachments

DESCRIPTION

Synchronizes the change attachments of the parent schedule task with the workflow's change attachments. The change attachments of the schedule tasks are the same as that of the workflow after executing this handler if no error is encountered during the operation.

This handler works with remote schedule tasks only. The workflow does not inherit the change relations for local schedule tasks.

SYNTAX

SCHMGT-sync-schedule-attachments [-attachment= *attachment-types* **]**

ARGUMENTS

-attachment

(Optional) Specify one or more of the following change attachment types to synchronize.

*   **problem_item**

*   **solution_item**

*   **impacted_item**

Separate multiple attachment types with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

If this argument is not specified, all three change attachments types are synchronized.

PLACEMENT

Place on the **Start** or **Complete** action of any task. Do not place on the **Perform** action.

Because this handler invokes Multi-Site operations, Siemens PLM Software recommends that you place this handler on a task marked for background processing.

RESTRICTIONS

None.

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

None.

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.

SCHMGT-reject-timesheetentries

DESCRIPTION

Retrieves the target objects, the scheduled task, and the corresponding schedule, in the reject branch of 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-reject-timesheetentries

ARGUMENTS

None.

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 along the reject path. Adding this handler to any other workflow process template causes the workflow process to fail.

SCHMGT-approve-timesheetentries

DESCRIPTION

Retrieves the target objects, the scheduled task, and the corresponding schedule, in the approve branch of 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

None.

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 along the approval path. Adding this handler to any other workflow process template causes the workflow process to fail.

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.

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.

RM-attach-tracelink-requirement

DESCRIPTION

Sends requirements tracelinked to Teamcenter objects in the **Targets** folder to the specified folder in the workflow assignee's worklist.

SYNTAX

RM-attach-tracelink-requirement
[-defining_complying_type=defining | complying]
[-folder_type=target | reference] [-tracelink_subtype=subtype**]**

ARGUMENTS

-defining_complying_type

Specifies if the **defining** or **complying** requirement is sent.

-folder_type

Specifies if the requirement is placed in the task's **target** or **reference** folder in the worklist.

-tracelink_subtype

Sends only the specified subtype of the tracelink object.

PLACEMENT

Place on the **Start** action of the root task of the workflow process.

RESTRICTIONS

None.

EXAMPLES

• This example sends the defining requirement linked to Teamcenter objects in the **Targets** folder with a tracelink to the **Targets** folder of the **Tasks to Perform** folder of the assignee's worklist.

Argument	Values
-defining_complying_type	**defining**
-folder_type	**target**

• This example sends the defining requirement linked to Teamcenter objects in the **Targets** folder with a tracelink to the **References** folder of the **Tasks to Perform** folder of the assignee's worklist.

Argument	Values
-defining_complying_type	**complying**
-folder_type	**reference**

RM-attach-SM-tracelink-requirement

DESCRIPTION

Sends requirements tracelinked to Schedule Manager tasks to the specified folder in the task assignee's worklist.

This action handler is implemented to attach defining or complying objects using the trace links on predecessor tasks.

SYNTAX

RM-attach-SM-tracelink-requirement
[-defining_complying_type=defining | complying]
[-folder_type=target | reference] [-tracelink_subtype=*subtype***]**

ARGUMENTS

-defining_complying_type

Specifies if the **defining** or **complying** requirement is sent. If this argument is not specified, **defining** is the default.

-folder_type

Specifies if the requirement is placed in the task's **target** or **reference** folder in the worklist. If this argument is not specified, **target** is the default.

-tracelink_subtype

Sends only the specified subtype of the tracelink object.

PLACEMENT

Place on the **Start** action of the root task of the workflow process.

RESTRICTIONS

This handler is implemented only for **RequirementRevision**, **ParagraphRevison**, and **RequirementSpecRevision** and its subtypes.

EXAMPLES

• This example sends a Schedule Manager task linked to a requirement with a tracelink to the **Tasks to Perform** folder of the assignee's worklist and places the defining requirement object in the task's **Targets** folder.

Argument	Values
-defining_complying_type	**defining**
-folder_type	**target**

• This example sends a Schedule Manager task linked to a requirement with a tracelink to the **Tasks to Perform** folder of the assignee's worklist and places the complying requirement object in the task's **References** folder.

Argument	Values
-defining_complying_type	**complying**
-folder_type	**reference**

RDV-tessellation-handler

DESCRIPTION

Tessellates NX datasets. It identifies which datasets to tessellate by reading the targets set in the **EPM_tessellation_target_type** 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

RDV-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

Runs 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 tesselation, do the following:

  o For asynchronous background tessellation, use the **-background** argument and place on the **Complete** action of the root task after the **EPM-set-status** handler.

  o 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.

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.

ENVIRONMENT VARIABLES

You must set the following environment variables before running the tessellation process with this action handler:

- *UGII_ROOT_DIR*
-

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

RDV-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 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 requires NX to be installed on all systems on which the handler runs. In a 2-tier environment, NX must be installed on all clients that run this workflow handler. In a four-tier environment, handlers run in the **tcserver** process, so NX must also be installed onto the enterprise tier servers (pool servers). The environment variables **UGII_BASE_DIR** and **UGII_ROOT_DIR** (normally set by the NX installation) are used to determine the location of the NX software. This example depicts the two environment variables set to NX on a Windows platform.

```
set UGII_BASE_DIR = c:\apps\nx75

set UGII_ROOT_DIR = c:\apps\nx75\ugii\
```

SYNTAX

RDV-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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. 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

Argument	Values
-type	**UGMASTER, UGPART**

Argument	Values
-text	**text**

RDV-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

RDV-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.

RDV-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

RDV-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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. 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

Argument	Values
-type	**UGMASTER, UGPART**

PUBR-unpublish-target-objects

DESCRIPTION

Unpublishes target objects (removes them) from the ODS.

SYNTAX

PUBR-unpublish-target-objects [-class=classname**] [-site=**site-ID**]**

ARGUMENTS

-class

Teamcenter classname of the target objects being unpublished. 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:

Argument	Values
-class	**Item, Dataset**

PUBR-publish-target-objects

DESCRIPTION

Publishes target objects (that is, enters them) in the Object Directory Services (ODS) database.

SYNTAX

PUBR-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.

• You can control the publication behavior of item revision objects by changing the setting of the **TC_publish_item_or_itemrev** preference. You can publish only the item revision object, only its parent item object, or both.

EXAMPLES

This example shows how to publish all item revision target objects to **Detroit** and **Tokyo** ODSs:

Argument	Values
-class	**ItemRevision**
-site	**Detroit, Tokyo**

PS-make-mature-design-primary

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

PS-make-mature-design-primary

ARGUMENTS

None.

PLACEMENT

Preferably placed on the **Complete** action.

RESTRICTIONS

Considers only item revisions or a subclass of them.

PS-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** 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 **WRKFLW_allow_replica_targets** preference is set to **true** and if any replica object qualifies to be attached as a workflow target, that object is attached as a **Replica Proposed Target** to the workflow process.
>
> If the preference is set to **false** or is undefined, the handler attaches replica objects as references instead of targets.

> **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

PS-attach-assembly-components [-depth=*depth-of-traversal***]**
[-owned_by_initiator][-owned_by_initiator_group] [-initiator_has_write_prev]
[[-exclude_released [-traverse_released_component]]] [-rev_rule=*revision-rule***]**
[-saved_var_rule=*saved-variant-rule* **]**
[[-exclude_related_type=*types-to-be-excluded***] |**
[-include_related_type=*types-to-be-included***]] [-add_excluded_as_ref]**
[-include_in_process_targets]

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_related_type

Defines the types to be excluded from being added as targets.

The **-exclude_related_type** and **-include_related_type** 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_related_type

Defines the types to be included as targets.

The **-exclude_related_type** and **-include_related_type** 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 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:

Argument	Values
-depth	**1**

- 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:

Argument	Values
-rev_rule	**Working**
-saved_var_rule	**GMC 300 Rule**

- 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:

Argument	Values
-owned_by_initiator	

- This example releases an assembly using the default revision rule and the default saved variant rule, releasing only the components owned by the group to which the workflow process initiator belongs:

Argument	Values
-owned_by_initiator_group	

- 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:

Argument	Values
-initiator_has_write_prev	

- 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:

Argument	Values
-depth	**all**
-rev_rule	**Latest Released**
-exclude_released	
-add_excluded_as_ref	

- 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:

Argument	Values
-depth	**all**
-rev_rule	**Latest Released**
-exclude_released	
-traverse_released_component	
-add_excluded_as_ref	

- 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:

Argument	Values
-exclude_related_type	**CORP_Build**

- 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:

Argument	Values
-include_related_type	**CORP_Build**

- This example releases an assembly containing targets already in process. This argument can only be used if the **EPM_multiple_processes_targets** preference is set to **ON**.

Argument	Values
-include_in_process_targets	

- 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:

Argument	Values
-depth	**all**
-rev_rule	**Latest Released**
-exclude_released	
-traverse_released_component	
-add_excluded_as_ref	
-exclude_related_type	**CORP_Build**

ADDITIONAL INFORMATION

This handler attaches component item revisions of the assembly to the workflow process. Therefore, you should not place the **EPM-disallow-adding-targets** handler before this handler.

Care should be taken when using this handler in conjunction with the **EPM-check-status-progression** and **PS-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.

PROJ-update-assigned-projects

DESCRIPTION

Updates the list of projects to which the workflow target objects are assigned. The handler arguments determine project names to be assigned to and removed from the targets. You can assign and remove projects using handler arguments only, using properties on a form attached to the workflow template, and using a combination of handler arguments and form properties.

> **Note**
>
> The ability to assign or remove a project is controlled by the **TC_project_validate_conditions** preference, the Access Manager privileges **Assign to Project** and **Remove from Project**, and whether you are a privileged or non-privileged member of the project.

SYNTAX

PROJ-update-assigned-projects [-source_task=*task-name.attachment-type***]**
[-type=*form_type_name***]**
[-assign_property=*property_name***] [-remove_property=***property_name***]**
[-assign_projects=*comma_separated_project_list***]**
[-remove_projects=*comma_separated_project_list***]**
[-bypass]

ARGUMENTS

-source_task

Specifies the task-name and attachment-type combination that associates a source form with the EPM task. The default reference attachments are those that are attached to the current task and are of the type specified by the **-type** argument.

task-name	Use one of the following values:
	• The name of the current task (the default value)
	• The **$ROOT** reserved keyword (the root task)
attachment-type	Use one of the following reserved keywords:
	• **$REFERENCE** for reference attachments
	• **$TARGET** for target attachments

-type

Specifies the type name of a form that contains project names to assign or remove from the target objects.

-assign_property

Specifies the name of a source-form property that designates projects to assign to the target objects.

If you use this argument, you must use the **-type** argument also.

-remove_property

Specifies the name of a source-form property that designates projects to remove from the target objects .

If you use this argument, you must use the **-type** argument also.

-assign_projects

Specifies a list of projects to assign to the target objects. Projects already assigned to a particular target remain assigned.

Separate multiple entries with commas.

-remove_projects

Specifies a list of projects to remove from the target objects. Projects not already assigned to a particular target remain unassigned.

Separate multiple entries with commas.

-bypass

Specifies that Access Manager access checks are bypassed for reading the source form and for writing the target objects. Otherwise, you must have both read access to the source form and write access to the target objects.

> **Note**
>
> If you use this argument, you must have the Access Manager privileges **Assign to Project** and **Remove from Project** for each project assigned to or removed from the target objects.

PLACEMENT

Place on any task action.

RESTRICTIONS

None

EXAMPLES

- This example assigns and removes projects from the target objects using handler arguments only. In this example, assume that the projects to be assigned are **Proj1** and **Proj2**, and that the projects to be removed are **Proj3** and **Proj4**.

Argument	Values
-assign_projects	**Proj1,Proj2**
-remove_projects	**Proj3,Proj4**

- This example assigns and removes projects from the target objects using properties of a form attached to the workflow template. In this example, assume the following:

  o The source form is associated with the root task as a reference attachment.

  o The form type is **Pwf0AssignProjForm**.

    o   The projects to be assigned are listed in the value of the **pwf0AssignProjects** form property.

    o   The projects to be removed are listed in the value of the **pwf0RemoveProjects** form property.

Argument	Values
-source_task	**$ROOT.$REFERENCE**
-type	**Pwf0AssignProjForm**
-assign_property	**pwf0AssignProjects**
-remove_property	**pwf0RemoveProjects**

- This example assigns and removes projects from the target objects using a combination of handler arguments and form properties. In this example, assume the following:

    o   The source form is associated with the root task as a reference attachment.

    o   The form type is **Pwf0AssignProjForm**.

    o   The projects to be assigned are **Proj2** and those that are listed in the value of the **pwf0AssignProjects** form property.

    o   The projects to be removed are **Proj4** and those projects that are listed in the value of the **pwf0RemoveProjects** form property.

Argument	Values
-source_task	**$ROOT.$REFERENCE**
-type	**Pwf0AssignProjForm**
-assign_projects	**Proj2**
-remove_projects	**Proj4**
-assign_property	**pwf0AssignProjects**
-remove_property	**pwf0RemoveProjects**

PROJ-assign-members

DESCRIPTION

Adds members to projects. You can specify the projects and the members using handler arguments only, using properties on a form attached to the workflow template, and using a combination of handler arguments and form properties.

- The list of projects to receive new members is specified directly by projects and indirectly by the **projects_property** argument.

- The list of nonprivileged members to be added to the projects is specified directly by members and indirectly by the **members_property** argument.

- The list of privileged members to be added to the projects is specified directly by privileged_members and indirectly by the **privileged_members_property** argument.

> **Note**
>
> To run this handler, you must be either the project administrator, or the project team administrator of each project receiving new members.

SYNTAX

PROJ-assign-members [-source_task=*task-name.attachment-type***]**
[-type=*form_type_name***]**
[-projects=*comma_separated_project_list***]**
[-members=*comma_separated_member_list***]**
[-privileged_members=*comma_separated_member_list***]**
[-projects_property=*property_name***]**
[-members_property=*property_name***]**
[-privileged_members_property=*property_name***]**
[-bypass]

ARGUMENTS

-source_task

Specifies the task-name and attachment-type combination that associates a source form with the EPM task. The default reference attachments are those that are attached to the current task and are of the type specified by the **-type** argument.

task-name   Use one of the following values:

- The name of the current task (the default value)

- The **$ROOT** reserved keyword (the root task)

attachment-type   Use one of the following reserved keywords:

- **$REFERENCE** for reference attachments

- **$TARGET** for target attachments

-type

Specifies the form type that designates properties to be used as the source of project names and member references.

-projects

Specifies a list of project names to receive new members. The privileged and non-privileged members are added to each project. Members already assigned to a particular project remain assigned.

Separate multiple entries with commas.

-members

Specifies a list of members to be added to the projects as non-privileged members. Each member is of the form group/role/user. An empty value can be specified for group, role, or user when necessary.

Separate multiple members with commas. Separate sub-groups with a period.

-privileged_members

Specifies a list of members to be added to the projects as privileged members. Each member is of the form group/role/user. An empty value can be specified for group, role, or user when necessary.

Separate multiple members with commas.

-projects_property

Specifies the name of a source-form property that designates project names to receive new members. The privileged and non-privileged members are added to each project. Members already assigned to a particular project remain assigned.

If you use this argument, you must use the **-type** argument also.

-members_property

Specifies the name of a source-form property that designates member references to be added to the projects as non-privileged members.

If you use this argument, you must use the **-type** argument also.

-privileged_members_property

Specifies the name of a source-form property that designates member references to be added to the projects as privileged members.

If you use this argument, you must use the **-type** argument also.

-bypass

Specifies that Access Manager access checks are bypassed for reading the project name and member references from the source form. Otherwise, you must have access to read properties from the source form.

PLACEMENT

Place on any task action.

RESTRICTIONS

None

- This example adds members to projects using handler arguments only. In this example, assume the following:

  o   The projects to receive members are named **Proj1** and **Proj2**.

  o   The user named **john** is to be added to both projects as a non-privileged member. This user has the **Designer** role in the **Engineering** group.

  o   The user named **jane** is to be added to both projects as a privileged member. This user has the **Manager** role in the **Engineering** group.

Argument	Values
-projects	**Proj1,Proj2**
-members	**Engineering/Designer/john**
-privileged_members	**Engineering/Manager/jane**

- This example adds members to projects using properties of a form attached to the workflow template. In this example, assume the following:

  o   The source form is associated with the root task as a reference attachment.

  o   The form type is **Pwf0ProjMemberForm**.

  o   The projects to receive members are listed in the value of the **pwf0Projects** form property.

  o   The non-privileged members to be added are listed in the value of the **pwf0NonPrivilegedMembers** form property.

  o   The privileged members to be added are listed in the value of the **pwf0PrivilegedMembers** form property.

Argument	Values
-source_task	**$ROOT.$REFERENCE**
-type	**Pwf0ProjMemberForm**
-projects_property	**pwf0Projects**
-members_property	**pwf0NonPrivilegedMembers**
-privileged_members_ property	**pwf0PrivilegedMembers**

- This example adds members to a project using a combination of handler arguments and form properties. In this example, assume the following:

  o   The source form is associated with the root task as a reference attachment.

  o   The form type is **Pwf0ProjMemberForm**.

  o   The projects to receive members are **Proj1** and those that are listed in the value of the **pwf0Projects** form property.

o The non-privileged members to be added are **john**, with the **Designer** role in the **Engineering** group, and those users that are listed in the value of the **pwf0NonPrivilegedMembers** form property.

o The privileged members to be added are **jane**, with the **Manager** role in the **Engineering** group, and those users that are listed in the value of the **pwf0PrivilegedMembers** form property.

Argument	Values
-source_task	**$ROOT.$REFERENCE**
-type	**Pwf0ProjMemberForm**
-projects	**Proj1**
-members	**Engineering/Designer/john**
-privileged_members	**Engineering/Manager/jane**
-projects_property	**pwf0Projects**
-members_property	**pwf0NonPrivilegedMembers**
-privileged_members_ property	**pwf0PrivilegedMembers**

PIE-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

PIE-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. The export file is saved to the server machine.

If the path is not specified, the file is placed in the **TC_TMP_DIR** directory on the server. 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.

Argument	Values
-context	**MyApplication**
-attach	**target**
-file	**tceng2myap.xml**
-revrule	**Latest Working**

PARTITION-activate-or-inactivate

DESCRIPTION

Marks a partition as active or inactive.

SYNTAX

PARTITION-activate-or-inactivate -activate={true | false}

ARGUMENTS

-activate

Marks the partition as active (**-activate=true**) or inactive (**-activate=false**).

PLACEMENT

Place in a new workflow specifically designed to activate or inactivate partitions. The ability to activate partitions must be enabled first by setting the **Ptn0EnableActivationBehavior** business object constant to **true**.

RESTRICTIONS

None.

OBJIO-send-target-objects

DESCRIPTION

Sends objects to other Multi-Site Collaboration sites.

Sends to or synchronizes objects at other Multi-Site Collaboration sites. If the object is not present at the remote site, the object is replicated; otherwise, it is synchronized.

SYNTAX

OBJIO-send-target-objects [**-class=***classname*] {**-target_site=***site-name* | **ALL** | **$SCHEDULE_SITE** | **-owning_site=***site-name* | **$SCHEDULE_SITE**} [**-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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. Use **ALL** to send the specified target objects to all sites.

Use the **$SCHEDULE_SITE** keyword to define the target site as the owning site of the schedule task or schedule task proxy link attached to the workflow process as **schedule_task**.

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.

Use the **$SCHEDULE_SITE** keyword to define the owning site as the owning site of the schedule task or schedule task proxy link attached to the workflow process as **schedule_task**.

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.

EXAMPLES

- This example shows how to send all item target objects to the **Detroit** and **Tokyo** sites without transferring ownership:

Argument	Values
-class	**Item**
-target_site	**Detroit, Tokyo**

- This example shows how to send item and dataset target objects to all sites without transferring ownership:

Argument	Values
-class	**Item, Dataset**
-target_site	**ALL**

- This example shows how to transfer site ownership of item and dataset target objects to the **Tokyo** site:

Argument	Values
-class	**Item, Dataset**
-owning_site	**Tokyo**

OBJIO-release-and-replicate

DESCRIPTION

Supports (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.

> **Note**
>
> A 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.

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

OBJIO-release-and-replicate [-revision_rule=*revision-rule-to-configure-assembly***]**
[-variant_rule=*variant-rule-to-configure-assembly***]**
[-check_precise] [-check_no_stubs] [-check_all_released]

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.

OBJIO-archive-target-objects

DESCRIPTION

Archives objects from the master site to the archive site.

The user executing **OBJIO-archive-target-objects** must be a system administrator with DBA privileges. (The user cannot be **infodba**.)

SYNTAX

OBJIO-archive-target-objects [-include_bom]

ARGUMENTS

-include_bom

Specifies to include assembly components of the BOM at all levels for processing. Caution must be exercised in using this option as all children components of BOM gets archived. This option cannot be used with 4GD target objects.

PLACEMENT

No restrictions.

RESTRICTIONS

Use in workflow processes with objects belonging to either class or subclass of **Item** or **Mdl0ApplicationModel** as targets.

OBJIO-acquire-site-ownership

DESCRIPTION

Sets the owning site of the workflow's **Replica Proposed Targets** attachments to the current site. Once the site ownership has been transferred, the objects are moved from the **Replica Proposed Targets** folder to **Target** attachment folder.

> **Note**
>
> Even if the ownership is successfully transferred, the objects might fail to be added as targets, possibly because of rule handler failures that govern target additions. Because these objects are no longer replicas due to site ownership transfer, they are not retained as **Replica Proposed Targets**. Such objects are moved to the **Reference** attachment folder of the workflow, and a corresponding message is sent to the user.

SYNTAX

OBJIO-acquire-site-ownership

ARGUMENTS

None.

PLACEMENT

Place on the **Start** or **Complete** action of any task. Do not place on the **Perform** action.

Because this handler invokes Multi-Site operations, Siemens PLM Software recommends that you place this handler on a task marked for background processing.

RESTRICTIONS

None.

MES-Update3DPDFReports

DESCRIPTION

Updates all 3DPDF reports attached to selected lines (processes and/or operations), according to the settings on the report creation. If a report update fails, the process continues until all update processes are complete.

The handler creates a dataset with a summary log, detailing for each report whether it successfully updated or not. Also, for each report that has failed to update, the handler creates a dataset with its log.

By default, the datasets are created in the **Newstuff** folder. You can define a different folder with the **MES_3DPDF_UPDATE_WORKFLOW_LOG_FOLDER** preference. If the handler does not complete in 10 minutes, a timeout error message is issued and the task fails. You can change the timeout wait time with the **MES_3DPDF_UPDATE_WORKFLOW_WAIT_TIME** preference.

SYNTAX

MES-Update3DPDFReports

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

Use only on process revision and operation revision business objects.

ME-update-mirror-mbom-AH

DESCRIPTION

Updates a manufacturing bill of materials (MBOM) based on an engineering bill of materials (EBOM). It allows different item types in the MBOM than were in the EBOM for certain nodes based on custom logic. By default, the MEMBOM_* preferences listed below set the item type to be created.

- MEMBOM_Mirror_MakeRules

- MEMBOM_Mirror_RemoveLineWithIDIC

- MEMBOM_Mirror_ReplaceMakeOnChange

- MEMBOM_Mirror_TypePrefixSuffix

You can also customize the item type to be created using the **USER_create_or_ref_item** exposed in the Business Modeler IDE through **BMF_ITEM_create_or_ref_id** on the item.

The target must be an item or item revision or a structure context object. The top line of the structure is where the update is started. If you need to start at a lower line, use the **-scopeid** or **-scopeidincontext** arguments.

.

SYNTAX

ME-update-mirror-mbom-AH
[**-revrule=***revision-rule*]
[**-mbomrevrule=***mbom-revision-rule*]
[**-depth=***depth*]
[**-clientdata=***data*]
[**-actiononrelease=** {*1* | *2* | *3* | *4*}]
[**-mscuid=***UID*]
[**-mbomroot=***root-itemid*]
[**-usemfk=** {**0** | **1**}]
[**-log=***log-file*]

ARGUMENTS

-revrule

Specifies the revision rule of the EBOM structure used to traverse. This argument is mandatory only if the target is an item or item revision. Do *not* use this argument if the target is a structure context object.

-mbomrevrule

Specifies the revision rule for the MBOM structure. This argument is mandatory only if the target is a structure context object. This argument is required if the target is an item revision.

-depth

(Optional) Specifies the depth up to which to create the MBOM nodes.

If you do not specify this value, Teamcenter creates all of the MBOM nodes.

-clientdata

(Optional) Data to be passed to any custom-registered user exit functions defined on the item.

-actiononrelease

(Optional) A value indicating the action to be taken if an MBOM node already exists (has a computed ID) and is released. Possible values are:

1 – Skip (the default).

2 – Revise and modify,

3 – Update properties on the released item.

4 – Update properties on the MBOM and its children.

-mscuid

The UID of the structure context object for the MBOM structure if the **mobmrevrule** is not suitable (for example, it is a private revision rule). Either this argument or the **mbomroot** argument is mandatory.

-mbomroot

(Optional) The ID of the root of the MBOM structure. Either this argument or the **mscuid** argument is mandatory.

-scopeid

(Optional) Specifies the item ID in the EBOM from which to begin the traversal. This argument cannot be used with **scopeidincontext**.

If you do not specify this value, Teamcenter begins the traversal at the top line in the EBOM.

-scopeidincontext

(Optional) Specifies the IDIC of the line in the EBOM from which to begin the traversal. This argument cannot be used with **scopeid**.

If you do not specify this value, Teamcenter begins the traversal at the top line in the EBOM.

-log

(Optional) Specifies the absolute path and name of the log file to capture details about the nodes created.

-usemfk

(Optional) Specifies whether to include the **MEMBOM_Mirror_TypePrefixSuffix** preference value with the EBOM item identifier as a multifield key that updates the existing MBOM item identifier. Values are:

0   Do not include the preference value in the multifield key. This value is the default.

1   If the preference defines an item type without a prefix or a suffix, include the value in the multifield key. This value is ignored if the preference also defines a prefix or a suffix.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EXAMPLES

In the following EBOM and MBOM, **M_000229/A;1–M_rear_drive_asm** is released and then the make/buy property on its child, **000230/A;2–rear_axle**, is changed from Buy to Make.
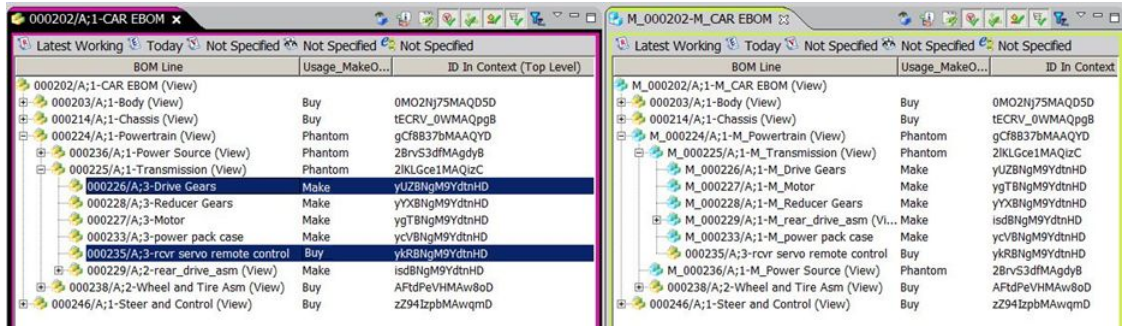
You revise the MBOM part so you have write access and run the update workflow using the following arguments on **ME-update-mirror-mbom-A** action handler for the target MBOM:

Argument	Values
-revrule=	**"Latest Working"**
-mbomrevrule=	**"Latest Working"**
-actiononrelease=	**2**

The results are the following:

ME-stamp-ids-AH

DESCRIPTION

Traverses a structure according to a closure rule and automatically assigns a value to a specific property based on a recipe determined by the value of the **MEIdGenerationPropertySetting** preference. The workflow targets must be items, item revision, or structure context objects.

The target item or item revision is used as the top line of the BOM window. Normally, the top line of the structure is where the transverse is started. If you need to start at a lower line, use the **-scopeid** or **-scopeidincontext** arguments.

SYNTAX

ME-stamp-ids-AH
[**-revrule**=*revision-rule*]
[**-scopeid**=*scope-ID*|**-scopeidincontext**=*scope-in-context-ID*]
[**-closurerule**=*closure-rule-name*]
[**-preference**=*preference-name*]
[**-forceupdate=1**]

ARGUMENTS

-revrule

Specifies the revision rule. This argument is mandatory only if the target is an item or item revision to set up the BOM window. Do not use this argument if the target is a structure context object.

-scopeid

(Optional) Specifies the item ID in the manufacturing BOM from which to begin the traversal. This argument cannot be used with **scopeidincontext**.

If you do not specify this value, Teamcenter begins the traversal at the top line in the manufacturing BOM.

-scopeidincontext

(Optional) Specifies the IDIC of the line in the manufacturing BOM from which to begin the traversal. This argument cannot be used with **scopeid**.

If you do not specify this value, Teamcenter begins the traversal at the top line in the manufacturing BOM.

-closurerule

(Optional) Specifies the closure rule that determines which lines in the structure Teamcenter stamps when it traverses the manufacturing BOM structure below the scope line.

If you do not specify a closure rule, every line in the structure below the given scope line is stamped.

-preference

(Optional) Specifies the preference name containing the rules for setting the BOM line property. The default preference is **MEIdGenerationPropertySetting**.

-forceupdate=1

(Optional) Specifies that an existing ID in a **Context** string should be ignored and that a new value is generated. By default, the old value is not overridden.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EXAMPLES

- This example creates in-context IDs that are based on the **Usage Address** property based on the constituent properties of item ID and item type. To do this:

  1. Define the recipe for the IDIC value by setting the **MEIdGenerationPropertySetting** to:

     type:Item,key: bl_usage_address,prop:bl_item_item_id, prop:bl_item_object_type

  2. Do one of the following:

     o Create the usage address property on each line under the top line.

Argument	Values
-revrule	**Latest Working**

     o Create the usage address on selected lines specified in a closure rule under a scope line determined by the specified IDIC (top level) value. In other words, the handler begins with a line that you specify by IDIC, traverses the structure from the IDIC line downward using the given closure rule, and stamps the resulting lines with the usage address string.

Argument	Values
-revrule	**Latest Working**
-scopeidincontext	**kJBtMh0hAABaaA**
-closurerule	**AccountabilityAll**

ME-mbom-resolve-AH

DESCRIPTION

Searches the specified engineering bill of materials (EBOM) for parts that resolve the search recipes defined in the target (root) manufacturing bill of materials (MBOM) and assigns them to the MBOM.

You can choose the scope of the resolution and whether to recursively resolve all nodes underneath the selected scope (**-recurse**) and remove previously assigned parts. Because you most often define the root of the EBOM as the target, be sure to set the **-recurse** argument to **1** to resolve the entire structure.

.

SYNTAX

ME-mbom-resolve-AH
[**-itemid=***UID* | **-scuid=***in-context-ID* | **-key=***multi-field-key-of-structure-root*]
[**-revrule=***revision-rule*]
[**-mbomrevrule=***revision-rule*]
[**-log=***log-file*]
[**-scopeid=***scope-ID* | **-scopeidincontext=***scope-in-context-ID* |
    -scopekey=*multi-field-key-of-structure-root*]
[**-mscopeid=***UID* | **-mscopeidincontext=***scope-in-context ID* |
    -mscopekey=*multi-field-key-of-structure-root*]
[**-recurse=1 | 0**]
[**-removepreviousresolvednodes=1 | 0**]

ARGUMENTS

-itemid

(Optional) Specifies the root of the EBOM structure to be searched.

One of the **-itemid**, **scuid**, or **key** arguments is mandatory. Therefore, do *not* use if you define a structure context or a key.

-scuid

(Optional) Specifies the structure context capturing the root of the EBOM structure and configuration to be searched.

One of the **-itemid**, **-scuid**, or **-key** arguments is mandatory. Therefore, do *not* use if you define an item or item revision or a key.

-key

(Optional) Specifies the key of the top line of the root EBOM structure to be searched when multiple attributes are used to form the unique item ID. Use the following format:

    [keyAttr1=keyVal1] [,keyAttr2=keyVal2]…[,keyAttrN=keyValN]

One of the **-itemid**, **-scuid**, or **-key** arguments is mandatory. Therefore, do *not* use if you define an item or item revision or structure context.

-revrule

(Optional) Specifies the revision rule of the EBOM structure to be searched. This argument is mandatory only if the EBOM is an item or item revision or key. Do *not* use if the target is a structure context object.

-mbomrevrule

(Optional) Specifies the revision rule for the MBOM structure where the recipes are defined. This argument is mandatory only if the target is *not* a structure context object.

-log

(Optional) Specifies the absolute path and name of the log file to capture details.

-scopeid

(Optional) Specifies the item ID in the EBOM from which to begin the search. This argument cannot be used with **scopeidincontext** or **scopekey**.

If you do not specify this value, Teamcenter begins searching at the top line of the EBOM.

Select one of the **-scopeid**, **-scopeidincontext**, or **-scopekey** arguments. Do *not* use if you define a structure context or a key.

-scopeidincontext

(Optional) Specifies the ID in top level context in the EBOM from which to begin the search. This argument cannot be used with **scopeid**.

If you do not specify this value, Teamcenter begins searching at the top line of the EBOM.

Select one of the **-scopeid**, **-scopeidincontext**, or **-scopekey** arguments. Do *not* use if you define an item or item revision or a key.

-scopekey

(Optional) Specifies the IDIC of the line in the EBOM from which to begin the search. This argument cannot be used with **scopeid**.

If you do not specify this value, Teamcenter begins searching at the top line of the EBOM.

Select one of the **-scopeid**, **-scopeidincontext**, or **-scopekey** arguments. Do *not* use if you define an item or item revision or structure context.

-mscopeid

(Optional) Specifies the item ID in the MBOM to resolve, for example, if you want to resolve for a particular phantom node. This argument cannot be used with **mscopeidincontext**.

If you do not specify this value, Teamcenter resolves at the top line of the MBOM.

Select one of the **-mscopeid**, **-mscopeidincontext**, or **-mscopekey** arguments. Do *not* use if you define a structure context or a key.

-mscopeidincontext

(Optional) Specifies the ID in top level context in the MBOM to resolve, for example, if you want to resolve for a particular phantom node.

If you do not specify this value, Teamcenter resolves the recipes starting at the top line of the MBOM.

Select one of the **-mscopeid**, **-mscopeidincontext**, or **-mscopekey** arguments. Do *not* use if you define an item or item revision or a key.

-mscopekey

(Optional) Specifies the IDIC of the line in the MBOM to resolve, for example, if you want to resolve for a particular phantom node. This argument cannot be used with **mscopeid**.

If you do not specify this value, Teamcenter resolves the recipes starting at the top line of the MBOM.

Select one of the **-mscopeid**, **-mscopeidincontext**, or **-mscopekey** arguments. Do *not* use if you define an item or item revision or structure context.

-recurse

(Optional) Specifies whether to resolve all nodes under the specified scope node. Valid values are **1** and **0**. The default value is **0** meaning Teamcenter only resolves the recipes at the specified scope node.

-removepreviousresolvednodes

(Optional) Specifies whether to remove the previously assigned parts. Valid values are **1** and **0**. The default value is **0** meaning Teamcenter does not remove parts that have already been resolved in the MBOM.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EXAMPLES

The following arguments search the EBOM (**000202/A;1–CAR_EBOM**) for parts that resolve the recipes defined at node **000224/A** of the target MBOM (**M_000202–M_CAR_MBOM**) and assigns them to the MBOM:

Argument	Values
-itemid=	**000202**
-revrule=	**"Latest Working"**
-mbomrevrule=	**"Latest Working"**
-mscopeid=	**000224**
-recurse	**1**
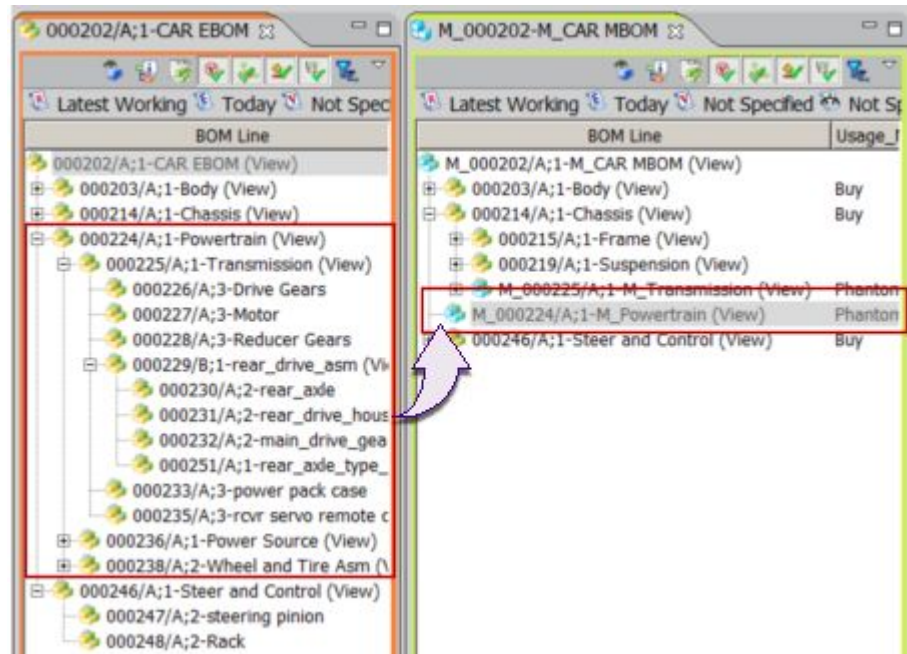
ME-create-revision-change-XML-AH

DESCRIPTION

Creates a revision change delta XML file. The manufacturing change notice (MCN) revision contains the item revisions to find revision changes. The configuration context object supplies the current configuration, and the MCN can optionally have a **was** configuration set on it. The generated XML file is attached to the request object.

SYNTAX

ME-create-revision-change-XML-AH
[**-filename=**_file-name_]

ARGUMENTS

(Optional) **-filename=**_file-name_

When you specify the **-filename=** argument, the system uses it as a base name; however, the actual filename is **RevisionChangeXML**_basename-randomstring_**.xml**

REFERENCES

- (Required) MCN revision object.

- (Required) Configuration context (execution plan type) object.

TARGETS

(Required) Request object.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

ME-create-mirror-mbom-AH

Creates a manufacturing bill of materials (MBOM) based on an engineering bill of materials (EBOM). It allows different item types in the MBOM than were in the EBOM for certain nodes based on custom logic. By default, the MEBOM_* preferences listed set the item type to be created.

- MEMBOM_Mirror_MakeRules

- MEMBOM_Mirror_RemoveLineWithIDIC

- MEMBOM_Mirror_ReplaceMakeOnChange

- MEMBOM_Mirror_TypePrefixSuffix

You can also customize the item type to be created using the **USER_create_or_ref_item** exposed in the Business Modeler IDE through **BMF_ITEM_create_or_ref_id** on the item.

Attachments that are associated with item revisions in the EBOM structure are carried forward. A user exit operation (**USER_sync_item**) must also be available on the item to align any additional attachment information or non-occurrence properties. The user exit is exposed in the Business Modeler IDE through **BMF_ITEM_sync** on the item.

The target must be an item or item revision or a structure context object. The top line of the structure is where the create starts.

.

SYNTAX

ME-create-mirror-AH
[**-revrule=***revision-rule*]
[**-mbomrevrule=***mbom-revision-rule*]
[**-depth=***depth*]
[**-clientdata=***data*]
[**-actiononrelease=** {*1* | *2* | *3* | *4*}]
[**-mscuid=***UID*]
[**-usemfk=** {**0** | **1**}]
[**-log=***log-file*]

ARGUMENTS

-revrule

Specifies the revision rule of the EBOM structure used to traverse. This argument is mandatory only if the target is an item or item revision. Do *not* use this argument if the target is a structure context object.

-mbomrevrule

Specifies the revision rule for the MBOM structure. This argument is mandatory only if the target is a an item or item revision. Do *not* use this argument if the target is a structure context object.

-depth

(Optional) Specifies the depth up to which to create the MBOM nodes.

If you do not specify this value, Teamcenter creates all of the MBOM nodes.

-clientdata

(Optional) Data to be passed to any custom-registered user exit functions defined on the item.

-actiononrelease

(Optional) Specifies a value indicating the action to be taken if an MBOM node already exists (has a computed ID linked to the EBOM) and is released. Possible values are:

> **1** – Skip (the default).
>
> **2** – Revise and modify,
>
> **3** – Update properties on the released item.
>
> **4** – Update properties on the MBOM and its children.

-mscuid

Specifies the UID of the structure context object for the MBOM structure if the **mobmrevrule** is not suitable (for example, it is a private revision rule).

-log

(Optional) Specifies the absolute path and name of the log file to capture details about the nodes created.

-usemfk

(Optional) Specifies whether to include the **MEMBOM_Mirror_TypePrefixSuffix** preference value with the EBOM item identifier as a multifield key that becomes the new MBOM item identifier. Values are:

0	Do not include the preference value in the multifield key. This value is the default.
1	If the preference defines an item type without a prefix or a suffix, include the value in the multifield key. This value is ignored if the preference also defines a prefix or a suffix.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EXAMPLES

The following examples of specifying arguments for the **ME-create-mirror-mbom-AH** action handler demonstrate its use and the differences in output caused by changing the arguments. The initial values of the preferences are as follows.

- **MEMBOM_Mirror_MakeRules=KEY:Usage_MakeOrPurchase,VALUE:Make |VALUE:Phantom**

- **MEMBOM_Mirror_RemoveLineWithIDIC=false**

- **MEMBOM_Mirror_TypePrefixSuffix=Company,M_**

• **MEMBOM_Mirror_ReplaceMakeOnChange=false**

• **Create an MBOM**

For the following EBOM, which is the target of the workflow:

The following arguments on **ME-create-mirror-mbom-AH**:

Argument	Values
-revrule=	"Latest Working"
-mbomrevrule=	"Latest Working"

Produce the following MBOM:

• **Create the MBOM to a specific level**

For the following EBOM, which is the target of the workflow:
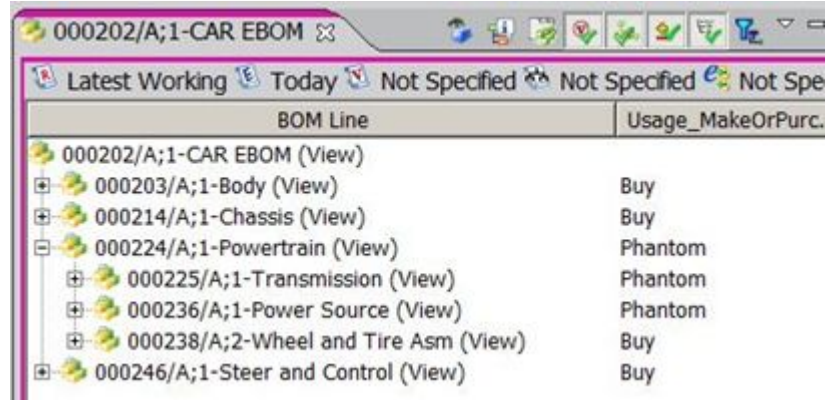
The following arguments:

Argument	Values
-revrule=	"Latest Working"
-mbomrevrule=	"Latest Working"
-depth=	2

Produce a new MBOM, which contains only two levels of structure. The remaining levels in the EBOM are ignored.
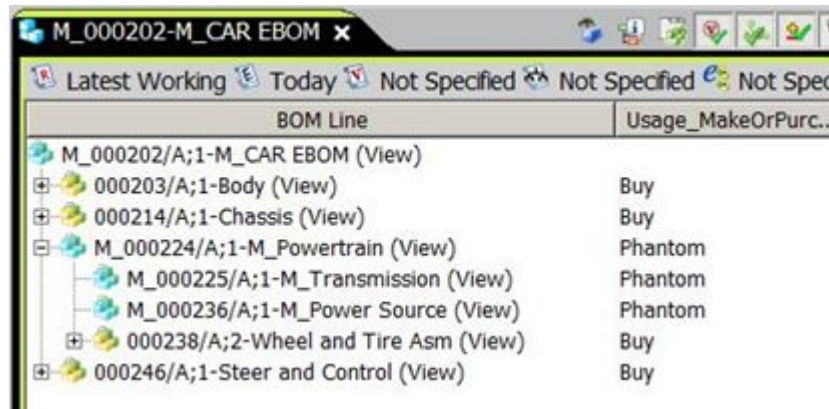
MDL-snapshot-baseline-revisions

DESCRIPTION

Sets the snapshot date and closure state for all baseline revisions that are targets of the root task.

Deleting and re-adding the baseline snapshot date is an alternative to revising a baseline. It avoids creating additional objects, but does not allow a record to be kept of the failed baseline attempt. The choice between a re-open versus a revise step is a business decision, and it is expected to be formally designed as a workflow.

Siemens PLM Software recommends that you use a workflow action handler to close a baseline before performing signoffs. Once signoffs are complete, we recommend using another workflow action handler to assign a status to the baseline.

SYNTAX

MDL-snapshot-baseline-revisions -snapshot = add | replace | delete -closure=_name_ **]**

ARGUMENTS

-snapshot

Sets the baseline revision snapshot date. The value can be one of the following:

- **add**

  Ensure the baseline revision has a snapshot date.

  If the baseline revision does not have a snapshot date, it is set to the current date.

  If the baseline revision already has a snapshot date, the snapshot date is unchanged.

- **replace**

  Sets the baseline revision snapshot date to the current date.

- **delete**

  Sets the baseline revision snapshot date to **null**.

-closure

Sets the baseline revision closure property to the specified value.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

MDL-promote-objects-to-history

DESCRIPTION

Promotes all targets and any related objects to history. For non-revisable targets, this handler checks the maturity status for object stability.

If the target object is revisable, the logical object is copied to POM history and its references are checked for stability.

If the target object is not revisable, it is checked for stability

SYNTAX

MDL-promote-objects-to-history

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

MDL-attach-subset-definition-changes

DESCRIPTION

Compares the **mdl0HistorySyncStatus** property for the content of all target subset definitions. Where content is out of synchronization, the handler adds the content to the workflow as a target.

An **Mdl0ModelElement** business object is in sync whenever the **mdl0HistorySyncStatus** property value is empty (**" "**).

Examine both the latest-history and latest configurations for both content and partitions. This is required to get the correct promote-to-history of obsoleted or configured-out content.

SYNTAX

MDL-attach-subset-definition-changes [-partition=[scheme1, [scheme2, ...]** | [* | all | any]**

ARGUMENTS

-partition

(Optional) Attaches the required partitions from the subset definition content up to the root partitions. You can specify multiple partition schemes by name, all partitions, or any partition.

If the **-partition** argument is used, partitions in the specified partition schemes are also attached if the partition is:

• Configured by the subset definition.

• Itself is out of sync.

• Lies on the path from the subset definition content to the root partitions.

PLACEMENT

Place before the **MDL-promote-objects-to-history** handler to synchronize the subset definition content with the history.

RESTRICTIONS

None.

MDL-attach-changes-to-baselines

DESCRIPTION

For all change item revisions that are targets of the root task, this handler finds any baseline revisions in the **Reference Items** folder and attaches the change item revision as a reference to the baseline.

If the attachment fails for any reason, an error is returned.

SYNTAX

MDL-attach-changes-to-baselines

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

LDF-set-task-result-to-property

DESCRIPTION

LDF-set-task-result-to-property reads the specified property from the remote object. **LDF-set-task-result-to-property** uses that property value to set the result string attribute of the task where this handler is located, or on the task specified by the **-target_task** argument. A common use for this handler is to control **Condition** task branching instead of using a scheme that requires a custom handler. Using this handler to set the result attribute of a **Condition** task branches the workflow process based on a remote property of the target source object.

SYNTAX

> **-property**
> **-source**
> **[-attachment_relation]**
> **[-include_type]**
> **[-target_task]**
> **[-remote_user_name]**

ARGUMENTS

-property   Specifies the property to be read from the identified remote object attached to the target with specified relation.

The values specified for this argument require a fully qualified property name with a prefix URL prepended to every property in a workflow argument value. The OSLC namespace prefix URL must be contained in angle brackets in the **<oslc-namespace-prefix-url > -property-name** format as shown in the **Examples** section below.

This is a mandatory argument.

-source   Determines which source object identifies the remote object property. Source object values are either **target** or **reference**. The remote object property is identified in the **-property** argument.

- **target**

  Declares that the remote object property is read from a target object. The **-include_type** argument specifies the target object type to use.

- **reference**

  Declares that the remote object property is from a reference object. The **-include_type** argument specifies the reference object type to use.

-attachment_relation Specifies the relation name to expand to get a linked object from a workflow attachment. Linked objects, attached to targets and references in a workflow with the relation specified by **-attachment_relation**, are searched. Linked objects not matching the specified relation criteria are ignored.

This is an optional argument.

-include_type Identifies the source type to read the specified property of the remote object. The property name is defined in the **-property** argument. If more than one target object of a given type exists, the first target on the list is used. If a valid **-include_type** argument is absent, the property is read from the first target on the list.

-target_task Identifies where the result string attribute is set. If not specified, then the task result attribute is set for the task containing this handler.

This is an optional argument.

- **$ROOT_TASK**—Sets the result string attribute on the root task of the process.

- **$DEPENDENT_TASK**—Sets the result string attribute on the parent process task which is dependent on this subprocess. The parent process task should be a **Condition** task.

-remote_user_name Used by the handler to connect to a remote system, like Polarion, for sending HTTP requests.

The **Restrictions** section below describes separate actions required to generate an encrypted password file.

> **Note**
>
> This argument is optional with SSO.

PLACEMENT

Typically placed on the **Start** action of the specified **Condition** task.

> **Note**
>
> You can apply the **LDF-set-task-result-to-property** handler to *any* task, but it sets the result on either a root or **Condition** task.
>
> The **Condition** task can contain the handler or be a parent of another dependent task that contains the handler.

RESTRICTIONS

- Do not place this handler on the **Perform** action.

- Do not use this handler in conjunction with other handlers that set the **result** attribute, such as **EPM-set-condition**, **EPM-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.

- This handler allows you to set the **result** attribute on the root task or any other **Condition** task.

- Use if you are using the Linked Data Framework for application integrations and you want Teamcenter workflows to create an object in a remote system. You must generate an encrypted password file in a Teamcenter command shell. To create an encrypted password:

  1. Run this command: **mkdir %TC_DATA%\polarionconnector**.

  2. Run this command: **%TC_ROOT%\bin\install -encryptpwf -f=%TC_DATA%\polarionconnector\<user name>**.

     Where **<user name>** is the user name of a remote system such as Polarion ALM. Configure this user name as a value of the **–remote_user_name** handler.

EXAMPLES

This **LDF_set_task_result_to_property** handler configuration branches a **Condition** task based on the remote object property **Priority**, which is attached to a target change request revision with the relation **Lcm0RelatedChangeRequest**.

Argument	Values
-property	**<http://polarion.plm.automation.siemens.com/oslc#>.priority**
-source	**target**
-include_type	**ChangeRequestRevision**
-attachment_relation	**Lcm0RelatedChangeRequest**
-remote_user_name	**admin**

LDF-create-object

DESCRIPTION

Creates an object in the remote system and relates it to the workflow attachment.

SYNTAX

**LDF-create-object
service_provider
-object_type
[-property::**<*oslc-namespace-prefix-url*>.*property-name***]
[-from_attach]
-attachment_relation
[-remote_user_name]**

ARGUMENTS

-service_provider

Service provider represents the services published by the external application.

> **Example**
>
> Polarion is registered in Teamcenter as a site and service provider under which my objects will be created.

This is a mandatory argument.

The values specified for this argument can be dynamic. Users can configure the handler argument to read the property values from workflow attachments and substitute them as the argument values. For example, **-service_provider=PROP::owning_project** where **owning_project** is the property of the Teamcenter workflow attachment. If corresponding **service_provider** is not found, this handler returns an error.

-object_type

This argument specifies the type of object created in the remote system.

This is a mandatory argument.

-property::<*oslc-namespace-prefix-url>.property-name*

Specifies the property name for the remote object to be created.

Requires a fully qualified property name with a prefix URL prepended to every property in a workflow argument, which is prepended by **-property::**. The OSLC namespace prefix URL must be contained in angle brackets, < and >, in the **<oslc-namespace-prefix-url >**.*property-name* format as shown in the examples section.

The values specified for this argument can be dynamic. User can configure the handler argument to read the property values from workflow attachments and substitute it as the argument value.

For example, **-property::<http://purl.org/dc/terms/>.title =PROP::object_name** where **object_name** is the property of the Teamcenter workflow attachment. User can also configure prefix or suffix.

The dynamic property values can also have prefix or suffix. For example, **-property::<http://purl.org/dc/terms/>.title =ABC PROP::object_name XYZ** *ABC* is the prefix, *PROP::object_name* is the dynamic value from Teamcenter object, and *XYZ* is the suffix.

-from_attach

target | reference

(Optional) Specifies which type of attachment (target or reference) to get the property value from when a property is specified in the **-property::**<*oslc-namespace-prefix-url>.property-name* argument. For example, **-property::**<*http://purl.org/dc/terms/>.title=PROP::object_name* where *object_name* is the property of the Teamcenter workflow attachment.

You can use this argument only when you get the property value from a property of the attachment object.

-attachment_relation

Specifies the relation name linking the remote object with the target. This relation name should match a relation name configured in Linked Data Framework. Refer to .

This is a mandatory argument.

-remote_user_name

> Used by the handler to connect to a remote system like Polarion for sending HTTP requests.
>
> The Restrictions section below describes separate actions required to generate an encrypted password file.

> **Note**
>
> This argument is optional with SSO.

PLACEMENT

Place on the **Start** or **Complete** action.

> **Note**
>
> Do not place on a **Perform** action requiring specific user interaction. Placement on the **Perform** action may cause the handler to be triggered multiple times.

RESTRICTIONS

Use if you are using the Linked Data Framework for application integrations, and you want Teamcenter workflows to create an object in a remote system.

You must generate an encrypted password file by following these steps in a Teamcenter command shell:

1. Run this command:

   mkdir %TC_DATA%\polarionconnector

2. Run this command:

   %TC_ROOT%\bin\install -encryptpwf -f=%TC_DATA%\polarionconnector *<user name>*

   Where *<user name>* is user name of remote system such as Polarion ALM. This user name should be configured as a value of the **–remote_user_name** handler.

EXAMPLES

- This example shows the **LDF-create-object** handler configuration to create an object in the remote system of type **changerequest** , and attaching the remote link of this object with target by **Lcm0RelatedChangeRequest** relation. Uses service provider and title values from target object properties **object_desc**, **object_name**, respectively.

Argument	Values
-service_provider	PROP::object_desc
-object_type	changerequest
-property:: <http://purl.org/dc/terms/>.title	PROP::object_name

Argument	Values
-from_attach	target
-attachment_relation	Lcm0RelatedChangeRequest
-remote_user_name	admin

- This example shows the **LDF-create-object** handler configuration to create an object in the remote system of type **issue**, and attaching the remote link of this object with target by **Lcm0AffectedByDefect** relation. Uses title and description values from target object properties **object_name**, **object_desc** respectively.

Argument	Values
-service_provider	Drive Pilot
-object_type	issue
-property:: <http://purl.org/dc/terms/>.title	PROP::object_name
-property:: <http://purl.org/dc/terms/>.description	PROP::object_desc
-from_attach	target
-attachment_relation	Lcm0AffectedByDefect
-remote_user_name	admin

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.

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.

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.

Argument	Values
-Approve	

• 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.

Argument	Values
-Reject	

• 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.

GMIMAN-invoke-subscription-event-on-item

DESCRIPTION

Notifies the subscribed user about an event by checking the release status of the item revision with the specified argument.

SYNTAX

GMIMAN-invoke-subscription-event-on-item -event=_event-type-release-status_

ARGUMENTS

-event

Valid event-type release status.

PLACEMENT

Add this handler after the **EPM-set-status** handler in the **Complete** action of the release workflow.

RESTRICTIONS

This handler can only be used when the GM Overlay is installed. The valid event-type release statuses are limited to the event types that are installed for the Subscription Administration.

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 to the export directory.

SYNTAX

ERP-transform-AI-contents-AH

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action of any task. Apply after the **AI-export-AH** handler.

RESTRICTIONS

None.

ERP-set-pathnames-in-logds-AH

DESCRIPTION

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.

SYNTAX

ERP-set-pathnames-in-logds-AH

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action of any task. Apply after the **EPM-set-pathnames-in-logds-AH** handler.

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:

    o   Removes transfer folders from the process and delete them.

    o   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-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-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-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 list of the relation types used to relate ERP forms to item revisions.

Separate multiple types with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

> **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.

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_rellog_file_path** 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 run before any other handler.

RESTRICTIONS

None.

EPM-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 object process 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

EPM-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 |
Processes_Referencing_This_Process | Reference_Object_Processes]

[-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 run 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.

If this argument is not specified, it defaults to the name of this handler: **EPM-trigger-action-on-related-process-task**.

This argument is optional.

-process_type

The workflow processes to find. It can have one of the following values:

• **Processes_Referencing_Target_Objects**

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.

This is the default value for this argument

• **Reference_Object_Processes**

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.

• **Processes_Referencing_This_Process**

Finds workflow processes that reference the current workflow process.

This argument is optional.

-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 **EPM-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

Do not use this handler in a subprocess.

EXAMPLES

The following example has two workflow process templates: **Initiate Item Revision** and **Initiate Dataset**. The **EPM-trigger-action-on-related-process-task** handler in the **Initiate Item Revision** process triggers the **Complete** action on the **ApproveDesignWork** task in the **Initiate Dataset** process.

This example uses the following item revision with a **UGMASTER** dataset:

Process Template
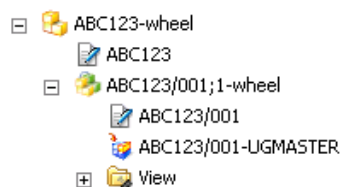
Process Template	Tasks	Steps to follow
Initiate Item Revision	**Start**→ **ApproveItemRevision** (**Review** task)→ **Finish**	In the root task in the **Start** action, add the **EPM-trigger-action-on-related-process-task** handler with the following arguments: • **-task=ApproveDesignWork** • **-action=COMPLETE** • **-comment=approved**

Process Template
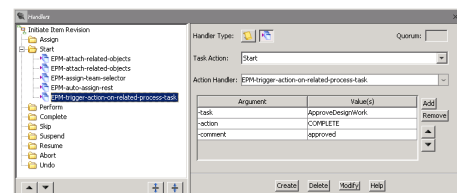
Process Template	Tasks	Steps to follow
Initiate Dataset 	**Start→** **CreateDesignWork** (**Review** task)→ **ApproveDesignWork** (**Review** task) → **Finish**	Create an **Initiate Dataset** workflow process for the **ABC123/001-UGMASTER** dataset and paste the **ABC123/001** item revision as the reference attachment. Sign off the **CreateDesignWork** task, which starts the **ApproveDesignWork** task. Then, create an **Initiate Item Revision** workflow process for the **ABC123/001** item revision.

Since **-process_type=Processes_Referencing_Target_Objects** is the default setting, and the **ABC123/001** item revision is a reference attachment of the **Initiate Dataset** process, the **Complete** action of the **ApproveDesignWork** task is triggered.

Note that the **Complete** action is successful only if all conditions for the completing a **Review** task are already met.

EPM-trigger-action

DESCRIPTION

Triggers the specified action on the task to which this handler is attached.

SYNTAX

EPM-trigger-action -action=*action* **-comment=***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 EPM-trigger-action handler** when the **Complete** action is logged in the workflow audit log file.

Argument	Values
-action	**EPM_complete_action**
-comment	**Triggering the Complete action from the EPM-trigger-action handler**

EPM-system

DESCRIPTION

Runs the first operating system argument passed to it.

The **EPM-system** handler cannot handle run-time command line arguments. For information about addressing such issues, see the **EPM-execute-follow-up** action handler. The **EPM-system** handler does not accept return values.

SYNTAX

EPM-system -command= *argument*

ARGUMENTS

-command

Operating system command to be run. 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**:

Argument	Values
-command	**mailx -s "Notification: Task has been approved" smith /tmp/approval_note.txt**

EPM-suspend-on-reject

DESCRIPTION

Suspends the task when the approval quorum cannot be met.

SYNTAX

EPM-suspend-on-reject

ARGUMENTS

None.

PLACEMENT

Place on the **Perform** action of the **perform-signoffs** task.

RESTRICTIONS

Place only on the **perform-signoffs** task.

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 or on the task specified by the **-target_task** argument. A common use for this handler is to control **Condition** task branching instead of using 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 [-source_task=*task-name***] [-include_type=***target-object-type***] [-target_task= $ROOT_TASK | $DEPENDENT_TASK] -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.

-source_task

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 **-source_task** argument is absent, the property is read from the task where the handler is located.

-include_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 **-include_type** argument is absent, the property is read from the first target on the list.

-target_task

Identifies where the result string attribute is set.

This is an optional argument. If **-target_task** is not specified, then the task **result** attribute will be set for the task containing the **EPM-set-task-result-to-property** handler.

- **$ROOT_TASK**

  Sets the result string attribute on the root task of the process.

- **$DEPENDENT_TASK**

    Sets the result string attribute on the parent process task which is dependent on this subprocess. The parent process task should be a **Condition** task.

    -property

    Specifies the property to be read from the identified source object (**task** or **target**).

Typically placed on the **Start** action of the specified **Condition** task.

However, this handler can be placed on any task but can set the result only on either the root task or a **Condition** task. The **Condition** task can be the task where the handler is placed or a parent task that is dependent on the task where the handler is placed.

RESTRICTIONS

- Do not place this handler on the **Perform** action.

- Do not use this handler in conjunction with other handlers that also set the **result** attribute, such as **EPM-set-condition**, **EPM-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.

- This handler allows you to set the **result** attribute on the root task or any other **Condition** task.

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.

Argument	Values
-source	**target**
-include_type	**ItemRevision**
-property	**item_revision_id**

    You then draw paths from the **Condition** task and assign custom flow path values by right-clicking the path and choosing **Custom**.

- 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.

Argument	Values
-source	**task**
-source_task	**Task01**
-property	**resp_party**

- 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 **-source_task** argument and therefore defaults to reading the responsible party attribute from the **Task02 Condition** task.

Argument	Values
-source	**task**
-property	**resp_party**

EPM-set-status

DESCRIPTION

Applies the appropriate release status to the workflow process targets. This handler gets the release status type that the **EPM-create-status** handler attaches to the root task.

> **Note**
>
> The **EPM-set-status** workflow handler is designed to work on release status effectivity, which is commonly used to express effectivity for item revisions used in a BOMView revision in Structure Manager.
>
> Release status effectivity is not applicable for Product Configurator or 4th Generation Design objects. However, you can use the **CONFMGMT-cut-back-effectivity** workflow handler to propagate the release status effectivity of an engineering change object to configurator and 4GD objects that are attached to the change object as solution items. This translates the release status effectivity to the effectivity model used in Product Configurator and 4th Generation Design.

> **Note**
>
> The **EPM_skip_dataset_purge** preference determines if dataset versions are purged when the **EPM-set-status** workflow handler adds a status.

> **Note**
>
> Configure the **WRKFLW_change_target_lmu** preference to indicate if the **last_mod_user** attribute of a workflow target is changed when the status is applied. Set the preference value to **TRUE** to indicate the attribute value is changed to the user who completes the task, or set the preference value to **FALSE** indicating the attribute value is not changed.

SYNTAX

EPM-set-status -action=append | rename | replace | delete
[-status=*name***]**
[-new_status=*new_name***]**
[-retain_release_date]
[-set_effectivity]
   [-status_not_shared]

ARGUMENTS

-action

• **append**

Attaches the root-task release status to the targets. Any previous statuses for the same targets are not affected.

- **rename**

  Renames the release status from *name* to *new_name*.

  If the *name* release status is not found, the handler renames the last status attached to the targets.

- **replace**

  Removes all release statuses attached to the targets, and attaches the root task release status to the targets.

  > **Note**
  >
  > If more than one status object exists on the root task, apply the **–status** argument variable =*status_name*. If the **–status** argument is not specified then replacement status is not guaranteed.

- **delete**

  Removes the release status specified by the **-status** argument from the targets.

  o If the **-status** argument is not used, all release statuses are removed from the targets.

  o This handler does not remove root-task release statuses that were created in the same workflow as the root task.

  This value can also be used to remove release statuses that were applied in other workflows.

-status

Specifies the name of the release status. When used with the **-action** argument, offers additional options to define the status.

> **Note**
>
> Enter the name as defined in the Business Modeler IDE, not the display name.

-action argument value	-status argument result
append	If the specified release status is not attached to the root task, the handler: • Creates a new status with the specified name. • Attaches the new status to the root task.
rename	The handler renames the release status to the value specified in **-new_status**.

-action argument value	-status argument result
replace	If the specified release status is not attached to the root task, the handler:
	• Creates a new status with the specified name.
	• Attaches the new status to the root task.
delete	The handler removes the release status from the targets, but does not remove the status from the root task.

-new_status

Specifies the new name for the release status. Use this argument only if you use the **-action** argument's **rename** value.

Enter the name as defined in the Business Modeler IDE, not the display name.

> **Caution**
>
> If the release status type is not defined, effectivity and configuration may be unavailable for the release status.

-retain_release_date

Retains the original release date on the target if it had previously been released. Not valid for **replace**.

-set_effectivity

If used, the handler creates the open-ended date effectivity with release date as start date.

-status_not_shared

Places on each target an individual copy of the root-task release status. By default, all targets share a reference to the release status.

PLACEMENT

Place on any action. Typically attached to the **Complete** action.

RESTRICTIONS

• By default, the **-action** argument and its **append** value are assumed if no argument is specified, or if an argument other than those specified is supplied to the handler.

• If the root task bears two or more statuses, and if the **-action** argument value is **replace**, the latest status on the root task replaces the status on the targets.

EXAMPLES

• This example adds the status object of the root task to the target object:

Argument	Values
-action	**append**

- This example adds the status object of the root task to the target object and retains the original released date of the target object:

Argument	Values
-action	**append**
-retain_release_date	

- This example replaces all existing status objects with the status object of the root task:

Argument	Values
-action	**replace**

- 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:

Argument	Values
-action	**replace**
-set_effectivity	

- This example renames all the status objects named **pre-released** to the name of the new status object, **released**:

Argument	Values
-action	**rename**
-status	**pre-released**
-new_status	**released**

- This example deletes all status objects from the target object but does not delete it from the root task:

Argument	Values
-action	**delete**

- This example deletes a status called **released** from the target object, but does not delete it from the root task:

Argument	Values
-action	**delete**
-status	**released**

- This example attaches a release status named **released** to the root task:

Argument	Values
-action	**append**

Argument	Values
-status	**released**

- This example places on each target an individual copy of the root-task release status.

Argument	Values
-action	**append**
-status_not_shared	

- This example creates a new release status named **released**, attaches that status to the root task. and places an individual copy on each target.

Argument	Values
-action	**append**
-status_not_shared	
-status	**released**

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. "Started" indicates that the start action is completed. If the handler is placed in the start action, other handlers placed in the same action will not consider the workflow ACL.

> **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 is applied to the task and all subsequent tasks in the workflow process unless it is changed by another instance of the **EPM-set-rule-based-protection** handler or the process completes.

You can also set workflow ACLs by editing the Named ACL attribute, which automatically updates this handler.

> **Note**
>
> - This handler affects the behavior of the tasks as well the targets. For example, the ACL can grant permission to promote or demote the tasks.
>
> - Accessors, such as approvers or the responsible party, are retrieved from the currently active tasks. So even if the named ACL is the same for two separate tasks, the actual user who gets access for each task could be different. For example, **waynej** is the responsible party for task 1, **bjorn** is the responsible party for task 2, and the ACL grants write access to the responsible party for both tasks. In this case, **waynej** gets write access for duration of task 1 and **bjorn** gets write access for duration of task 2.
>
> - If you have multiple workflow processes in effect at the same time for the same target object, and each process sets its own ACL, a user gets access if any of the ACLs grants that access. To deny access in that situation, all ACLs must deny that access.

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 **-acl=***named-ACL*

ARGUMENTS

-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.

Argument	Values
-acl	**engineering_release_start0**

- This example tells Access Manager to give write access to the responsible party only for the second task in a four-task workflow. The other three tasks are read-only.

  o **Task 1**—read-only access for all users.

  The **Vault** ACL gives read and copy access to users, but not write access.

Argument	Values
-acl	**Vault**

  o **Task 2**—write access for the responsible party.

  The **Grant-Write-to-RP** ACL gives write access only to the responsible party.

Argument	Values
-acl	**Grant-Write-to-RP**

  o **Task 3**—read-only access for all users.

  The **Vault** ACL revokes write access starting with this task.

Argument	Values
-acl	**Vault**

  o **Task 4**—read-only access for all users.

No handler is needed because the ACL in **Task 3** still applies.

- This example, when placed on the **Review Specification** task, tells Access Manager to give demote access to only the task's responsible party. Promote access is denied to everybody, including the responsible party.
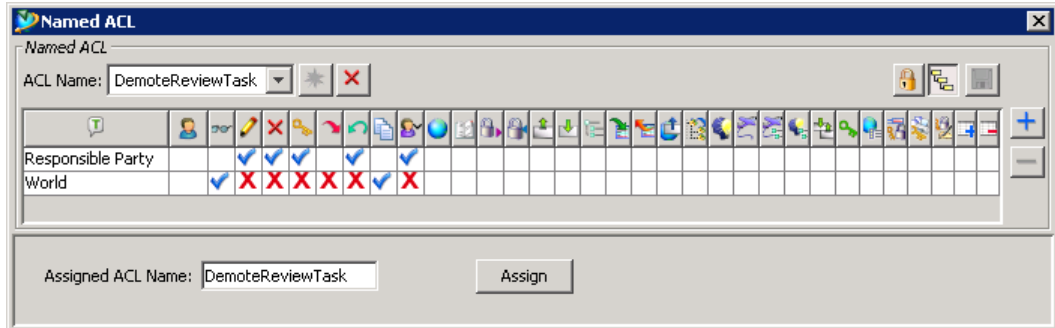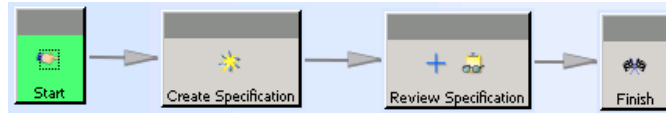
Argument	Values
-acl	**DemoteReviewTask**

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 **-property** argument, and the values are listed in the **-value** argument. There should be a one-to-one correspondence between the properties on the **-property** list and the values on the **-value** 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 -property=*list-of-properties* **-value=[** *comma-separated-value-list*] **[[[-to_attach=***attachment-type* **]** **[-to_relation=***relation-type*] **| -to_lov=***lov-name*]] **[[[-from_attach=***attachment-type* **]** **[-from_relation=***relation-type*]] | -from_lov=***lov-name*]] **[-include_type=***comma-separated-type-list* **|** **-exclude_type=***comma-separated-type-list*] **-bypass**

ARGUMENTS

-property

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 **-property** argument and the values indicated on the **-value** argument. The value types should be compatible with the property types. If a property listed on the **-property** argument does not exist for a specified update object, the update for the property is skipped.

Separate multiple properties with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

-value

Specifies zero or more values to be used to set the associated properties in the **-property** 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 run.

Separate multiple values with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

-to_attach

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.

Value	-to_att_type is used by itself	-to_att_type is used with -to_relation
TARGET	Updates target attachments.	Uses target attachments as a starting point when searching for derived objects. Updates only the derived objects.
REFERENCE	Updates reference attachments.	Uses reference attachments as a starting point when searching for derived objects. Updates only the derived objects.
BOTH	Updates both target and reference attachments.	Uses both target attachments and reference attachments as a starting point when searching for derived objects. Updates only the derived objects.

> **Note**
>
> Lower case values are also valid.

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_attach** argument, which identifies attachment types.

-to_attach value	-to_relation behavior
TARGET	Updates objects with the specified relation to the target attachments.
REFERENCE	Updates objects with the specified relation to the reference attachments.
BOTH	Updates objects with the specified relation to both the target and reference attachments.

-to_lov

Specifies an LOV to define which objects are to be updated.

For an overview of using LOVs in handlers, see *Lists of values as argument values*.

-from_attach

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_attach** 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.

Value	-from_attach is used by itself	-from_attach is used with -from_relation
TARGET	Reads property values from the first target attachment object.	Locates the first object with the specified relation to a target attachment object and reads property values from the related object.

Value	-from_attach is used by itself	-from_attach is used with -from_relation
REFERENCE	Reads property values from the first reference attachment object.	Locates the first object with the specified relation to a reference attachment object and reads property values from the related object.
BOTH	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.	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.

> **Note**
>
> Lower case values are also valid.

-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**.

• For references, use **IMAN_reference**.

• For BOM views, use **PSBOMViewRevision**.

This argument must be used with the **-from_attach** argument. A derived object is identified by starting with objects of the specified attachment type indicated by the **-from_attach** 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 **-property** list.

For an overview of using LOVs in handlers, see *Lists of values as argument values*.

-include_type

Updates specified objects only if their type matches one of the types on the list. Do not use this argument with the **-exclude_type** argument.

-exclude_type

Updates all specified objects unless their type is one of the types that appears on the **-exclude_type** list. Do not use this argument with the **-include_type** argument.

-bypass

Specifies that the user has bypass privileges and allows the property to be set.

LOV

For an overview of using LOVs in handlers, see *Lists of values as argument values*.

The LOV can contain multiple optional lines containing filter options followed by multiple lines containing multilevel object paths.

> **Note**
>
> For an overview and examples of multilevel object paths in handlers, see *Defining 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=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, 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 an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

(ItemRevision).IMAN_specification.(Dataset)

Attaches any datasets attached to the specification relation to any revisions found.

For more examples, see the Examples section.

PLACEMENT

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_attach** handler.

- The **-from_relation** argument must be used in conjunction with the **-from_attach** handler.

- The **-to_lov** argument is mutually exclusive of the **-to_attach** and **-to_relation** arguments.

  For an overview of using LOVs in handlers, see *Lists of values as argument values*.

- The **-from_lov** argument is mutually exclusive of the **-from_attach** and **-from_relation** arguments.

- Do not use the **-include_type** argument and the **-exclude_type** 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**.

Argument	Values
-property	**object_desc**
-value	**Component Template**
-to_attach	**TARGET**
-bypass	

- Sets the target object's **backup_date** date property to a value of **2009-03-01**.

Argument	Values
-property	**backup_date**
-value	**2009-03-01**
-to_attach	**TARGET**
-bypass	

- 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.

Argument	Values
-property	**archive_date,archive_info,has_variant_module**
-value	**$CURRENT_DATE,Archiving completed process,False**
-to_attach	**TARGET**
-bypass	

- Uses values from an object with a specifications relation to the reference attachment to set the target objects' properties.

Argument	Values
-property	**object_desc**
-value	**PROP::object_desc**
-from_attach	**REFERENCE**
-from_relation	**IMAN_specification**
-to_attach	**TARGET**
-bypass	

- 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.

Argument	Values
-property	**object_desc**
-value	**PROP::object_desc**
-from_attach	**REFERENCE**
-from_relation	**IMAN_specification**
-to_attach	**TARGET**
-to_relation	**IMAN_specification**
-bypass	

• 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.

Argument	Values
-property	**object_desc**
-value	**PROP::object_desc**
-from_attach	**REFERENCE**
-from_relation	**IMAN_specification**
-to_attach	**TARGET**
-to_relation	**IMAN_manifestation**
-include_type	**UGMASTER**
-bypass	

• 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.

Argument	Values
-property	**object_desc**
-value	**PROP::object_desc**
-from_attach	**REFERENCE**
-from_relation	**IMAN_specification**
-to_attach	**BOTH**
-to_relation	**IMAN_specification**
-include_type	**UGMASTER**
-bypass	

• Uses an LOV to obtain values that are used to update target property values.

Argument	Values
-property	**object_desc**

Argument	Values
-value	**PROP::object_desc**
-from_lov	**SYS_EPM_main_objects**
-to_attach	**TARGET**
-bypass	

- Uses an empty string to reset a property on a **TARGET** object. In this example, the **object_desc** property is reset to **""**.

Argument	Values
-property	**object_desc**
-value	
-to_attach	**TARGET**
-bypass	

- 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**.

Argument	Values
-property	**object_desc,sequence_limit**
-value	**,6**
-to_attach	**TARGET**
-bypass	

- 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 **""**.

Argument	Values
-property	**object_desc,sequence_limit,CUST_text_field**
-value	**,,**
-to_attach	**TARGET**
-bypass	

- 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.

Argument	Values
-property	**prop_soln**

Argument	Values
-value	**PROP::item_id**
-from_attach	**TARGET**
-to_attach	**TARGET**
-include_type	**CMII CR Form**
-to_relation	**IMAN_specification**

EPM-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

EPM-set-parent-result -value= true | false

ARGUMENTS

-value

Set to **true** or **false**.

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.

EPM-set-owning-project-to-task

DESCRIPTION

This handler takes the owning project (or program) from the first target object of the workflow and sets it for all Workflow objects (for example, **EPMTask**, and **EPMJob**). The system can restrict access to workflow objects properly since the project is set at the workflow object level. The Access Manager rule tree is also modified to deny general access, but can grant access based on project teams for the workflow (**EPMTask**) objects by adding a new named ACL for tasks (**EPMTask**) in projects. Once the workflow processes are created with these changes, the users from the owning project team of the first target object can access the workflow tasks, whereas other users cannot access them. The process initiator, responsible parties, and reviewers of the workflow are required to be members of the owning project to proceed with the workflow tasks.

SYNTAX

EPM-set-owning-project-to-task

ARGUMENTS

None.

PLACEMENT

Place on the Start action of a root task.

RESTRICTIONS

Uses only the owning project of first target to set it on workflow objects. It does not consider other assigned projects or the owning project of other targets. If the owning project is not set on first target object, this handler fails to operate.

EPM-set-job-protection

DESCRIPTION

Denies the **world:delete** and **world:write** process object protections, allowing an object ACL to be applied to an instance of an **EPMJob** object. This protection prevents the workflow process from being deleted when it 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
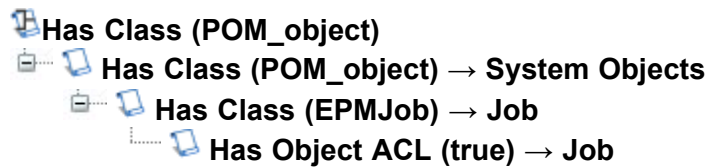
SYNTAX

EPM-set-job-protection

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action of a task.

RESTRICTIONS

None.

ERP-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

ERP-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-duration

DESCRIPTION

Defines time dependence during process design. The handler is triggered when the task is started. The five handler arguments are 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.

Due date calculations based on the duration setting in this handler consider the user's calendar and the value of the **Default_Base_Calendar_Preference** preference.

SYNTAX

EPM-set-duration -year=_year-value_ **-week=**_week-value_ **-day=**_day-value_
-hour=_hour-value_ **-minute**_minute-value_

ARGUMENTS

-year

Defines the number of years of the duration.

-week

Defines the number of weeks of the duration.

-day

Defines the number of days of the duration.

-hour

Defines the number of hours of the duration.

-minute

Defines the number of minutes of the duration.

PLACEMENT

Place on the **Start** action.

RESTRICTIONS

Argument values are limited to positive integers. The **Task Manager** daemon must be running or the application shuts down.

The **EPM-set-duration** handler, along with the following calendars and preferences, all work together, and are dependent on each other to define and control time parameters.

• The working time setting in the organization calendar.

• **SiteTimeZone**

• **Default_Base_Calendar_Preference**

• **Schedule Manager** preferences: **SM_Hours_Per_Day_Preference**, **SM_Hours_Per_Week_Preference**, and **SM_Hours_Per_Year_Preference**

Example

The end date is calculated as the sum of duration of the user input multiplied by the preference value.

To calculate time: `Year (SM_Hours_Per_Year_Preference) + Week (SM_Hours_Per_Week_Preference) + Day (SM_Hours_Per_Day_Preference) + Hours + Minutes.`

For example, the preference settings for a 24-hour duration calendar schedule are:

Year

SM_Hours_Per_Year_Preference=8760 (365 days x 24 hours)

Week

SM_Hours_Per_Week_Preference=168 (7 days x 24 hours)

Day

SM_Hours_Per_Day_Preference=24

EXAMPLES

- This example sets the task to be due 5 years, 4 weeks, 3 days, 2 hours, and 1 minute after it is started:

Argument	Values
-year	**5**
-week	**4**
-day	**3**
-hour	**2**
-minute	**1**

EPM-set-condition

DESCRIPTION

Condition tasks have a result attribute that you can set to one of these values: **True**, **False**, or **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. Using Business Modeler IDE conditions, the task can evaluate the condition criteria against target objects and user session information.

When queries are used for condition evaluation with this handler, one of the following queries is performed:

- Target query

   Performed on workflow process attachments.

- Task query

   Performed on the task to which this handler is added.

- Subprocesses query

   Performed on the subprocesses that the **Condition** task depends on.

Use **All | Any | None** to determine whether all, any, or none of the target attachments or subprocesses must meet the query criteria to set the result to **True**; these values apply only to target and subprocess queries.

The **-include_replica** argument queries the **Replica Proposed Targets** along with the targets if the **-query_type** argument is **target**.

SYNTAX

EPM-set-condition
{-condition_name=condition-name | { **-query=**query-name
[-query_type=task | target | sub_process] [-log] }}
[-check_targets=all | any | none] [-log] [-reference][-include_replica]

ARGUMENTS

-condition_name

Defines the BMIDE condition to be evaluated against target objects. The condition signature accepts a **WorkspaceObject** and **UserSession** in that sequence . The BMIDE condition in the handler argument is evaluated against the target objects based on the value of the **check_targets** argument. The handler decides the true or false path based on the evaluation result of BMIDE condition.

> **Note**
>
> The **-condition_name** and **-query** arguments are mutually exclusive.

-query
Defines the query to be run.

> **Note**
>
> The **-condition_name** and **-query** arguments are mutually exclusive.

-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.

- **sub_process**

    Performs a query on the subprocesses that the **Condition** task depends on.

-check_targets

This argument determines the target objects against which to evaluate the BMIDE condition or query.

It determines whether **all**, **any**, or **none** of the target attachments or subprocesses must meet the query criteria to set the result to **True**. This argument applies only to **Target** and **Sub-Processes** queries for the **-query** argument.

When used in conjunction with **-condition_name** argument, the BMIDE condition is evaluated against targets to determine whether **all**, **any** or **none** of the targets meet the condition.

If this argument is not specified and used in conjunction with **-condition_name** argument, the value for this is considered as **all** by default.

-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 or BMIDE condition to the task attachments references list.

-include_replica

(Optional) Queries the **Replica Proposed Targets** as well as the target objects if the **-query_type** is set to **target**.

PLACEMENT

- If the **-query_type** argument is set to **task** or **target**, place on the **Start** action.

- If the **-query_type** argument is set to **sub_process**, place on the **Complete** action.

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.
>
> No user interface support is provided to add this handler while using BMIDE conditions with the **-condition_name** argument. The handler must be added manually from 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**.

Argument	Values
-query	**CM II CN Type**
-query_type	**target**
-check_targets	**any**

- In this example, an **EPMTask** query, **BM - Has Multiple Targets**, uses the run-time property **num_targets** to count the workflow target objects. If the query result is more than one, the result on the **Condition** task is set to **True**.

> **Note**
>
> The **BM - Has Multiple Targets** query is created using the search class **EPMTask** and is not included in the Teamcenter install.

Argument	Values
-query	**BM - Has Multiple Targets**
-query_type	**task**

- In this example, the BMIDE **Fnd0DocRevSubTypes** condition is evaluated against all target attachments one-by-one. The condition evaluation returns **TRUE** if any of the target attachments is a subtype of **Document Revision**, and the workflow takes the **TRUE** path.

Argument	Values
-condition_name	**Fnd0DocRevSubTypes**
-check_targets	**any**

> **Note**
>
> The condition used in the handler example above:
>
> ```
> Fnd0DocRevSubTypes (WorkspaceObject o ,
> UserSession u) = ((o != null) AND
> u.fnd0ConditionHelper.fnd0isSubTypeOf
> (o, "DocumentRevision"))
> ```

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 environment variable.

SYNTAX

EPM-run-external-command -lov=*lov-name*

ARGUMENTS

-lov

Specifies the List of Values (LOV) used to configure all options.

LOV

For an overview of using LOVs in handlers, see *Lists of values as argument values*.

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%*.

```
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
```

o **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=QmBJ0uKNh9KRfCAAAAAAAAAAAAA
```

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
   OwningUser=Tim (tim) OwningGroup=Designers
```

o   **ARG**

Specifies optional arguments to be sent to the external command. The format is:

ARG~*%formatted string%*

For example:

```
ARG~-cfg=%$CONFIG_FILE%
ARG~-files=%$DATASET_FILE%
ARG~-data=%$DATA_FILE%
```

o   **DATA**

Specifies information to be extracted from targets, references, and their related objects. The possible formats are:

▪   **DATASETS**

DATA~DATASETS[=*options*]

writes a fixed format data file containing information about attached datasets that can optionally be exported with **INPUT ExportPath**.

This option is used to extract details about datasets attached to the objects specified by **INPUT~Target**. If **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 **process_tag_datasets.txt** in the current directory or in the directory specified using **INPUT~DataPath**. This file name can be extracted in a *%formatted strings%* using **$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 **UGPART** references in **UGMASTER**and **UGPART** datasets in the **IMAN_specification** relation.

```
DATA~DATASETS=IMAN_specification~UGMASTER,UGPART~UGPART
```

The datasets data file is written in a fixed format as follows:

```
item_id~rev_id~relation type~dataset type~dataset
```

```
name~dataset_tag~reference type~file name
```

- **LOV**

  For an overview of using LOVs in handlers, see *Lists of values as argument values*.

  DATA~LOV=*lov-name*

  writes a data 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:multi-level.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
  ```

- **%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%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**.

  ```
  %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.

  ```
  CFG~Project=%$TARGET.IMAN_master_form.project_id%
  ```

  If the project is **Project X**, the configuration file contains the following line:

  ```
  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:

```
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.

$DATA_FILE   Gets the name of the data file generated by the handler for **DATA~LOV**. The format of the name is:

*DataPath\process_tag_***data.txt**

or, if **CallPerTarget** is set to **YES**

*DataPath\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:

*DataPath\process_tag_***datasets.txt**

or, if **CallPerTarget** is set to **YES**

*DataPath\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.

Argument	Values
-lov	SYS_EPM_run_cad_checks

  The **SYS_EPM_run_cad_checks** LOV contains the following data:

LOV usage

Value	Description
INPUT~Target=$TARGET.(ItemRevision)	Specifies that the main objects from which data is to be extracted is the job targets which is of class **ItemRevision**.
	If multiple targets are found then the application will either be called separately for each target or once with all of the data from all targets, depending on the setting **CallPerTarget** which is defined just below.
INPUT~ErrorMsg1=Cad checks errors (Error %$SYSTEM_ERROR%)	Defines an error message which is displayed to the user if the application returns an error status.
INPUT~ErrorMsg2=Please see your e-mail for details	Defines an optional second error message which is displayed to the user as well as **ErrorMsg1**.
INPUT~Application= ${CUST_CAD_CHECK_APPLICATION}	Defines the external application which is to be run. This application is defined by a system environment variable, which in this example is **CUST_CAD_CHECK_APPLICATION**.
INPUT~CallPerTarget=YES	Calls the application for each target.
INPUT~DataPath=C:\WF\Data	Sets a path for data files.

LOV usage

Value	Description
INPUT~ExportPath=C:\WF\Exports	Sets a path for exported dataset files
CFG~JobTag=%$PROCESS.object_tag%	Writes the process tag (PUID) to the configuration file as **JobTag=**_Job Tag_.
CFG~JobName=%$PROCESS .object_name%	Writes the workflow process name to the configuration file as **JobName=**_Job Name_.
CFG~RevID=%$TARGET.item_revision_id%	Writes the target object revision ID to the configuration file as **RevID=**_RevID_.
CFG~ItemID=%$TARGET.item.item_id%	Writes the target object item ID to the configuration file as **ItemID=**_ItemID_.
CFG~Project= %$TARGET.IMAN_master_form.project_id%	Writes the target object Project ID, from the revision master form, to the configuration file as **Project=**_ProjectID_.
CFG~CadProc=${CUST_CAD_CHECK_PROC}	Writes the environment variable value to the configuration file as **CadProc=**_cad_proc_.
CFG~OwningUser=%$TARGET.owning_user%	Writes the target object owning user to the configuration file as **OwningUser=**_user_.
CFG~OwningGroup=%$TARGET.owning_group%	Writes the target object owning group to the configuration file as **OwningGroup=**_group_.
CFG~Email=%$USER.E_Mail%	Writes the current user's e-mail to the configuration file, where **E_Mail** is the label from the person form.
CFG~SMTPServer= ${CUST_RELEASE_SMTP_SERVER}	Writes the environment variable value to the configuration file.
CFG~FunctionsFile= ${CUST_RELEASE_FUNC_FILE}	Writes the environment variable value to the configuration file.
CFG~SysAdminEmail= ${CUST_RELEASE_SA_MAIL}	Writes the environment variable value to the configuration file.
CFG~AppsArray=Apps1	Writes the value **AppsArray=Apps1** to the configuration file.
CFG~WarningDir= ${CUSTOMER_RELEASE_WARNING_DIR}	Writes the environment variable value to the configuration file.
CFG~UPG=${UPG}	Writes the environment variable value to the configuration file.
CFG~Desc=%$TARGET.object_desc%	Writes the target object description to the configuration file.

LOV usage

Value	Description
DATA~DATASETS= IMAN_specification~UGMASTER~UGPART	Extracts information about **UGPART** references in **UGMASTER** datasets attached to the target revision.
ARG~cfg=%$CONFIG_FILE%	Sends the configuration file name as an argument.
ARG~files=%$DATASET_FILE%	Sends the dataset data file name as an argument.

- **Example 2**

   The following example shows the use of **DATA~LOV=lov-name** to extract various details.

Argument	Values
-lov	SYS_EPM_send_ecr_relation_data

   when the **SYS_EPM_send_ecr_relation_data** LOV contains the following data:

DATA~LOV=lov-name

Value	Description
INPUT~Target=(ItemRevision)	Specifies that the main object from which data is to be extracted is the job target which is of the **ItemRevision** class.
INPUT~Application= ${CUST_ECR_EXT_APPLICATION}	Defines the external application that is run. This application is defined by a system environment variable.
ARG~-item=%$TARGET.item.item_id %	Sends the target object's item ID as an argument to the application.
ARG~-rev=%$TARGET.item_revision_id %	Send the target object's revision ID as an argument to the application.
ARG~-dest=${CUST_RELEASE_DEST}	Send the environment variable's value as an argument to the application.
ARG~-type=ECR	Sends the value as an argument to the application.
ARG~-data=%$DATA_FILE%	Sends the name of the data file, to be produced by **DATA~LOV**, as an argument to the application.

DATA~LOV=lov-name

Value	Description
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

Value	Description
PROP:%item.id%~ECR Started~%creation_date%~%owning_user% ~%IMAN_master_form.ecr_prty%	Extract properties from the target revision.
OBJECT:(ItemRevision).Mini.(ItemRevision)!Buy Revision!Customer Revision!RawMaterial Revision	From any **ItemRevision** targets, find any **ItemRevision** objects attached to the **Mini** relation, except for specific types, for example, Buy Revision.
PROP:%item.item_id%~%item_revision_id%~Mini	Extract properties from any **Mini** relation revisions.
OBJECT:(ItemRevision).Major.(ItemRevision)!Buy Revision!Customer Revision!RawMaterial Revision	From any **ItemRevision** targets, find any **ItemRevision** objects attached to the **Major** relation, except for specific types, for example, **Buy Revision**.
PROP:% item.item_id %~% item_revision_id %~Major	Extract properties from any **Major** relation revisions.
OBJECT:(ItemRevision).Minor.(ItemRevision)!Buy Revision!Customer Revision!RawMaterial Revision	From any **ItemRevision** targets, find any **ItemRevision** objects attached to the **Minor** relation, except for specific types, for example, **Buy Revision**.
PROP:% item.item_id %~% item_revision_id %~Minor	Extract properties from any **Minor** relation revisions.

EPM-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. The **password** and **username** are authenticated before the task can be completed.

SYNTAX

EPM-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.

* Do not use this handler when the user logs on with PKI authentication. Use the **EPM-request-PKI-authentication** handler to prompt for the PKI PIN.

EPM-request-PKI-authentication

DESCRIPTION

Displays a PKI authentication box in the **Perform** dialog box or panel of the task within which it has been placed. Users must type their PKI PIN in the box before the task can be completed.

> **Note**
>
> This handler requires an environment configured with PKI enabled Teamcenter client communication system (TCCS) security services to use the PKI serial number as **userid**, with the value of the **WRKFLW_PKI_user_validation_fieldname** preference set to **SERIALNUMBER**.

SYNTAX

EPM-request-PKI-authentication

ARGUMENTS

None.

PLACEMENT

Place either on the **Perform** action of the **perform-signoffs** task or the **Complete** action of the following tasks:

• **Do** task

• **Condition** task

• **select-signoff-team** task

On a **Route** task, place on the **Complete** action of the **select-signoff-team** subtask of the **Review** task.

RESTRICTIONS

None.

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.

The **-include_replica** argument keeps or removes the **Replica Proposed Targets** along with the targets specified by the **-keep_targets** or **-remove_targets** argument.

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 .

SYNTAX

EPM-remove-objects {[{-remove_targets=types **| -keep_targets=**types**}]** **[{-remove_refs=**types **| -keep_refs=**types**}] | -lov=**lov-name**}** **[-include_replica]**

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 and examples of 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.

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.
>
> The **keep_targets** argument removes all targets of types that do not match the types specified by the **keep_targets** argument.

-remove_refs

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 **-keep_refs** and **-remove_refs** arguments are mutually exclusive.

-keep_refs

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*.

> **Note**
>
> The **-keep_refs** and **-remove_refs** arguments are mutually exclusive.
>
> The **keep_refs** argument removes all reference objects of types that do not match the types specified by the **keep_refs** argument.

-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 *Lists of values as argument values*. See the LOV row, next, for required LOV format.

-include_replica

(Optional) Keeps or removes the **Replica Proposed Targets** as well as the target objects specified by the **-keep_targets** or **-remove_targets** argument.

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 **EPM-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.

> **Note**
>
> If **EPM-remove-objects** and **EPM-set-rule-based-protection** are both
> used at the start of the same task, the workflow ACL is not active yet
> and cannot support **EPM-remove-objects**. The rule tree does not
> consider workflow ACLs before the entire task start action is completed,
> which is after the successful execution of all the handlers on the start
> action. In such a case, the **EPM-remove-objects** handler may need
> placing on the complete action to realize access changes asserted by the
> **EPM-set-rule-based-protection** handler on the start action.

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:

$TARGET.(Folder),(Item)

- This example retains only item revisions, removing all other targets:

Argument	Values
-keep_targets	**(ItemRevision)**

EPM-notify-signoffs

DESCRIPTION

Informs users of a **Route** task's status through operating system e-mail. Any surrogates for the specified users are also notified. If the **-attachment** argument is included in the definition of the **EPM-notify-signoffs** handler, the recipients also receive program mail. The recipients list is filled dynamically when running the **Review** task with the **Route** task. Links to the workflow process in the rich client, thin client, and Active Workspace are added based on the value of the **EPM_notify_url_format** preference.

> **Note**
>
> Use the **Mail_OS_from_address** preference to specify the **From** address displayed in the notification e-mail. The preference value must be a valid e-mail address.

SYNTAX

EPM-notify-signoffs
 [-subject=*string* | *$TARGET* | *string $TARGET string***]**
 [-comment=*string***]**
[-url={rich|dhtml|activeworkspace|none}]
 [-attachment= {target | process | reference }]
[-log]

ARGUMENTS

-subject

Displays the string identified by this argument in the subject line of the OS e-mail. The **-subject** argument supplies value options, such as "**-subject=***$TARGET*." Variants of the **-subject** argument values allow for a prefix or suffix string to the target name.

> **Note**
>
> If the "**-subject***$TARGET*" produces zero targets then the default subject line is used.

When no subject argument is provided, the default subject line for OS e-mail is **Review of "<Process_name (Parent Task_name)>" is in progress.**

-comment

User-defined comment that is embedded in the body of the e-mail.

-url

Inserts URLs into the notification e-mail that links to the workflow process in either the rich client (**rich**), thin client (**dhtml**), Active Workspace (**activeworkspace**), or all (no value). The URL is generated only when the **WEB_default_site_server** preference is set to the thin client server node name. Rich client URL functionality must be enabled for links to rich client workflow processes to launch the rich client.

- If the argument is specified with no value, rich client, thin client, and Active Workspace links are added to the notification e-mail.

- If the argument is not specified, the notification e-mail contains links depending on the value of the **EPM_notify_url_format** preference, which can be one or more of the following:

  o **rich**

  o **dhtml**

  o **activeworkspace**

  > **Note**
  >
  > One of the two following preferences must be defined:
  >
  > ■ **ActiveWorkspaceHosting.URL**
  >
  > ■ **ActiveWorkspaceHosting.WorkflowEmail.URL**

  o **none**

  No links are inserted into the notification e-mail.

- If the argument is not specified and the **EPM_notify_url_format** preference is not set, rich client, thin client, and Active Workspace are added.

-attachment
Adds an attachment to Teamcenter mail and adds table(s) containing information on the specified attachments to the operating system e-mail. Accept a comma separated or single value from following:

- **target**

  Attaches the target to the program mail.

- **process**

  Attaches the workflow process to the program mail.

- **reference**

  The task attachments reference list is included in the mail.

-log
Records notification activity in the workflow audit file.

PLACEMENT

Place on the **Complete** action of the **Notify** task.

RESTRICTIONS

None.

EPM-notify-report

DESCRIPTION

Sends a report through the operating system (OS) mail to all task reviewers. **EPM-notify-report** does not notify users through Teamcenter e-mail. If you want to send the report using Teamcenter e-mail, use the **EPM-notify** handler.

The **-report** argument differentiates **EPM-notify-report** handler from the **EPM-notify** handler. In notification e-mail, the **-report** argument appends a report describing the signoff data associated with the **perform-signoffs** task. **EPM-notify-report** is designated for use on the **perform-signoffs** task. The **EPM-notify** handler is used on any type of task.

> **Note**
>
> • Use the **Mail_OS_from_address** preference to specify the **From** address displayed in the notification e-mail. The preference value must be a valid e-mail address.
>
> • When placed on the **Start** action of **perform-signoffs** task, the **EPM-notify** or **EPM-notify-report** handlers are automatically re-executed when a signoff is delegated.

SYNTAX

EPM-notify-report
-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**

| **$PROCESS_OWNER | $TARGET_OWNER [**_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 | $TARGET | string $TARGET string***]**
[-comment=*string***]**

[-url={*rich|dhtml|activeworkspace|none***}]**
[-attachment= {target | process | reference }]

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.

- **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.

-recipient

(Optional) Specifies the task reviewers to receive notification. Any surrogates for the specified users are also notified. 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:

  o **$RESOURCE_POOL_ALL**

  o **$RESOURCE_POOL_SUBSCRIBED**

  o **$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:

  o **$GROUP**

    The current user's current group.

  o **$ROLE**

    The current user's current role.

  o **$TARGET_GROUP [***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.

  o **$PROCESS_GROUP**

    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:

o **$GROUP**

The current user's current group.

o **$ROLE**

The current user's current role.

o **$TARGET_GROUP [***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.

o **$PROCESS_GROUP**

The owning group of the workflow process.

> **Note**
>
> The **$ROLE_IN_GROUP** keyword (formerly **$ROLEINGROUP**) cannot be used. Use **allmembers:$GROUP::$ROLE** instead.

- **$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.

- **$PROCESS_OWNER**

Sends notification to the workflow process owner.

- **$TARGET_OWNER [***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.

- **$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
  $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.

> **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.
>
> 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_***XXXXX* 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

Displays the string identified by this argument in the subject line of the OS e-mail. The
-subject argument supplies value options, such as "**-subject=***$TARGET*." Variants of
the **-subject** argument values allow for a prefix or suffix string to the target name.

> **Note**
>
> If the "**-subject***$TARGET*" produces zero targets then the default subject
> line is used.

Reports are formatted by type and e-mailed with a default subject line.

•   The progress report (`report=progress`) default subject line is: **Review of
     "Process_name (Task_name)" is in progress.**

•   The level report (`report=level`) default subject line is: **"Process_name
     (Task_name)" is being <upcoming state>.**

- The rejection report (`report=rejection`) default subject line is: **"Process_name (Task_name)" is in rejected.**

-comment

(Optional.) Inserts the specified string in the body of the e-mail.

-url

(Optional.) Inserts a link to the workflow process into the notification e-mail, based on the value for **-url**. If no value is specified for **-url**, the rich client, thin client, and Active Workspace 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.

If the **-url** argument is not defined and the **EPM_notify_url_format** preference is not set in the preference file, rich client, thin client, and Active Workspace links are added to the notification e-mail by default.

This argument and the **EPM_notify_url_format** preference can take the following values:

- **rich**

  Inserts a rich client link to the workflow process into the notification e-mail.

  > **Note**
  >
  > Rich client URL functionality must be enabled for links to rich client workflow processes to launch the rich client.

- **dhtml**

  Inserts a thin client (DHTML) link to the workflow process into the notification e-mail.

  > **Note**
  >
  > The URL is generated only when the **WEB_default_site_server** preference is set to the thin client server node name.

- **activeworkspace**

  Inserts an Active Workspace link to the workflow process into the notification e-mail.

  > **Note**
  >
  > One of the two following preferences must be defined:
  >
  > o **ActiveWorkspaceHosting.URL**
  >
  > o **ActiveWorkspaceHosting.WorkflowEmail.URL**

- **none**

  No links are inserted into the notification e-mail.

-attachment

Adds an attachment to Teamcenter mail and adds table(s) containing information on the specified attachments to the operating system e-mail. Accept a comma separated or single value from following options.

> **Warning**
>
> Hide target names from users without read access rights by using the **-url** argument.

- **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

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 **EPM-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 **EPM-notify-report** 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.

- 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 **EPM-notify-report** handler should be placed on **Complete** action of **perform-signoffs** task.

Argument	Values
-report	**progress**
-subject	**Manufacturing Release Process Completed**
-recipient	**user:smith, os:peters, os:john, allmembers:manufacturing, allmembers:::manager, addresslist:VendorList, $PROJECT_MEMBER**

- This example designates the workflow process owner as the recipient of a progress report with the subject **Manufacturing Release Process Completed**.

  The **EPM-notify-report** handler should be placed on **Complete** action of **perform-signoffs** task.

Argument	Values
-report	**progress**
-subject	**Manufacturing Release Process Completed**
-recipient	**$PROCESS_OWNER**

- This example builds a list of all users assigned as reviewers for the **perform-signoffs** task.

  The **EPM-notify-report** handler should be placed on **Start** action of **perform-signoffs** task.

Argument	Values
-report	**progress**
-recipient	**$PROPOSED_REVIEWERS**

- 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 **EPM-notify-report** handler on the **Start** action of the **perform-signoffs** task.

Argument	Values
-report	**review**
-subject	**TASK REVIEW NOTIFICATION**
-comment	**Please review the task**
-recipient	**$PROCESS_OWNER, $PROPOSED_REVIEWERS, $RESOURCE_POOL_SUBSCRIBED**

- This example illustrates creating a workflow process template with a **Review** task. Add the **EPM-notify-report** handler in the **Undo** action of the **perform-signoffs** task. Place an **EPM-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.

Argument	Values
-report	**rejection**
-subject	**TASK REJECTION & DEMOTE NOTIFICATION**
-recipient	**$RESOURCE_POOL_ALL, $PROCESS_OWNER, $PROPOSED_RESPONSIBLE_PARTY, $PROPOSED_REVIEWERS**

- 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 **EPM-notify-report** handler on the **Complete** action of the **perform-signoffs** task.

Argument	Values
-report	**Progress**
-subject	**Manufacturing Release Process Completed**
-recipient	**$REQUESTOR**

- This example builds a list of all users in the current task level where the handler has been placed and sends mail to all of them.

Argument	Values
-report	**Progress**

EPM-notify

Informs users of a task's status through e-mail.

The **EPM-notify** handler can send notifications to users through Teamcenter mail only if the **Mail_internal_mail_activated** preference is set to **True**.

The **-report** argument on the **EPM-notify-report** handler differentiates the **EPM-notify-report** handler from the **EPM-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 **EPM-notify-report** handler on the **perform-signoffs** task, whereas the **EPM-notify** handler is more generic and can be used on any type of task.

If you place the **EPM-notify** handler on the **Perform** action (**EPM_perform_action**), an email notification is sent each time a **Perform** sub-action is triggered. These multiple notifications can cause unnecessary processing.

For example, a handler on the **Perform** action is executed three times by the **Add Attachment** sub-action (**EPM_add_attachment_action**). If the handler is **EPM-notify**, reviewers receive the same notification at three different intervals.

In addition to **Add Attachment**, the **Perform** action can include the following sub-actions:

*   **Remove Attachment** (**EPM_remove_attachment_action**)

*   **Approve** (**EPM_approve_action**)

*   **Reject** (**EPM_reject_action**)

*   **Promote** (**EPM_promote_action**)

*   **Demote** (**EPM_demote_action**)

*   **Assign Approver** (**EPM_assign_approver_action**)

Use the **Mail_OS_from_address** preference to specify the **From** address displayed in the notification e-mail. The preference value must be a valid e-mail address.

When placed on the **Start** action of **perform-signoffs** task, the **EPM-notify** or **EPM-notify-report** handlers are automatically re-executed when a signoff is delegated.

> **Note**
>
> Use caution when entering special characters into argument fields of mail notification handlers. Depending on your configuration and mail client, using special characters and character entities in argument values may not display correctly in email notifications. These characters can interfere with the mail notification utility **tc_mail_smtp**, and should be tested before deployment.

SYNTAX

EPM-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
| $TARGET_OWNER | $PROCESS_OWNER
| $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 | $TARGET | string $TARGET string***]**
[-comment=*string***]**
[-url = {rich | dhtml | activeworkspace | none}]

   [-attachment={target | process | reference}]

ARGUMENTS

Adds an attachment to Teamcenter mail and adds attachment information for operating system e-mail. The value can be any of the following:

-recipient

Specifies the task reviewers receiving notification. Any surrogates for the specified users are also notified. 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.
$TARGET_GROUP[*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.
$PROCESS_GROUP	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**, **$TARGET_GROUP** and **$PROCESS_GROUP**.

> **Note**
>
> The **$ROLE_IN_GROUP** keyword (formerly **$ROLEINGROUP**) cannot be used. Use **allmembers:$GROUP::$ROLE** instead.

- **$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.

  If you use **$RESPONSIBLE_PARTY**, add the handler to the **Start** action of the task, not the **Assign** action.

- **$PROPOSED_RESPONSIBLE_PARTY**

  Sends e-mail to the member assigned as the proposed responsible party of the first target object in the workflow process.

- **$PROCESS_OWNER**

  Sends e-mail to the workflow process owner.

- **$TARGET_OWNER [*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_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 **$RESOURCE_POOL_**_XXXXX_ argument is not defined and the **$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:
>
> 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 string identified by this argument in the subject line of the OS e-mail. The **-subject** argument supplies value options, such as "**-subject=**_$TARGET_." Variants of the **-subject** argument values allow for a prefix or suffix string to the target name.

> **Note**
>
> If the "**-subject**$TARGET$" produces zero targets then the default subject line is used.

When no subject argument is provided, the default subject line for OS e-mail is **"Process_name (Task_name)" is being <upcoming state>.**

-comment

Embeds user-defined comments in the body of the e-mail.

-url

Insert links to the workflow process into the notification e-mail, based on values for **-url**. If no value is specified for **-url**, the rich client, thin client, and Active Workspace 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.

If the **-url** argument is not defined and the **EPM_notify_url_format** preference is not set in the preference file, rich client, thin client, and Active Workspace links are added to the notification e-mail by default.

This argument and the **EPM_notify_url_format** preference can take the following values:

- **rich**

  Inserts a rich client link to the workflow process into the notification e-mail.

  > **Note**
  >
  > Rich client URL functionality must be enabled for links to rich client workflow processes to launch the rich client.

- **dhtml**

  Inserts a thin client (DHTML) link to the workflow process into the notification e-mail.

  > **Note**
  >
  > The URL is generated only when the **WEB_default_site_server** preference is set to the thin client server node name.

- **activeworkspace**

  Inserts an Active Workspace link to the workflow process into the notification e-mail.

> **Note**
>
> One of the two following preferences must be defined:
>
> o   **ActiveWorkspaceHosting.URL**
>
> o   **ActiveWorkspaceHosting.WorkflowEmail.URL**

- **none**

  No links are inserted into the notification e-mail.

-attachment

Adds an attachment to Teamcenter mail and adds table(s) containing information on the specified attachments to the operating system e-mail. Accept a comma separated or single value from the following options.

> **Warning**
>
> Hide target names from users without read access rights by using the **-url** argument.

- **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

There are no specific restrictions on placement for this handler except the following:

- When **$REVIEWERS** or **$UNDECIDED** is used as the keyword, place on the **Start** or **Complete** action of the **perform-signoffs** task.

- When **$RESPONSIBLE_PARTY** is used as the keyword, place on the **Start** action of the task, not the **Assign** action.

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.

Argument	Values
-subject	**Lower Right Subassembly Review**
-recipient	**DistributionList:design, DistributionList:qualityControl**
-comment	**Please review new subassembly and report any concerns directly to the Product Manager**
-attachment	**target, process**

- This example sends an e-mail and Teamcenter mail to the designated responsible party for the task. If the responsible party is a resource pool, no e-mail is sent.

Argument	Values
-recipient	**$RESPONSIBLE_PARTY, $RESOURCE_POOL_NONE**

- 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**.

Argument	Values
-subject	**Manufacturing Release Procedure Completed**
-recipient	**OS:peters, OS:john, User:smith, Group:manufacturing, Role:manager, DistributionList:purchasing**

- 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**.

Argument	Values
-subject	**Manufacturing Release Procedure Completed**
-recipient	**OS:peters, OS:john, User:smith, allmembers:manufacturing::, $CHANGE_REVIEW_BOARD**

- This example designates the recipient **PROCESS_OWNER** of an email with the subject **"Process Notification for Design_item"** when **Design_item** is the first target object of the workflow process.

Argument	Values
-subject	**Process Notification for $TARGET**

Argument	Values
-recipient	**$PROCESS_OWNER**

EPM-move-attached-objects

DESCRIPTION

Changes or copies workflow attachments from one attachment type to another. If the handler requires attaching replica objects as workflow targets, the handler attaches them as **Replica Proposed Targets**.

SYNTAX

EPM-move-attached-objects
-from_attach=_attachment-type_ **-to_attach=**_attachment-type_
[-include_type=_comma-separated-type-list_ **|**
-exclude_type=_comma-separated-type-list_**]**
[-copy]

ARGUMENTS

-from_attach

Specify one of the following attachment types from which the attached objects should be selected. This is a mandatory argument.

- **target**

- **reference**

- **problem_item**

- **solution_item**

- **impacted_item**

-to_attach

Specifies one of the following new attachment types for the attached objects. This is a mandatory argument.

- **target**

- **reference**

- **problem_item**

- **solution_item**

- **impacted_item**

-include_type

(Optional) Specifies the object types whose attachment type is to be changed. The handler changes the attachment type defined in the **-from_attach** argument of objects that are the types or their subtypes specified in this argument. Do not use this argument with the **-exclude_type** argument.

Separate multiple types with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

-exclude_type

(Optional) Ignores the object types specified by this argument. Attachments to these objects are not changed by this handler. Do not use this argument with the **-include_type** argument.

Separate multiple types with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

-copy

(Optional) Adds the attachments with the new relation defined by the **-to_attach** argument and leaves the attachments with the original relation. If this argument is not specified, the objects are removed from the attachment type specified by the **-from_attach** argument.

PLACEMENT

Place on the **Start** or **Complete** action of any task. Do not place on the **Perform** action.

RESTRICTIONS

None.

EPM-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 Manager** 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. The **Task Manager** daemon must have been installed using Teamcenter Environment Manager.

SYNTAX

EPM-late-notification -recipient=*user* | *group* | **$OWNER**
| **$REVIEWERS** | **$PROPOSED_REVIEWERS**
| **$RESPONSIBLE_PARTY** | **$PROPOSED_RESPONSIBLE_PARTY**
| **$UNDECIDED**
| **$PROJECT_ADMINISTRATOR**
| **$PROJECT_TEAM_ADMINISTRATOR**
| **$PROJECT_AUTHOR** | **$PROJECT_MEMBER**
| **$TARGET_OWNER** | **$PROCESS_OWNER**
| **$RESOURCE_POOL_ALL** | **$RESOURCE_POOL_NONE**
| **$RESOURCE_POOL_SUBSCRIBED**
| **$REQUESTOR**
| **$ANALYST**
| **$CHANGE_SPECIALIST1**
| **$CHANGE_SPECIALIST2**
| **$CHANGE_SPECIALIST3**
| **$CHANGE_REVIEW_BOARD**
| **$CHANGE_IMPLEMENTATION_BOARD** | *distribution-list*

ARGUMENTS

-recipient

- *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.

- **$PROPOSED_REVIEWERS**

  Sends e-mail to all members assigned as the proposed reviewers of the first target object in the workflow process.

- **$RESPONSIBLE_PARTY**

  Specifies the responsible party of 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.

- **$UNDECIDED**

  Specifies 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 notifications to those members. The project team is determined by the project team associated with the first target object.

- **$TARGET_OWNER**

  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.

- **$PROCESS_OWNER**

  Sends e-mail to the workflow process owner.

- **$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_PARTY**, 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_PARTY**.

    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_PARTY**, 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.

-

    $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.

- *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 **EPM-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.

Argument	Values
-recipient	**$REVIEWERS, $OWNER**

*   This example sends e-mail to reviewers of the task who have not performed the signoff.

Argument	Values
-recipient	**$UNDECIDED**

*   This example sends e-mail to user **Smith**, a distribution list (**VendorList**), and members of the **Purchase** group.

Argument	Values
-recipient	**Smith**,**VendorList**, **Purchase**

> **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**.

EPM-invoke-system-action

DESCRIPTION

Runs an external command (specified with the **-command** 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 **-command** argument (for example, **-command=**_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

EPM-invoke-system-action -command=_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] [-comment=_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

-command=*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 color or 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 color or 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=_[[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.

-comment=*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:*responsible-party***[; Task:***task-name***]**

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=[User:*user-id***] [; Group:***group***] [; Role:***role***] [; Level:***level***]**

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, a comma, or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

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.

-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, **ItemRevision** matches objects of the type **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.

-list_sibling_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.

-depth=_maximum-recursion-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 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.

-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.

Place on the **Start** or **Complete** action of any task. If this handler is configured to set the signoff decisions on a **perform-signoffs** task (for example, if the **-comment** argument is specified), then place on the **Complete** action of the **perform-signoffs** task.

- Do not add to a workflow process containing *any* handler using resource pools.

- You cannot use the **-trigger_on_go** argument to start a task if any of the tasks previous to it in the workflow process are not complete.

- This example shows how to run the **tc_check_renderings_pl** script using the **-command** 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.

Argument	Values
-command	**tc_check_renderings_pl**

- This example shows how to run the **test_action.bat** script in a Windows system. The script is the following:

  ```
  set rc=2
  echo %rc% >> c:\temp\test.log
  exit 0
  ```

  It is used in the following workflow process:

  Create one signoff profile for the **Review** task and place the **EPM-invoke-system-action** handler on the **Complete** action of the **Review** task with the following arguments:

Argument	Values
-command	**c:\temp\test_action.bat**
-expand	
-debug	

- This example shows how to run the **test_action** script in an UNIX system. The script is the following:

```
#!/bin/sh
rc=2
export rc
echo $rc > /tmp/test.log
exit $rc
```

It is used in the following workflow process:
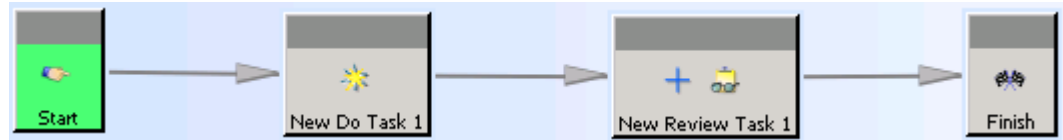
Create one signoff profile for the **Review** task and place the **EPM-invoke-system-action** handler on the **Complete** action of the **Review** task with the following arguments:

Argument	Values
-command	**/tmp/test_action**
-expand	
-debug	

• This example, placed on the **Complete** action of the **perform-signoffs** task, runs the **tc_check_install_assembly_pl** script using the **-command** argument. The script looks at a vehicle structure and checks to ensure each component has:

    o   A valid release status for the structure development stage and not **In Process**.

    o   All occurrences are precise and have an occurrence note indicating its usage at this stage.

    o   Every target attachment is a component of only one multilevel product item.

If the target of the original workflow process is a component of only one multilevel product item, the **-initiate_process** argument starts the **Initiate VPPS** workflow process specified in the Perl script and attaches the vehicle as a target and its work orders as references.

> **Note**
>
> The script is in the **sample\task_handlers** directory and should be placed in the *TC_ROOT*/**bin** directory.

Argument	Values
-command	**tc_check_install_assembly_pl**
-initiate_process	

EPM-inherit

DESCRIPTION

Inherits specified attachment types from a specified task.

SYNTAX

EPM-inherit -task=$PREVIOUS | $CALLER | $ROOT
-attachment=target | reference | signoffs

ARGUMENTS

-task

Task that contains the attachments to be inherited. Choices are the **$PREVIOUS** task, the parent task (**$CALLER**) or the **$ROOT** task. You can use multiple values by separating them with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

-attachment

Attachment types that are inherited from the tasks specified in the **-task** argument. Choices are **target**, **reference**, or **signoffs**. You can use multiple values by separating them with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EXAMPLES

• This example copies the reference attachments from the parent task to the current task.

Argument	Values
-task	**$CALLER**
-attachment	**reference**

• This example copies the signoffs from the previous task and the targets from the root task to the current task. The handler is placed on the **perform-signoffs** subtask of the second **Review** task.

Argument	Values
-task	**$PREVIOUS, $ROOT**
-attachment	**signoffs, target**

EPM-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.

If the **–required** argument is specified; the signoffs will be added as required signoffs which cannot be altered by users.

If the **–condition_name** argument is specified; the handler will add the reviewers only if the condition is met.

> **Note**
>
> A user is added to **select-signoff-team** task as a reviewer only once. If the same user participates in multiple signoff profiles, use the value **resourcepool:**_group_**::**_role_ with the **-assignee** argument.

SYNTAX

EPM-fill-in-reviewers
-assignee= [user:_user_ **| person:**_person_ **| addresslist:**_list_
| resourcepool:_group_**::**_role_ **| allmembers:**_group_**::**_role_
| user:PROP::_property_name_
| resourcepool:PROP::_property_name_
| allmembers:PROP::_property_name_
| $PROPOSED_RESPONSIBLE_PARTY | $PROPOSED_REVIEWERS | $USER
| $PROCESS_OWNER | $TARGET_OWNER [_type_**]**
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $PROJECT_AUTHOR | $PROJECT_MEMBER_[group_**::**_role]_
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1 | $CHANGE_SPECIALIST2 | $CHANGE_SPECIALIST3
| $CHANGE_REVIEW_BOARD | $CHANGE_IMPLEMENTATION_BOARD]
[-from_include_type=_object-type1[,object-type2,...]_**]**
[-from_exclude_type=_object-type1[,object-type2,...]_**]**
[-from_attach= target | reference | schedule_task]
[-from_relation=_relation-type_**]**
[-from_include_related_type=_object-type1[,object-type2,...]_ **|**
-from_exclude_related_type=_object-type1[,object-type2,...]_**]**
[-add_excess_as_adhoc]
[-target_task=_review-task-name_ **|** _multilevel-task-path_**]**
[-required]
[-project_scope=all | owning_project]
[-check_first_object_only=_true_ **|** _false_**]**
[-condition_name=_condition1_**]**
[-condition_scope=all | any | none]

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 *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 **$TARGET_GROUP[***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 **$PROCESS_GROUP**

     Owning group of the workflow process.

> **Note**
>
> The **$ROLE_IN_GROUP** keyword (formerly **$ROLEINGROUP**) cannot be used. Use **resourcepool:$GROUP::$ROLE** instead.

- **allmembers:**_group_**::**_role_

  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 **$TARGET_GROUP[**_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 **$PROCESS_GROUP**

  Owning group of the workflow process.

- **user:PROP::**_property_name_

  Adds the user specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::**_property_name_

  Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **allmembers:PROP::**_property_name_

  Adds all members of a group/role combination that is specified by the property name to the signoff member list.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **$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.

- **$USER**

Adds the current user to the signoff member list.

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** preference, as follows:

o **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.

o **2**

Attempts to match the current user's group/role values first, default values of which the current user is a member second.

o **3**

Attempts to match the current user's group/role values.

- **$PROCESS_OWNER**

Adds the workflow process owner to the signoff member list.

- **$TARGET_OWNER [***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[***group***::***role***]**

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.

If the **$PROJECT_MEMBER[***group***::***role***]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

You can specify a sub-group with the syntax *group***++***sub-group***::***role*.

- **$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.

-from_include_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_exclude_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**) to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). If this argument is not specified, the default is **target**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_relation

(Optional) Specifies the relation of the objects to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). It must be a valid relation.

• 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 **-from_attach** argument. A derived object is identified by starting with objects of the specified attachment type indicated by the **-from_attach** argument and then locating the first secondary object with the specified relation indicated by the **-relation** argument.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_include_related_type=object-type1[,object-type2]

(Optional) Specifies the related object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

This argument should not be used with the **-from_exclude_related_type** argument.

-from_exclude_related_type=object-type1[,object-type2]

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

This argument should not be used with the **-from_include_related_type** argument.

-add_excess_as_adhoc

(Optional.) Adds the rest of the assignees as ad hoc users if the profile is satisfied.

-target_task

(Optional) Specifies either the single **Review** task name or multilevel task path to which the reviewers are added. The path is from the root task to the **select-signoff-team** subtask with the path levels separated with colons (:). For example: **Change Request Review:QA Review:select-signoff-team**

-required

(Optional) If specified, all signoffs added through this handler instance are marked as mandatory.

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the objects identified for assigning reviewers from. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects. Values are the following:

all   All objects should meet the condition. This is the default behavior if this argument is not supplied with the **–condition_name** argument.

any   Any object should meet the condition.

none   No object should meet the condition.

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**.

Argument	Values
-assignee	**user:tom, allmembers:engineering::**
-target_task	**$ROOT.Review Task 1**

•   This example shows the current user added as a reviewer.

Argument	Values
-assignee	**user:$USER**
-target_task	**Review Task 1**

•   This example designates members assigned as reviewers for the first target object as reviewers for the **Review** task called **Review Task 1**.

Argument	Values
-assignee	**$PROPOSED_REVIEWERS**
-target_task	**Review Task 1**

- 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**.

Argument	Values
-assignee	**user:tom, allmembers:engineering::,$REQUESTOR**
-target_task	**Review Task 1**

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 assigns all members of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the signoff team as optional signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

- This example assigns all members of the **Engineering** group and **Designer** role of the owning project team associated with the first target found by the system to the signoff team as optional signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-project_scope	**owning_project**
-check_first_object_only	

- This example assigns all members of the **Engineering** group and **Designer** role of all project teams associated with the first target found by the system to the signoff team as required signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-project_scope	**all**
-check_first_object_only	**true**
-required	

- This example assigns all members of the **Engineering** group and **Designer** role of the first project team associated with each target found by the system to the signoff team as optional signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-check_first_object_only	**false**

EPM-execute-follow-up

DESCRIPTION

Runs 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.

> **Note**
>
> The user is already authenticated in the instance of the same Teamcenter server. For this reason, the code does not perform the login process again and auto login flags are not checked.

SYNTAX

EPM-execute-follow-up   -command=_argument_

ARGUMENTS

-command

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.

```
/* Sample code; file: test_itk_main.c */
#include tc.h
#include pom.h
int ITK_user_main(
        int argc,        /* I number of command line arguments */
        char* argv[]   /* I list of command line arguments */
        )
/*
 *   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 ...\n"); fflush(stdout);
    job_tag_string = ITK_ask_cli_argument("-zo=");
    if (!job_tag_string)
    {
        printf ("ERROR: no process tag string passed\n");
        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 ...\n");
    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 */
}
```

EPM-display-form

DESCRIPTION

Displays specified forms attached to a specified *custom* task , which is an instance of the **EPMTaskTemplate**. By default, all attachments of the **FormType** object attached to the current task are displayed.

The custom task template is used to define custom forms and other site-specific tasks for the user to complete and is designed to accept customization. This template contains no innate customized interface behavior.

> **Note**
>
> Do not use this handler on other task templates, such as **Do**, **Review**, and **Route**. Other task templates have their own user interface that overrides any attached forms. The task templates either are not meant to display a customized interface (such as the **Add Status** task template) or already have customized interface behavior assigned (such as the **Review** and **Route** task templates).
>
> For example, the **Do** task template already has customized interface behavior assigned. While form handlers can be added to the **Do** task template, the template's original interface behavior is displayed, not the forms. If the default display required is a customized form, use an instance of the custom task template.

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* **[-source_task=***task-name.attachment-type***]**

ARGUMENTS

-type

Valid **FormType** object.

-source_task

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**

Argument	Values
-type	**ItemRevision Master**
-name	**MyForm**
-description	**My item revision form**
-target_task	**$ROOT.$REFERENCE**

- On the **Perform** action: **EPM-display-form**

Argument	Values
-type	**ItemRevision Master**
-source_task	**$ROOT.$REFERENCE**

- On the **Complete** action: **EPM-hold**

Argument	Values
true	

EPM-demote-on-reject

DESCRIPTION

Demotes the current task to the previous task, or to the task specified on the **-target_task** argument of the **EPM-demote** handler placed on the **Undo** action of the current task.

By default, the handler checks the approval quorum requirements at each rejection and demotes the task when the quorum limit cannot be met. Consider a **perform-signoffs** task assigned to seven reviewers with an approval quorum of three. The first four rejections do not demote the task. The fifth rejection, which would prevent the approval 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 requirement by setting this argument to **2**. The task demotes on the second rejection, instead of the fifth.

To set the number of rejections needed to the number where the quorum cannot be met, set **-num_rejections** to **-1**. Using the above example of seven reviewers with a quorum of three, the **-1** value sets the required number of rejections to five. When five rejections are recorded, the task is demoted.

> **Note**
>
> This handler takes precedence if success and failure paths exist.

SYNTAX

EPM-demote-on-reject [-num_rejections=*number-of-rejections***]**

ARGUMENTS

-num_rejections

Number of rejections required to demote the task.

Specifying **-1** reads the approval quorum value and demotes the task when the number of rejections recorded makes it no longer possible to meet the quorum.

This argument is optional.

PLACEMENT

Place on the **Perform** action of the **perform-signoffs** subtask of a **Review** task.

RESTRICTIONS

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:

    EPM-demote-on-reject

• This example demotes a process when the second rejection is received:

Argument	Values
-num_rejections	**2**

- This example demotes a process when the number of rejections recorded prevents the quorum from being met. For example:

  o If there are two reviewers and a quorum of one, both reviewers would have to reject the signoff.

  o If there are three reviewers and a quorum of two, two reviewers would have to reject the signoff.

  o If there are four reviewers and a quorum of two, three reviewers would have to reject the signoff.

Argument	Values
-num_rejections	**-1**

EPM-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.

> **Caution**
>
> Do not use this handler on any tasks other than **Review** tasks.

SYNTAX

EPM-demote [-target_task=*task-name***]**

ARGUMENTS

-target_task

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**.

Argument	Values
-target_task	**design**

EPM-debug

DESCRIPTION

Allows you to print information (for example, state, action, and arguments) about the last action triggered. Typically used for debugging.

SYNTAX

EPM-debug -comment=_string_

ARGUMENTS

-comment

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 runs by printing **Complete**, **action is executing** to the standard output device.

Argument	Values
-comment	**action is executing**

Note

This example assumes you have attached this handler to a **Complete** action.

EPM-create-sub-process

DESCRIPTION

This handler starts subprocesses from a workflow process. The new subprocess can take on attachments of the parent process, and those attachments can be grouped by property.

This action handler 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 of the target/reference objects of the parent process, finds the corresponding object type, and adds them as target/reference attachments of the new subprocess. This handler allows you to 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.

If you want the progress of the parent process to be dependent on the subprocess completing, use the **-dependency** argument with this handler and place the handler on the **Start** action of the parent task to start the subprocess correctly. However, the parent task checks if the dependent subprocess is complete only when the parent task reaches the **Complete** action.

For example, if you place this handler with the **-dependency** argument on a **Review** task, it starts the subprocess, allows users to select a signoff team and perform signoffs, then checks the subprocess for its completion status. If the subprocess is not complete when the signoffs are completed, an error is displayed.

The **-include_replica** argument adds the parent's **Replica Proposed Targets** to the newly created subprocesses.

> **Note**
>
> When this handler creates a subprocess, the process owner and responsible parties for the new subprocess are defined as the current session's user. It may not match the responsible party of the workflow task having this handler, particularly when the task is automated and its actions are triggered after completing a previous task.
>
> If the process owner and responsible parties should be different than the currently logged-in user, use or in the subprocess template.

SYNTAX

EPM-create-sub-process
-template=_process-template-name_
[-from_attach=Target | Reference | ALL]
[-to_attach=Target | Reference | ALL]
[-include_type=_object-type_**]**
[-exclude_type=_object-type_**]**
[-process_name=_name-for-process_**]**
[-description=_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_type=*type-of-related-components-to-be-included***]**
[-exclude_related_type=*type-of-related-components-to-be-excluded***]**
[-include_replica]
[-group_by_property=*property-to-be-used-for-grouping***]**

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 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_type=*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_type=*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 used identifies the new workflow process. You can use the **$TARGET** keyword, which is replaced by the target display name *targetname-item-name*.

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 the subprocess would be called **subprocess**. In this case, to include a number in the name, put it in the argument name and only one is created. If the workflow process name is not given and the **-multiple_process** argument is not used, the parent process name is **parentprocess**; in this case, it is **parentprocess:1**. The same is true for cases where there are no targets on the parent process.

If the workflow process name is not given, and the **-multiple_processes** argument is used, the name assigned is in the format of *subprocesstargetdisplayname-item-name*:*count*. In this case, that would be

item1/A-wheel:1, **item2/B-axle:2**, **item3/A-bearing:3**. In the case where the parent had no targets, the name is **parentprocess:1**.

If the workflow process name is given with the **$TARGET** keyword, such as **subprocess1_$TARGET**, and the **-multiple_processes** argument is used, the name assigned is in the format *subprocess1_subprocesstargetname-item-name*:*count* format. In this case, that is **subprocess1_item1/A-wheel:1**, **subprocess1_item2/B-axle:2**, **subprocess1_item3/A-bearing:3**. In a case where the parent had no targets, the name is **subprocess1_:1**.

This argument is optional.

-description=*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:

> **Change Approval:QA Review:perform-signoffs::Design Change:**
> **Part Review:perform-signoffs**

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.
>
> • By default, if you do not use this argument, the signoff details for the subprocess are not included in the parent process signoff report for standard tasks. To include the details for an independent subprocess, change the value of the **WRKFLW_signoff_report_show_sub_process** preference.

-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_type**, and **-exclude_related_type** 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_type**, and **-exclude_related_type** 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_type=_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.

- 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_type=_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_type** and **-exclude_related_type** arguments can be used in conjunction with each other. If used in conjunction, the **-include_related_type** argument takes precedence; first the objects are processed against **-include_related_type**, and then **-exclude_related_type**.

-include_replica

(Optional) Adds the parent's **Replica Proposed Targets** to the newly created subprocesses under these conditions:

- If the **-from_attach** argument specifies either **Target** or **ALL**, the **Replica Proposed Targets** are also attached to subprocess with the targets.

- If the **-to_attach** argument specifies **Target** and any of the qualified objects are replicas, they are attached as **Replica Proposed Targets** instead of targets.

- If the **-include_replica** argument is not used, the handler does not add the **Replica Proposed Targets** attachments to the subprocess.

-group_by_property

- Input attachments are grouped according to the property assigned such as **object_type** and **object_owner**. One subprocess is spawned for each group. Each subprocess has objects (attachments) in that group.

- When used with the **-multiple_processes** arguments , one subprocess is spawned for each target object.

This argument is optional, but must be used with **-from attach**.

Place in the **Start** or **Complete** action of a task template.

> **Note**
>
> If you use the **-dependency** argument and the current task is dependent 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.

- When using **-relation** or **-process assembly,** the targets/reference attachments for the subprocess are processed based on the secondary related/assembly components of the parent target/reference attachments.

- 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.

- The **-depth** and **-rev_rule** arguments are used only when the **-process_assembly** argument is used. The **-exclude_related_type** and **-include_related_type** arguments are used only when **-process_assembly** or **-relation** is used.

- For the **group_by_property** argument, these p are not supported: **PROP_operationinput**, **PROP_unknown**, or properties containing multiple values, for instance, **array/list**.

- For the **group_by_property** argument, these property value types are not supported: **PROP_external_reference**, **PROP_untyped**.

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 dependent 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 **Change Approval** 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	**Change Approval**
-dependency	**::**
-process_name	**SubProcess_001**

- The example creates a new workflow process using the **Change Approval** template with no attachments. The **-process_name** and **-process_desc** are optional.

Argument	Values
-template	**Change Approval**
-process_name	**0006/A_Change Approval**
-description	**This is a demo description text**

- This example creates a new workflow process on the **Change Approval** 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 *Parentprocess*:*count* format. The workflow process description is left blank.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**ALL**

- This example creates a new workflow process on the **Change Approval** template by inheriting all the target attachments of the parent process as target attachments for the subprocess.

Argument	Values
-template	**Change Approval**
-from_attach	**TARGET**
-to_attach	**TARGET**

- This example creates a new workflow process on the **Change Approval** template by inheriting all the attachments (target and reference) of the parent process as target attachments for the subprocess.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**TARGET**

- This example launches a new workflow process on the **Change Approval** template. All target and reference attachments of the **ItemRevision** and **UGMASTER** types of the parent process are attached as targets for the new process.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**TARGET**
-include_type	**ItemRevision, UGMASTER**

- This example launches a new workflow process on the **Change Approval** 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.

Argument	Values
-template	**Change Approval**
-include_type	**ItemRevision, UGMASTER**
-from_attach	**TARGET**
-to_attach	**ALL**

- This example launches a new workflow process on the **Change Approval** template. All objects of the **ItemRevision** type of the parent process are excluded as targets for the new workflow process.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**TARGET**
-exclude_type	**ItemRevision**

* This example launches a new workflow process on the **Change Approval** template by specifying the **-include_type** and **-exclude_type** arguments. It specifies the list of attachment types to be included in **-include_type** and the list of types to be excluded in **-exclude_type**. This argument launches a subprocess with only **ItemRevision**.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**ALL**
-include_type	**ItemRevision**
-exclude_type	**UGMASTER**

* This example launches a new workflow process on the **Change Approval** 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 **Change Approval_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.

Argument	Values
-template	**Change Approval**
-dependency	**DesignReview:DoChecklist::Change Approval_001:QA Review:perform-signoffs**

* This example launches a new workflow process using the **Change Approval** 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 **Change Approval_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.

Argument	Values
-template	**Change Approval**
-dependency	**::Change Approval_001:QA Review:perform-signoffs**

- This example launches new workflow processes on the **Change Approval** 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.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**TARGET**
-include_type	**ItemRevision**
-multiple_processes	

- 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.

Argument	Values
-template	**Change Approval**
-from_attach	**TARGET**
-to_attach	**TARGET**
-multiple_processes	
-process_assembly	
-depth	**All**
-rev_rule	**Latest Working**

- The following handler configuration starts a subprocess on the **UGMaster** dataset attached to the target objects with **Iman_specification** relation.

Argument	Values
-template	**Change Approval**
-from_attach	**TARGET**
-to_attach	**TARGET**
-multiple_processes	
-relation	**Iman_specification**
-include_related_type	**UGMaster**

- 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.

Argument	Values
-template	**Change Approval**
-from_attach	**TARGET**
-to_attach	**TARGET**
-multiple_processes	
-process_assembly	
-depth	**All**
-rev_rule	**Latest Working**
-relation	**Iman_specification**

- The following handler configuration starts a subprocess using the **Change
  Approval** template. All target objects of the **Dataset** type except for **MSWord** type
  objects are attached as targets to the subprocess.

Argument	Values
-template	**Change Approval**
-from_attach	**TARGET**
-to_attach	**TARGET**
-include_type	**Dataset**
-exclude_type	**MSWord**

- The following configuration initiates the parent process on 000020/A (with
  assembly components) and 001180/A (with a dataset).

Argument	Values
-template	**SubProcess1**
-from_attach	**ALL**
-to_attach	**ALL**
-relation	**IMAN_specification**
-process_assembly	
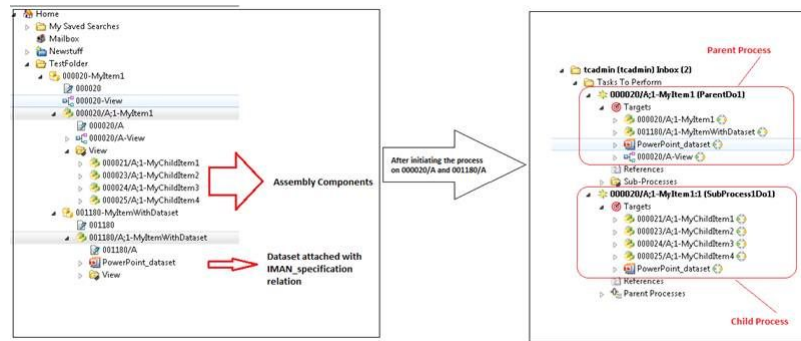
- The following handler configuration starts a subprocess using the **Change Approval** template. It spawns a Change Approval subprocess for each group formed.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-to_attach	**ALL**
-group_by_property	**Object_type**

- The following handler configuration starts a subprocess using the **Change Approval** template. It spawns one **Change Approval** subprocess for each target object in each group. The subprocesses spawned are named per the value in the **-process_name** argument.

Argument	Values
-template	**Change Approval**
-from_attach	**ALL**
-group_by_property	**Object_type**
-to_attach	**ALL**
-multiple_processes	
-process_name	**newSubprocess**

RESTRICTIONS ON ARGUMENTS

These examples show how *not* to use this handler.

- Do not create a workflow process without specifying the **-template** name.

Argument	Values
-process_name	**0006/A_Change Approval**
-from_attach	**TARGET**
-to_attach	**TARGET**

- Do not create a workflow process with the **-multiple_processes** argument but not providing the **-from_attach** and **-to_attach** arguments.

Argument	Values
-template	**Change Approval**
-multiple_processes	

- Do not create a workflow process by only specifying either one of the arguments: **-from_attach** or **-to_attach**.

Argument	Values
-template	**Change Approval**
-from_attach	**TARGET**

EPM-create-status

DESCRIPTION

Attaches the specified status type to the root task.

SYNTAX

EPM-create-status -status=status-type

ARGUMENTS

-status

Adds the specified status type to the root task. If this argument is not supplied, the task name where the handler is attached is used. The name provided should be the name of a status type already defined in the , not the display name.

If the status type is not already defined, 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.

Argument	Values
-status	**Released**

EPM-create-relation

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_attachment= target | reference**
-primary_type=_type-of-primary-object_ **-secondary_attachment=target | reference**
-secondary_type=_type-of-secondary-object_

ARGUMENTS

-relation

The relation type to be created.

-primary_attachment

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.

This argument checks for the exact type name and does not consider the subtypes.

-secondary_attachment

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.

This argument checks for the exact type name and does not consider the subtypes.

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:

Argument	Values
-relation	IMAN_requirement
-primary_attachment	target

Argument	Values
-primary_type	**ItemRevision**
-secondary_attachment	**reference**
-secondary_type	**UGPART**

EPM-create-form

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-create-form, EPM-display-form, and EPM-hold

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**]**
[[-property=field-name**] [-value=**value**]] [-target_task=**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**.

-property

Specifies the particular field of the form that has a default value. Users can choose to set the default value to more than one field by adding the field names separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. The default value for each field is set by the **-value** argument. Do not use this argument for field names of **Typed_Reference** and **Untyped_Reference** types. This argument is optional.

> **Note**
>
> Use this argument with the **-value** 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.

-value

Specifies the default value for a particular field of the form specified by the **-property** argument. Users can choose to set the default values for more than one field by adding the values separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference in the same order as listed in the **-property** argument values. Do not use this argument for field names of **Typed_Reference** and **Untyped_Reference** types. This argument is optional.

> **Note**
>
> Use this argument with the **-property** 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.

-target_task

Task name and attachment type receiving the new form as an attachment. The default value is the current task.

Accepts one of four keywords for *attachment-type*:

*   **$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 root task of the workflow process.

Argument	Values
-type	**ECN Form**
-name	**ECN**
-description	**Engineering Change Management Form**

Argument	Values
-target_task	**$ROOT.$REFERENCE**

- This example attaches the form as a target attachment to the current task:

Argument	Values
-target_task	**$ROOT.$TARGET**

To attach the form as a reference attachment to the current task, do not set the **-target_task** argument, because this is the default location this handler uses when this argument is not defined.

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= approve | reject]

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:

Argument	Values
-decision	**reject**

• This example requires the user to type comments before approving a signoff:

Argument	Values
-decision	**approve**

EPM-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

EPM-change-target-group-owner [-owner=*user-id***][-group=***group-id***]**

ARGUMENTS

-owner
Valid Teamcenter **user_id**.

-group
Valid Teamcenter **group_id**.

PLACEMENT

Place on the **Complete** action.

RESTRICTIONS

None.

EXAMPLES

*   This example changes the group and owner of the targets to **engineering** and **jim**, respectively.

Argument	Values
-owner	**jim**
-group	**engineering**

*   This example changes the only group of the targets to **production**.

Argument	Values
-group	**production**

*   This example changes only the owner of the targets to **smith**.

Argument	Values
-owner	**smith**

EPM-change-target-group

DESCRIPTION

Changes the group ownership of the target objects to the current **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

EPM-change-target-group

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action.

RESTRICTIONS

None.

EPM-change-ownership

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 (**-active=**action). If the handler is called as part of trigger to another action, the handler silently returns immediately.

SYNTAX

EPM-change-ownership -assignee=$REVIEWERS | $RESPONSIBLE_PARTY [-active= action **[-active=**other-action**]][-depth=**level**] [-debug]**

ARGUMENTS

-assignee

User to whom the ownership is given.

Use **$REVIEWERS** if this handler is used in a **Review** task. Use **$RESPONSIBLE_PARTY** otherwise.

[-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. You can use the following valid action names as values.

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 can be useful when the handler is placed on the **Perform** action. These actions automatically run the following **Perform** action handlers, raising the potential for unnecessary processing.

This argument is optional.

-depth

Recursion depth. This argument is optional and the default is set to **1**.

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.

Argument	Values
-assignee	**$REVIEWERS**

EPM-change-group-owner

DESCRIPTION

Changes the owning group for the item master of any item type whose revision is attached as target.

SYNTAX

EPM-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**.

Argument	Values
-group	**engineering**

EPM-change-all-started-to-pending

DESCRIPTION

Ensures that all tasks that are started, but not are not completed, are cleaned up at the conclusion of the workflow process.

SYNTAX

EPM-change-all-started-to-pending

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action of the root task.

RESTRICTIONS

None.

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
-assignee=$REVIEWERS | $RESPONSIBLE_PARTY
-action=check-in | check-out
  [-include_related_type=_dataset-type::bom-view-type_**]**
   [-include_replica]

ARGUMENTS

-assignee

> **Note**
>
> The **-assignee** argument is _optional_ and not required for **-action=check-in**.

Use **$REVIEWERS** for **Review** tasks. Use **$RESPONSIBLE_PARTY** otherwise.

> **Note**
>
> The object is checked out to the first reviewer.

-action

Action to check in (**check-in**) or check out (**check-out**) the objects.

-include_related_type

(Optional) Also check in/out the type specified in the form of _dataset-type_**::**_bom-view-type_. 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.

-include_replica

(Optional) Remote checks-in or remote checks-out the **Replica Proposed Targets** objects of the workflow along with the target objects. For remote check-outs, the objects are checked out to the current site executing the workflow.

PLACEMENT

- For **Review** and **Route** tasks where **-assignee=$REVIEWERS**:

   o   If **-action=check-out**, place the handler on the **Complete** action of the **select-signoff-team** subtask, or **Start** action of the **perform-signoffs** subtask.

   o   If **-action=check-in**, place the handler on the **Complete** action of the **perform-signoffs** subtask.

- For all other tasks or where **-assignee=$RESPONSIBLE_PARTY**:

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 **EPM-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.

Argument	Values
-assignee	$REVIEWERS
-action	check-out
-include_related_type	UGMASTER::view

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.

This example, placed on a **Review** task, checks in the objects once the task is completed and all reviewers sign off. You can place this action handler in the **Complete** action of the **Review** task using the **Check in** action, or in the **Complete** action of the **perform-signoffs** subtask using the **Check in** action.

Argument	Values
-action	check-in

EPM-auto-assign-rest

DESCRIPTION

Automatically makes the specified assignee the responsible party for any unassigned subtasks of the parent task to which this handler is added.

- 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

EPM-auto-assign-rest
-assignee= [user:*user* **| person:***person* **| resourcepool:***group***::***role*
| user:PROP::*property_name*
| resourcepool:PROP::*property_name*
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESS_OWNER | $TARGET_OWNER [*type***]**
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR]
| $PROJECT_AUTHOR | $PROJECT_MEMBER*[group***::***role]*
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3
[-from_include_type=*object-type1[,object-type2,...]***]**
[-from_exclude_type=*object-type1[,object-type2,...]***]**
[-from_attach= target | reference | schedule_task]
[-from_relation=*relation-type***]**
[-from_include_related_type=*object-type1[,object-type2,...]***] |**
-from_exclude_related_type=*object-type1[,object-type2,...]***]**
[-project_scope=all | owning_project]
[-check_first_object_only=*true* **|** *false***]**
[-condition_name=*condition1***]**
[-condition_scope=all | any | none]

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:

  o **$GROUP**

  Current user's current group.

  o **$ROLE**

  Current user's current role.

  o **$TARGET_GROUP[***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 **$PROCESS_GROUP**

  Owning group of the workflow process.

  > **Note**
  >
  > The **$ROLE_IN_GROUP** keyword (formerly **$ROLEINGROUP**) cannot be used. Use **resourcepool:$GROUP::$ROLE** instead.

- **user:PROP::***property_name*

Adds the user specified by the property name to the signoff member list for the task to which it is attached.

If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::***property_name*

Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **$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.

- **$PROCESS_OWNER**

  Adds the workflow process owner to the signoff member list and as the responsible party.

- **$TARGET_OWNER [***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**, **$PROJECT_AUTHOR**, **$PROJECT_MEMBER[***group***::***role***]**

  Dynamically makes the first project team member belonging to the role specified in the argument value as the responsible party. The project team is determined by the project team associated with the first target object.

  o   If the **$PROJECT_MEMBER[***group***::***role***]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

  o   If the value is specified as **$PROJECT_AUTHOR** or **$PROJECT_MEMBER[***group***::***role***]**, the relevant first project member is selected.

  o   You can specify a sub-group with the syntax *group***++***sub-group***::***role*.

- **$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.

-from_include_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be used to get the property value
from when a property is specified in the **-assignee** argument (for example,
-assignee=user:PROP::*property_name*). They must be valid object types.

-from_exclude_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be excluded when getting the
property value when it is specified in the **-assignee** argument (for example,
-assignee=user:PROP::*property_name*). They must be valid object types.

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**)
to get the property value from when a property is specified in the **-assignee** argument
(for example, **-assignee=user:PROP::***property_name*). If this argument is not
specified, the default is **target**.

-from_relation

(Optional) Specifies the relation of the objects to get the property value
from when a property is specified in the **-assignee** argument (for example,
-assignee=user:PROP::*property_name*). It must be a valid relation.

•	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 **-from_attach** argument. A derived object is
identified by starting with objects of the specified attachment type indicated by the
-from_attach argument and then locating the first secondary object with the specified
relation indicated by the **-relation** argument.

-from_include_related_type=*object-type1[,object-type2]*

(Optional) Specifies the related object types to be used to get the property value
from when a property is specified in the **-assignee** argument (for example,
-assignee=user:PROP::*property_name*). They must be valid object types.

Use this argument when a property is designated and you use the **-from_relation** argument.

This argument should not be used with the **-from_exclude_related_type** argument.

-from_exclude_related_type=_object-type1[,object-type2]_

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

Use this argument when a property is designated and you use the **-from_relation** argument.

This argument should not be used with the **-from_include_related_type** argument.

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the identified objects from which to assign tasks. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects. Values are the following:

all	All objects should meet the condition. This is the default behavior if this argument is not supplied with the **–condition_name** argument.
any	Any object should meet the condition.
none	No object should meet the condition.

PLACEMENT

Place on the **Start** action. Typically placed on the root task after the **EPM-assign-team-selector** handler.

RESTRICTIONS

None.

- In this example, a five-task workflow process containing the task templates below is initiated by user **Jones**. The **EPM-auto-assign-rest** handler is placed on the root task, and the **EPM-auto-assign** handler is placed on the fourth task, set with the **-assignee=$PROCESS_OWNER** argument.

  The workflow consists of a **Do** task, **Review** task, **Review** task, and **Do** task.

  Because the **EPM-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 **EPM-auto-assign -assignee=$PROCESS_OWNER** 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.

Argument	Values
-assignee	**user:Smith**

- 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.

Argument	Values
-assignee	**$PROPOSED_RESPONSIBLE_PARTY**

- 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.

Argument	Values
-assignee	**$ANALYST**

- This example assigns the first member of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the remaining tasks as responsible party.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

EPM-auto-assign

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 or 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

EPM-auto-assign [-subtasks]
[**-assignee= {user:**user | **person:**person | **resourcepool:**group::role
| **user:PROP::**property_name
| **resourcepool:PROP::**property_name
| **$PROPOSED_RESPONSIBLE_PARTY | $USER**
| **$PROCESS_OWNER | $TARGET_OWNER [**type**]**
| **$PROJECT_ADMINISTRATOR**
| **$PROJECT_TEAM_ADMINISTRATOR**
| **$PROJECT_AUTHOR | $PROJECT_MEMBER**[group::role]
| **$REQUESTOR | $ANALYST**
| **$CHANGE_SPECIALIST1**
| **$CHANGE_SPECIALIST2**
| **$CHANGE_SPECIALIST3**}]
[**-from_include_type=**object-type1[,object-type2,...]]
[**-from_exclude_type=**object-type1[,object-type2,...]]
[**-from_attach= target | reference | schedule_task**]
[**-from_relation=**relation-type]
[**-from_include_related_type=**object-type1[,object-type2,...] |
-from_exclude_related_type=object-type1[,object-type2,...]]
[**-target_task=**multilevel-task-path]
[**-project_scope=all | owning_project**]
[**-check_first_object_only=**true | false]
[**-condition_name=**condition1]
[**-condition_scope=all | any | none**]

ARGUMENTS

-subtasks

Propagates task assignments to subtasks of the current task (nonrecursively). Optional.

-assignee

Assigns as the responsible party for the task to which this handler is added either the specified person, user, resource pool, or the user or resource pool the specified keyword evaluates to.

Accepts one of the following in the format specified below:

- **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 **$TARGET_GROUP[***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 **$PROCESS_GROUP**

  Owning group of the workflow process.

  > **Note**
  >
  > The **$ROLE_IN_GROUP** keyword (formerly **$ROLEINGROUP**) cannot be used. Use **resourcepool:$GROUP::$ROLE** instead.

- **user:PROP::***property_name*

  Adds the user specified by the property name to the signoff member list for the task to which it is attached.

If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::**_property_name_

    Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

    If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **$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.

- **$PROCESS_OWNER**

    Adds the workflow process owner to the signoff member list and as the responsible party.

- **$TARGET_OWNER [**_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**, **$PROJECT_AUTHOR**, **$PROJECT_MEMBER[**_group_**::**_role_**]**

    Dynamically makes the first project team member belonging to the role specified in the argument value as the responsible party. The project team is determined by the project team associated with the first target object.

    o   If the **$PROJECT_MEMBER[**_group_**::**_role_**]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

    o   If the value is specified as **$PROJECT_AUTHOR** or **$PROJECT_MEMBER[**_group_**::**_role_**]**, the relevant first project member is selected.

    o   You can specify a sub-group with the syntax _group_**++**_sub-group_**::**_role_.

- **$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.

-from_include_type=_object-type1[,object-type2,...]_

(Optional) Specifies the object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

-from_exclude_type=_object-type1[,object-type2,...]_

(Optional) Specifies the object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**) to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). If this argument is not specified, the default is **target**.

-from_relation

(Optional) Specifies the relation of the objects to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). It must be a valid relation.

• 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 **-from_attach** argument. A derived object is identified by starting with objects of the specified attachment type indicated by the **-from_attach** argument and then locating the first secondary object with the specified relation indicated by the **-relation** argument.

-from_include_related_type=_object-type1[,object-type2]_

(Optional) Specifies the related object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

Use this argument when a property is designated and you use the **-from_relation** argument.

This argument should not be used with the **-from_exclude_related_type** argument.

-from_exclude_related_type=_object-type1[,object-type2]_

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

Use this argument when a property is designated and you use the **-from_relation** argument.

This argument should not be used with the **-from_include_related_type** argument.

-target_task

(Optional) Specifies the multilevel task path to which the reviewers are added. The path is from the root task to the subtask with the path levels separated with colons (:). For example: **Change Request Review:QA Review:perform-signoff**

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the identified objects from which to assign tasks. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects. Values are the following:

all	All objects should meet the condition. This is the default behavior if this argument is not supplied with the **–condition_name** argument.

any	Any object should meet the condition.
none	No object should meet the condition.

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.

Argument	Values
-subtasks	
-assignee	**user:Smith**

- This example makes the workflow process owner the responsible party for the task to which this handler is assigned.

Argument	Values
-assignee	**$PROCESS_OWNER**

- This example makes the engineer role within manufacturing group resource pool the responsible party for the task to which this handler is assigned.

Argument	Values
-assignee	**resourcepool:manufacturing::engineer**

- This example makes the responsible party group the responsible party for the task to which this handler is assigned.

Argument	Values
-assignee	**$PROPOSED_RESPONSIBLE_PARTY**

- 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.

Argument	Values
-assignee	**$PROJECT_ADMINISTRATOR**

- 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.

Argument	Values
-assignee	**$ANALYST**

- This example assigns the first member of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the task as responsible party.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

EPM-attach-related-objects

DESCRIPTION

Attaches the specified related objects of the target objects as target or 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 or reference attachments. If a secondary object is already part of the target list, it is ignored.

> **Note**
>
> If the **WRKFLW_allow_replica_targets** preference is set to **true** and if any replica object qualifies to be attached as a workflow target, that object is attached as a **Replica Proposed Target** to the workflow process. If the intended attachment type is not a target, the replica object is attached as the attachment type defined in **-attachment** argument.
>
> If the preference is set to **false** or is undefined, the handler reports an error and attaches replica objects as targets.
>
> Further, if the **-from_attach** argument is set to **schedule_task** and if the attached schedule task is a proxy link, the handler ignores the schedule task proxy link for any processing.

> **Note**
>
> If the handler attempts to attach related objects that are checked out, the workflow process fails. You can use a **Validate** task to branch to a workflow path to have the objects checked in.

> **Note**
>
> To replace the obsolete **EPM-attach-item-revision-targets** handler, use the following two instances of the **EPM-attach-related-objects**:
>
> • **EPM-attach-related-objects**
>
> | Arguments | Values |
> | --- | --- |
> | -relation | IMAN_specification |
> | -attachment | target |
>
> • **EPM-attach-related-objects**
>
> | Argument | Values |
> | --- | --- |
> | -relation | PSBOMViewRevision |
> | -attachment | target |

> **Note**
>
> Enable debugging functionality for this handler with the
> **TC_HANDLERS_DEBUG** environment variable.

EPM-attach-related-objects
{**-relation=***relation-name* | **-property=***property-name*}
[-include_related_type=*object-type1[,object-type2,...]* |
| **-exclude_related_type=***object-type1[,object-type2,...]***]] |**
[-lov=*lov-name***]**
-attachment= target | reference
[-from_attach= target | reference | schedule_task]
  [-allowed_status=*status1*
  [,null,*status2,...***] | * | all | any | null | none | IN_PROCESS]**
  [-disallowed_status=*status1*
  [,null,*status2,...***] | * | all | any | null | none | IN_PROCESS]**

-relation=*relation-name* **| -property=***property-name*

Specifies the relation or the property that is used to locate and attach secondary objects. You can use only one of these two arguments.

-relation=*relation-name*

Specifies the relation of the secondary objects to be attached to the target. The relation name must be valid for the relation type.

Relation type	Valid relation name
Manifestation	**IMAN_manifestation**
Specification	**IMAN_specification**
Requirement	**IMAN_requirement**
Reference	**IMAN_reference**
BOM view	**PSBOMViewRevision**

> **Note**
>
> You cannot use this argument with the **-property** argument.

-property=*property-name*

Specifies the target object property whose value lists the secondary objects to be attached to the target.

> **Note**
>
> You cannot use this argument with the **-relation** argument.

-include_related_type=_object-type1[,object-type2]_

Specifies object types to be attached.

They must be valid object types. This argument is optional.

This argument should not be used with the **-exclude_related_type** argument.

-exclude_related_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 **-include_related_type** argument.

-lov=_lov-name_

Specifies a list of values (LOV) to use to define which objects to attach.

Use only with the **-attachment**, **-allowed_status** and **-disallowed_status** arguments. This argument is mutually exclusive of the **-relation**, **-include_related_type**, and **-exclude_related_type** arguments.

For an overview of using LOVs in handlers, see _Lists of values as argument values_.

-attachment= target | reference

Attachment type with which the objects are attached to the workflow process.

-from_attach= target | reference | schedule_task

(Optional) Finds the related objects with the specified relation or property argument from the specified types of attachments (**target**, **reference**, or **schedule_task**).

-allowed_status=_status1_**[,null,**_status2_,**…] | * | all | any | null | none | IN_PROCESS**

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.

IN_PROCESS checks whether the object is currently in a workflow process.

> **Note**
>
> The **-allowed_status** and **-disallowed_status** arguments are mutually exclusive. If you use one of them, you cannot use the other in the same handler instance.

-disallowed_status=_status1_**[,null,**_status2_,**…] | * | all | any | null | none | IN_PROCESS**

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.

IN_PROCESS checks whether the object is currently in a workflow process.

> **Note**
>
> The **-allowed_status** and **-disallowed_status** arguments are mutually exclusive. If you use one of them, you cannot use the other in the same handler instance.

LOV

For an overview of using LOVs in handlers, see *Lists of values as argument values*.

The LOV can contain multiple optional lines containing filter options followed by multiple lines containing multilevel object paths.

> **Note**
>
> For an overview and examples of multilevel object paths in handlers, see *Defining 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:

- **REV RULE={LATEST|***Rule***}**

  Specifies the revision rule used to select the revision attached to the workflow process if initiated on an item. Use the **LATEST** keyword 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 an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

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.

PLACEMENT

Typically placed on the **Start** action of the root task so that 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 **EPM-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-rule-based-protection** handler to ensure that the required change access is asserted.

> **Note**
>
> If **EPM-attach-related-objects** and **EPM-set-rule-based-protection** are both used at the start of the same task, the workflow ACL is not active yet and cannot support **EPM-attach-related-objects**. The rule tree does not consider workflow ACLs before the entire task start action is completed, which is after the successful execution of all the handlers on the start action. In such a case, the **EPM-attach-related-objects** handler may need placing on the complete action to realize access changes asserted by the **EPM-set-rule-based-protection** handler on the start action.

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 **-attachment** argument to **target**, and one to attach the references using the **-attachment** 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:

Arguments	Values
-relation	**IMAN_specification**
-attachment	**target**

> **Note**
>
> If an object is already attached as target, it is not added.

- In this example, all objects listed in the **altid_list** property value are attached to the workflow process as target objects, when a workflow process is initiated on an item revision:

Arguments	Values
-property	**altid_list**
-attachment	**target**

> **Note**
>
> o The property named in the argument value must exist on the target.
>
> o 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:

Argument	Values
-relation	**IMAN_reference**
-attachment	**reference**

- This example attaches the BOM view revision type **View** as a target:

Argument	Values
-relation	**PSBOMViewRevision**
-include_related_type	**view**
-attachment	**target**

Alternatively, you can use these LOV settings:

Argument	Values
-lov	**SYS_EPM_attach_view_bvr**

where the **SYS_EPM_attach_view_bvr** LOV contains the value:

$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.

Argument	Values
-relation	**IMAN_specification**
-include_related_type	**UGMASTER, UGPART**
-attachment	**target**

Alternatively, you can use these LOV settings:

Argument	Values
-lov	**SYS_EPM_attach_UGMASTER_UGPART**

where the **SYS_EPM_attach_UGMASTER_UGPART** LOV contains the data:

$TARGET.(ItemRevision).IMAN_specification.UGMASTER,UGPART

• This example uses the **-exclude_related_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**.

Argument	Values
-relation	**IMAN_specification**
-exclude_related_type	**UGMASTER, Text**
-attachment	**target**

Alternatively, you can use these LOV settings:

Argument	Values
-lov	**SYS_EPM_exclude_UGMASTER**

where the **SYS_EPM_exclude_UGMASTER** LOV contains the data:

$TARGET.(ItemRevision).IMAN_specification.(*)!UGMASTER!Text

> **Note**
>
> Use an ***** for any class, then exclude **UGMASTER** and **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.

Argument	Values
-lov	**SYS_EPM_attach_main_objects**
-attachment	**target**
-allowed_status	**null**

Where the **SYS_EPM_attach_main_objects** LOV contains the data:

Value	Description
$TARGET.(ItemRevision).Specification.*	**Attach all objects in target revision Specification relation**
$TARGET.(ItemRevision).IMAN_specification. UGMASTER.UGPART-ATTR.UGPartAttr	**Attach all forms attached to datasets in target revision # Specification relation as a Form reference, but excluding the # form type UGPartAttr.**
$TARGET.(ItemRevision).PSBOMViewRevision.*	**Attach all BOM View Revisions in target revision**
$TARGET.(ItemRevision).Manifestation.(Form)	**Attach all forms in target revision Manifestation relation**

- 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**.

Argument	Values
-lov	**SYS_EPM_attach_main_objects**
-attachment	**target**
-allowed_status	**null, Pending**

Where the **SYS_EPM_attach_main_objects** LOV contains the data:

Value	Description
INCLUDE PARENTS = YES	**Set option for all lines to include all objects found**
REV RULE = LATEST	**Set the revision rule for any items**
$TARGET.(ItemRevision).IMAN_specification, PSBOMViewRevision.*.* ~	**Attach required objects from REVISION targets**
$TARGET.(Item).Revisions.*.IMAN_specification, PSBOMViewRevision.*.*	**Attach required objects from latest revisions in ITEM targets**
$TARGETS.(Folder).*.(Item).Revisions.* .IMAN_specification, PSBOMViewRevision.*.*	**Attach required objects from FOLDER targets**
$TARGETS.(Folder).*.(ItemRevision). IMAN_specification, PSBOMViewRevision.*.*	**Look for items and revisions in the folders**

ADDITIONAL
INFORMATION

With the addition of this handler, these handlers are deprecated:

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

Argument	Values
-relation	**IMAN_specification**
-attachment	**target**

EPM-attach-related-objects

Argument	Values
-relation	**PSBOMViewRevision**
-attachment	**target**

• Adding the **EPM-attach-related-objects** handler once using an LOV:

EPM-attach-related-objects

Argument	Values
-lov	**SYS_EPM_attach_default_objects**
-attachment	**target**

Where the **SYS_EPM_attach_main_objects** LOV contains the data:

$TARGET . (ItemRevision) . Specification, PSBOMViewRevision . *

EPM-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

EPM-assign-team-selector
-assignee= [user:user **| person:**person **| resourcepool:**group::role
| user:PROP::property_name
| resourcepool:PROP::property_name
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESS_OWNER | $TARGET_OWNER [type**]**
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $PROJECT_AUTHOR | $PROJECT_MEMBER[group::role]
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3]
[-from_include_type=object-type1[,object-type2,...]**]**
[-from_exclude_type=object-type1[,object-type2,...]**]**
[-from_attach= target | reference | schedule_task]
[-from_relation=relation-type**]**
[-from_include_related_type=object-type1[,object-type2,...] **|**
-from_exclude_related_type=object-type1[,object-type2,...]**]**
[-target_task=multilevel-task-path**]**
[-project_scope=all | owning_project]
[-check_first_object_only=true | false**]**
[-condition_name=condition1**]**
[-condition_scope=all | any | none]

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 **$TARGET_GROUP [***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 **$PROCESS_GROUP**

    Owning group of the workflow process.

- **user:PROP::***property_name*

  Adds the user specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::***property_name*

  Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **$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.

- **$PROCESS_OWNER**

  Adds the workflow process owner to the signoff member list.

- **$TARGET_OWNER [***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[***group***::***role***]**

  Dynamically makes the first project team member belonging to the role specified in the argument value as the responsible party. The project team is determined by the project team associated with the first target object.

  o If the **$PROJECT_MEMBER[***group***::***role***]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

  o If the value is specified as **$PROJECT_AUTHOR** or **$PROJECT_MEMBER[***group***::***role***]**, the relevant first project member is selected.

  o You can specify a sub-group with the syntax *group***++***sub-group***::***role*.

- **$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.

-from_include_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_exclude_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**) to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). If this argument is not specified, the default is **target**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_relation

(Optional) Specifies the relation of the objects to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). It must be a valid relation.

- 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 **-from_attach** argument. A derived object is identified by starting with objects of the specified attachment type indicated by the **-from_attach** argument and then locating the first secondary object with the specified relation indicated by the **-relation** argument.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_include_related_type=*object-type1[,object-type2]*

(Optional) Specifies the related object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

This argument should not be used with the **-from_exclude_related_type** argument.

-from_exclude_related_type=*object-type1[,object-type2]*

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

This argument should not be used with the **-from_include_related_type** argument.

-target_task

(Optional) Specifies the multilevel task path to which the reviewers are added. The path is from the root task to the **select-signoff-team** subtask with the path levels separated with colons (:). For example: **Change Request Review:QA Review:select-signoff-team**

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the identified objects from which to assign tasks. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects. Values are the following:

all	All objects should meet the condition. This is the default behavior if this argument is not supplied with the **–condition_name** argument.
any	Any object should meet the condition.
none	No object should meet the condition.

PLACEMENT

Place on the **Start** action of the root task.

RESTRICTIONS

None.

- This example assigns the user **jim** all **select-signoff-team** tasks in that workflow process.

Argument	Values
-assignee	**user:jim**

- This example assigns the person **Jim Smith** all **select-signoff-team** tasks in that workflow process.

Argument	Values
-assignee	**person:Jim Smith**

- This example assigns the owner of the workflow process all **select-signoff-team** tasks in that workflow process.

Argument	Values
-assignee	**$PROCESS_OWNER**

- This example assigns the user or resource pool assigned as the responsible party for all **select-signoff-team** tasks in that workflow process.

Argument	Values
-assignee	**$PROPOSED_RESPONSIBLE_PARTY**

- 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.

Argument	Values
-assignee	**$PROJECT_ADMINISTRATOR**

- 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.

Argument	Values
-assignee	**$REQUESTOR**

- This example assigns the first member of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the **select-signoff-team** task.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

EPM-assign-signoff-dynamic-participant

DESCRIPTION

Assigns the specified users or resource pools as the dynamic participant to the target attachment.

If the BMIDE constant **Fnd0ParticipantEligibility** is defined for the dynamic participant, the handler gets the corresponding group member which matches the group and role criteria defined in the BMIDE constant. If the user identified through the **–assignee** argument does not have the correct group and role membership, the handler reports an error and does not assign the user to the dynamic participant.

SYNTAX

EPM-assign-signoff-dynamic-participant
-name= {PROPOSED_REVIEWERS
| CHANGE_REVIEW_BOARD
| CHANGE_IMPLEMENTATION_BOARD}
[-assignee= [user:*user* | **person:***person* | **resourcepool:***group*::*role*
| user:PROP::*property_name*
| resourcepool:PROP::*property_name*
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESS_OWNER | $TARGET_OWNER [*type***]**
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $PROJECT_AUTHOR | $PROJECT_MEMBER*[group::role]*
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3]]
[-from_include_type=*object-type1[,object-type2,...]***]**
[-from_exclude_type=*object-type1[,object-type2,...]***]**
[-to_include_type=*object-type1[,object-type2,...]***]**
[-to_exclude_type=*object-type1[,object-type2,...]***]**
[-from_attach= target | reference | schedule_task]
[-from_relation=*relation-type***]**
[-from_include_related_type=*object-type1[,object-type2,...]* |
-from_exclude_related_type=*object-type1[,object-type2,...]***]**
[-clear] [-first_object_only]
[-bypass_condition_check]
[-project_scope=all | owning_project]
[-check_first_object_only=*true* | *false***]**
[-condition_name=*condition1***]**
[-condition_scope=all | any | none]

ARGUMENTS

-name

Specifies the keyword of the dynamic participant that you want to assign participants to. Accepts one of the following:

- **PROPOSED_REVIEWERS**

- **CHANGE_REVIEW_BOARD**

- **CHANGE_IMPLEMENTATION_BOARD**

> **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.

-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 **$TARGET_GROUP [**_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 **$PROCESS_GROUP**

  Owning group of the workflow process.

- **user:PROP::***property_name*

  Adds the user specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::***property_name*

  Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **$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.

- **$PROCESS_OWNER**

  Adds the workflow process owner to the signoff member list.

- **$TARGET_OWNER [***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[***group***::***role***]**

  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.

  If the **$PROJECT_MEMBER[***group***::***role***]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

  You can specify a sub-group with the syntax *group***++***sub-group***::***role*.

- **$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.

-from_include_type=_object-type1[,object-type2,...]_

(Optional) Specifies the object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_exclude_type=_object-type1[,object-type2,...]_

(Optional) Specifies the object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-to_include_type=_object-type1[,object-type2,...]_

(Optional) Specifies the object types to be used while assigning participants on the target attachment. They must be valid object types.

The **-to_include_type** and **-to_exclude_type** arguments are mutually exclusive. If you use one, you cannot use the other.

-to_exclude_type=_object-type1[,object-type2,...]_

(Optional) Specifies the object types to be excluded while assigning participants on the target attachment. They must be valid object types.

The **-to_include_type** and **-to_exclude_type** arguments are mutually exclusive. If you use one, you cannot use the other.

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**) to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). If this argument is not specified, the default is **target**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_relation

(Optional) Specifies the relation of the objects to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). It must be a valid relation.

*   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**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_include_related_type=*object-type1[,object-type2]*

(Optional) Specifies the related object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

Use this argument when a property is designated

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

-from_exclude_related_type=*object-type1[,object-type2]*

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

-clear

(Optional) Removes all previously assigned participants before assigning new participants. If this argument is not specified, new participants are appended to existing participants list.

-first_object_only

(Optional) Sets the participants on the first target attachment matching the **-to_include_type** and **-to_exclude_type** arguments. If this argument is not specified, the participants are set on all target attachments matching the **-to_include_type** and **-to_exclude_type** arguments.

-bypass_condition_check

(Optional) Bypasses the Business Modeler IDE condition check before assigning participants. If this argument is not specified, the Business Modeler IDE conditions are enforced before assigning participants.

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the identified objects from which to assign participants. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects. Values are the following:

all	All objects should meet the condition. This is the default behavior if this argument is not supplied with the **–condition_name** argument.
any	Any object should meet the condition.
none	No object should meet the condition.

PLACEMENT

Place on the **Start** action.

RESTRICTIONS

Can only be used to assign dynamic participants that resolve to a multiple users. For example:

PROPOSED_REVIEWERS or **CHANGE_REVIEW_BOARD**

EXAMPLES

• Assigns the users **Smith** and **David** as the **PROPOSED_REVIEWERS** participant for all target objects in the workflow process.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**user:Smith,David**

• Reads the **owning_user** and **last_mod_user** properties from the target and assigns the user as the **PROPOSED_REVIEWERS** participant for the first target object only.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**user:PROP::owning_user,user:PROP::last_mod_user**
-first_object_only	

- Reads the **owning_user** and **last_mod_user** properties from the **Document Revision** type target and assigns the user as the **PROPOSED_REVIEWERS** participant.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**user:PROP::owning_user,user:PROP::last_ mod_user**
-from_include_type	**DocumentRevision**

- Traverses the **References** relation from the **Part Revision** types of the targets to get the **Document Revision** objects. It then reads the **owning_user** and **last_mod_user** properties from the **Document Revision** and assigns the user as the **PROPOSED_REVIEWERS** participant for all target objects.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**user:PROP::owning_user,user:PROP::last_ mod_user**
-from_include_type	**Part Revision**
-from_relation	**IMAN_reference**
-from_include_related_type	**DocumentRevision**

- This example assigns all members of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the dynamic participant.

Argument	Values
-name	**PROPOSED_RESPONSIBLE_PARTY**
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

- This example assigns all members of the **Engineering** group and **Designer** role of the owning project team associated with the first target found by the system to the dynamic participant.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

Argument	Values
-project_scope	**owning_project**
-check_first_object_only	

- This example assigns all members of the **Engineering** group and **Designer** role of all project teams associated with the first target found by the system to the dynamic participant.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-project_scope	**all**
-check_first_object_only	**true**

- This example assigns all members of the **Engineering** group and **Designer** role of the first project team associated with each target found by the system to the dynamic participant.

Argument	Values
-name	**PROPOSED_REVIEWERS**
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-check_first_object_only	**false**

EPM-assign-responsible-party-dynamic-participant

DESCRIPTION

Assigns the specified user or resource pool as the dynamic participant to the target attachment.

> **Note**
>
> Participants can be assigned to **Item Revision** subtypes only. **Non-Revision Items** are removed from processing and, if no **Targets** are left, may result in this warning: **No attachment of the specified type can be found.**

If the BMIDE constant **Fnd0ParticipantEligibility** is defined for the dynamic participant, the handler gets the corresponding group member which matches the group and role criteria defined in the BMIDE constant. If the user identified through the **–assignee** argument does not have the correct group and role membership, the handler reports an error and does not assign the user to the dynamic participant.

If the value is specified as **$PROJECT_AUTHOR** or **$PROJECT_MEMBER[**group**::**role**]**, the relevant first project member is selected.

> **Note**
>
> Use the **WRKFLW_display_participants** preference to specify which dynamic-participant types are displayed when assigning dynamic participants for an object.

SYNTAX

EPM-assign-responsible-party-dynamic-participant
-name= {PROPOSED_RESPONSIBLE_PARTY
| ANALYST
| CHANGE_SPECIALIST1
| CHANGE_SPECIALIST2
| CHANGE_SPECIALIST3}
[-assignee= [user:user **| person:**person
| resourcepool:group**::**role
| user:PROP::property_name
| resourcepool:PROP::property_name
| $PROPOSED_RESPONSIBLE_PARTY | $USER
| $PROCESS_OWNER | $TARGET_OWNER [type**]**
| $PROJECT_ADMINISTRATOR
| $PROJECT_TEAM_ADMINISTRATOR
| $PROJECT_AUTHOR | $PROJECT_MEMBER[group**::**role]
| $REQUESTOR | $ANALYST
| $CHANGE_SPECIALIST1
| $CHANGE_SPECIALIST2
| $CHANGE_SPECIALIST3]]
[-from_include_type=object-type1[,object-type2,...]]

-from_exclude_type=*object-type1[,object-type2,...]***]**
[-to_include_type=*object-type1[,object-type2,...]***]**
-to_exclude_type=*object-type1[,object-type2,...]***]**
[-from_attach= target | reference | schedule_task]
[-from_relation=*relation-type***]**
[-from_include_related_type=*object-type1[,object-type2,...]* **] |**
-from_exclude_related_type=*object-type1[,object-type2,...]***]**
[-first_object_only]
[-bypass_condition_check]
[-project_scope=all | owning_project]
[-check_first_object_only=*true* | *false***]**
[-condition_name=*condition1***]**
[-condition_scope=all | any | none]

ARGUMENTS

-name

Specifies the keyword of the dynamic participant that you want to assign participants. Accepts one of the following:

- **PROPOSED_RESPONSIBLE_PARTY**

- **ANALYST**

- **CHANGE_SPECIALIST1**

- **CHANGE_SPECIALIST2**

- **CHANGE_SPECIALIST3**

> **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.

-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 **$TARGET_GROUP [***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 **$PROCESS_GROUP**

    Owning group of the workflow process.

- **user:PROP::***property_name*

  Adds the user specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::***property_name*

  Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **$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.

- **$PROCESS_OWNER**

  Adds the workflow process owner to the signoff member list.

- **$TARGET_OWNER [***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[***group***::***role***]**

  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.

  If the **$PROJECT_MEMBER[***group***::***role***]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

  You can specify a sub-group with the syntax *group***++***sub-group***::***role*.

- **$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.

-from_include_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_exclude_type=object-type1[,object-type2,...]

(Optional) Specifies the object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-to_include_type=object-type1[,object-type2,...]

(Optional) Specifies the object types to be used while assigning participants on the target attachment. They must be valid object types.

The **-to_include_type** and **-to_exclude_type** arguments are mutually exclusive. If you use one, you cannot use the other.

-to_exclude_type=object-type1[,object-type2,...]

(Optional) Specifies the object types to be excluded while assigning participants on the target attachment. They must be valid object types.

The **-to_include_type** and **-to_exclude_type** arguments are mutually exclusive. If you use one, you cannot use the other.

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**) to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). If this argument is not specified, the default is **target**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_relation

(Optional) Specifies the relation of the objects to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). It must be a valid relation.

- 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**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_include_related_type=_object-type1[,object-type2]_

(Optional) Specifies the related object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

-from_exclude_related_type=_object-type1[,object-type2]_

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**_property_name_). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

-first_object_only

(Optional) Sets the participants on the first target attachment matching the **-to_include_type** and **-to_exclude_type** arguments. If this argument is not specified, the participants are set on all target attachments matching the **-to_include_type** and **-to_exclude_type** arguments.

-bypass_condition_check

(Optional) Bypasses the Business Modeler IDE condition check before assigning participants. If this argument is not specified, the Business Modeler IDE conditions are checked before assigning participants.

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the identified objects from which to assign participants. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects. Values are the following:

all   All objects should meet the condition. This is the default behavior
                                 if this argument is not supplied with the **–condition_name**
                                 argument.

any   Any object should meet the condition.

none   No object should meet the condition.

PLACEMENT

Place on the **Start** action.

RESTRICTIONS

Can only be used to assign dynamic participants that resolve to a single user. For
example:

PROPOSED_RESPONSIBLE_PARTY or **ANALYST**

EXAMPLES

- Assigns the user **Smith** as the **PROPOSED_RESPONSIBLE_PARTY** participant
  for all target objects in the workflow process.

Argument	Values
-name	**PROPOSED_RESPONSIBLE_PARTY**
-assignee	**user:Smith**

- Reads the **owning_user** property from the target and assigns the user as the
  PROPOSED_RESPONSIBLE_PARTY participant for the first target object only.

Argument	Values
-name	**PROPOSED_RESPONSIBLE_PARTY**
-assignee	**user:PROP::owning_user**
-first_object_only	

- Reads the **owning_user** property from the **Document Revision** type target and
  assigns the user as the **PROPOSED_RESPONSIBLE_PARTY** participant.

Argument	Values
-name	**PROPOSED_RESPONSIBLE_PARTY**
-assignee	**user:PROP::owning_user**
-from_include_type	**DocumentRevision**

- Traverses the **References** relation from the **Part Revision** types of the targets
  to get the **Document Revision** objects. It then reads the **owning_user**
  property from the **Document Revision** and assigns the user as the
  PROPOSED_RESPONSIBLE_PARTY participant for all target objects.

Argument	Values
-name	**PROPOSED_RESPONSIBLE_PARTY**
-assignee	**user:PROP::owning_user**

Argument	Values
-from_include_type	**Part Revision**
-from_relation	**IMAN_reference**
-from_include_related_type	**DocumentRevision**

- This example assigns the first member of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the dynamic participant.

Argument	Values
-name	**PROPOSED_RESPONSIBLE_PARTY**
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

EPM-apply-digital-signature

DESCRIPTION

Applies the digital signature of the logged-on user to the target objects and, optionally, the schedule task.

SYNTAX

EPM-apply-digital-signature [-include_schedule_task]

ARGUMENTS

-include_schedule_task

(Optional) Applies the digital signature to the schedule task and all target objects of the workflow. If this argument is not provided, the digital signature is applied only on the target objects of the workflow.

PLACEMENT

Place either on the **Perform** action of the **perform-signoffs** task or the **Complete** action of the following tasks:

- **Do** task

- **Condition** task

- **select-signoff-team** task

On a **Route** task, place on the **Complete** action of the **select-signoff-team** subtask of the **Review** task.

RESTRICTIONS

Do not place a workflow handler that modifies digital signature key property values before this handler on the same action on the same workflow task. Modifying digital signature key properties after applying a digital signature voids the signature.

EPM-adhoc-signoffs

DESCRIPTION

> **Note**
>
> The Teamcenter rich client and thin client display the **Ad Hoc done** checkbox, but the Active Workspace client does not.

Determines the behavior of the **Ad-hoc done** check box in 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.

If the **–required** argument is specified; the signoffs will be added as required signoffs which cannot be removed or marked as optional by users. 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 run 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 **$TARGET_GROUP** keyword argument with this handler and the handler is run 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 run. When you use this method, keyword arguments always resolve to the workflow process initiator.

Alternatively, if the **-ce** argument is used, the handler is not run when the workflow process is initiated. The handler is run instead when the **select-signoff-team** task starts.

If the –condition_name argument is specified; the handler will add the reviewers or set auto complete only if the condition is met. However, it will not reset the auto-complete flag if it is already set on the select-signoff-team task.

SYNTAX

EPM-adhoc-signoffs
[-**auto_complete**]
[-**assignee= {user:***user* | **person:***person* | **addresslist:***list*
| **resourcepool:***group***::***role*
| **allmembers:***group***::***role*
| **user:PROP::***property_name*
| **resourcepool:PROP::***property_name*
| **allmembers:PROP::***property_name*
| **$PROPOSED_RESPONSIBLE_PARTY | $PROPOSED_REVIEWERS | $USER**
| **$PROCESS_OWNER | $TARGET_OWNER** *[type]*
| **$PROJECT_ADMINISTRATOR**
| **$PROJECT_TEAM_ADMINISTRATOR**
| **$PROJECT_AUTHOR | $PROJECT_MEMBER***[group***::***role]*
| **$REQUESTOR | $ANALYST**
| **$CHANGE_SPECIALIST1 | $CHANGE_SPECIALIST2 | $CHANGE_SPECIALIST3**
| **$CHANGE_REVIEW_BOARD | $CHANGE_IMPLEMENTATION_BOARD**}]
[-**from_include_type=***object-type1[,object-type2,...]*]
[-**from_exclude_type=***object-type1[,object-type2,...]*]
[-**from_attach=target | reference | schedule_task**]
[-**from_relation=***relation-type*]
[-**from_include_related_type=***object-type1[,object-type2,...]* |
-**from_exclude_related_type=***object-type1[,object-type2,...]*]
[-**quorum=***quorum-value*]
[-**ce**] [-**clear_signoffs**]
[-**target_task=***multilevel-task-path*]
[-**required**]
[-**project_scope=all | owning_project**]
[-**check_first_object_only=***true* | *false*]
[-**condition_name=***condition1*]
[-**condition_scope=all | any | none**]

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 **EPM-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.

Separate multiple assignees with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

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.

- **user:PROP::***property_name*

  Adds the user specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **resourcepool:PROP::***property_name*

  Adds the resource pool specified by the property name to the signoff member list for the task to which it is attached.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **allmembers:PROP::***property_name*

  Adds all members of a group/role combination that is specified by the property name to the signoff member list.

  If the property is a multi-value property, only the first value is used when only a single user is assigned in the workflow. When more than one user is assigned, all property values are used.

- **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 *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 **$TARGET_GROUP[***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 **$PROCESS_GROUP**

     Owning group of the workflow process.

- **allmembers:***group***::***role*

  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 **$TARGET_GROUP[***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 **$PROCESS_GROUP**

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.

- **$USER**

Adds the current user to the signoff member list.

- **$PROCESS_OWNER**

Adds the workflow process owner to the signoff member list.

- **$TARGET_OWNER [***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[***group***::***role***]**

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.

If the **$PROJECT_MEMBER[***group***::***role***]** argument is specified, only the project members of the qualifying projects which belong to the specified group and role are selected for assignment. If the group and role are not specified, all the project members from qualifying projects are selected.

You can specify a sub-group with the syntax *group***++***sub-group***::***role*.

- **$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.

-from_include_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_exclude_type=*object-type1[,object-type2,...]*

(Optional) Specifies the object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_attach= target | reference | schedule_task

(Optional) Specifies which type of attachment (**target**, **reference**, or **schedule_task**) to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). If this argument is not specified, the default is **target**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_relation

(Optional) Specifies the relation of the objects to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::***property_name*). It must be a valid relation.

- 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**.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**).

-from_include_related_type=object-type1[,object-type2]

(Optional) Specifies the related object types to be used to get the property value from when a property is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

This argument should not be used with the **-from_exclude_related_type** argument.

-from_exclude_related_type=object-type1[,object-type2]

(Optional) Specifies related object types to be excluded when getting the property value when it is specified in the **-assignee** argument (for example, **-assignee=user:PROP::**property_name). They must be valid object types.

You can use this argument only when you get the assignee from a property on an object (**user:PROP::** or **resourcepool:PROP::)**) and you use the **-from_relation** argument.

This argument should not be used with the **-from_include_related_type** argument.

-quorum

(Optional.) Determines the approval 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

(Optional.) Disables the default behavior of running this handler when the workflow process is initiated. Instead, the handler is run when the **select-signoff-team** task is initiated in the workflow.

If **-ce** is specified, the **select-signoff-team** task does not auto-complete even if a process assignment list is assigned during process initiation. For the **select-signoff-team** task to auto-complete, you must also use the **-auto_complete** handler argument.

-clear_signoffs

(Optional.) If specified, all existing signoffs are removed from the **select-signoff-team** subtask before the new signoffs are added. If you specify this argument, you must also use the **-ce** argument before it.

-target_task

(Optional) Specifies the multilevel task path to which the reviewers are added. The path is from the root task to the **select-signoff-team** subtask with the path levels separated with colons (:). For example: **Change Request Review:QA Review:select-signoff-team**

-required

(Optional) If specified, all signoffs added through this handler instance are marked as mandatory.

-project_scope

(Optional) Specifies which projects are used to resolve project-based assignments. The **all** value specifies all projects in the list of projects. The **owning_project** value specifies the owning project only.

If this argument is not specified, the default value is the first project in the project list.

-check_first_object_only

(Optional) The **true** value specifies that only the first object is checked. If the value is **false**, all objects are checked. If this argument is not specified, or if it is specified without a value, only the first object is checked.

If the **–include_type**, **-exclude_type**, **-include_related_type**, or **–exclude_related_type** arguments are specified, they determine the types of objects that are checked.

-condition_name

(Optional) The name of the condition to evaluate against the objects identified for assigning reviewers from. The condition signature should accept a **WorkspaceObject & UserSession**. The handler assigns the reviewers only if the condition results are successful, based on the **–condition_scope** argument.

-condition_scope

(Optional) The criteria for evaluating condition results against workflow objects.

all	All objects should meet the condition. This is the default behavior if this argument is not supplied with the **–condition_name** argument.
any	Any object should meet the condition.
none	No object should meet the condition.

PLACEMENT

Place on the **Start** action of a **select-signoff-team** subtask.

This handler runs at workflow process initiation if the **-ce** argument is not specified. If **-ce** is specified, the handler runs 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 run when the workflow process is initiated, not when tasks are approved.

If the **-ce** argument is not specified, all instances of this handler are run when the workflow process is initiated and in this case the keywords refer to the initiating user.

- 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.

Argument	Values
-ce	

- 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 run at the time of workflow process initiation.

Argument	Values
-assignee	**user:Smith, resourcepool:$GROUP::$ROLE, addresslist:List1, $PROPOSED_REVIEWERS**
-quorum	**80%**

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 run when the task action on which it is attached is run. The quorum is set to **80%** which means that of all the signoff members, 80% need to approve for the task to move ahead.

Argument	Values
-assignee	**user:Smith, resourcepool:$GROUP::$ROLE, addresslist:List1, $PROPOSED_REVIEWERS**
-quorum	**80%**
-ce	

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

Argument	Values
-assignee	**user:Smith**

- This example assigns the owning user ID of the first **UGMASTER** target found by the system to the signoff team.

Argument	Values
-assignee	**user:$TARGET_OWNER[UGMASTER]**

- This example assigns the project team administrator of the project team associated with the first target found by the system to the signoff team.

Argument	Values
-assignee	**user:$PROJECT_TEAM_ADMINISTRATOR**

- This example assigns all members of the **jhList** address list to the signoff team.

Argument	Values
-assignee	**addresslist:jhList**

- This example assigns the **manufacturing** resource pool (any role within the manufacturing group) to the signoff team.

Argument	Values
-assignee	**resourcepool:manufacturing::**

- This example assigns the **$PROCESS_GROUP** resource pool (any role within the **xyz** group, where **xyz** is the owning group of the workflow process) to the signoff team.

Argument	Values
-assignee	**resourcepool:$PROCESS_GROUP::**

- This example assigns the **$TARGET_GROUP** resource pool (any roles within the **abc** group, where **abc** is the group of the first item revision target) to the signoff team.

Argument	Values
-assignee	**resourcepool:$TARGET_GROUP::**

- This example assigns the engineer role within the manufacturing group resource pool to the signoff team.

Argument	Values
-assignee	**resourcepool:manufacturing::engineer**

- This example assigns the current logged on role within the current logged on group resource pool to the signoff team.

Argument	Values
-assignee	**resourcepool:$GROUP::$ROLE**

- This example assigns the engineer role within any group resource pool to the signoff team.

Argument	Values
-assignee	**resourcepool:::engineer**

- 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 run when the task action on which it is attached is run.

Argument	Values
-assignee	**user:smith, $PROPOSED_REVIEWERS**
-quorum	**51%**
-ce	

- 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 run when the task action on which it is attached is run.

Argument	Values
-assignee	**allmembers:Engineering::Engineer**
-quorum	**80%**
-ce	

- 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 run when the task action on which it is attached is run.

Argument	Values
-assignee	**resourcepool:Engineering::Engineer, addresslist:list1**
-quorum	**5**
-ce	

- 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 run when the task action on which it is attached is run.

Argument	Values
-assignee	**user:smith,resourcepool:$GROUP::$ROLE, addressList:list1,$PROPOSED_REVIEWERS**
-ce	

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 run when the task action on which it is attached is run.

Argument	Values
-assignee	**user:smith,resourcepool:$GROUP::$ROLE, $PROJECT_MEMBER,$PROPOSED_REVIEWERS**
-ce	

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 run when the task action on which it is attached is run.

Argument	Values
-assignee	**user:smith,resourcepool:$GROUP::$ROLE, $CHANGE_REVIEW_BOARD,$ANALYST**
-ce	

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 run when the task action on which it is attached is run. 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.

Argument	Values
-ce	
-clear_signoffs	
-assignee	**$PROPOSED_RESPONSIBLE_PARTY**
-auto_complete	

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 assigns all members of the **Engineering** group and **Designer** role of the first project team associated with the first target found by the system to the signoff team as optional signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**

•   This example assigns all members of the **Engineering** group and **Designer** role of the owning project team associated with the first target found by the system to the signoff team as required signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-project_scope	**owning_project**
-check_first_object_only	
-required	

•   This example assigns all members of the **Engineering** group and **Designer** role of all project teams associated with the first target found by the system to the signoff team as optional signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-project_scope	**all**
-check_first_object_only	**true**

- This example assigns all members of the **Engineering** group and **Designer** role of the first project team associated with each target found by the system to the signoff team as optional signoffs.

Argument	Values
-assignee	**$PROJECT_MEMBER[Engineering::Designer]**
-check_first_object_only	**false**

- This example places the handler on the **Start** action of the **select-signoff-team** subtask. The **-ce** argument ensures that the **$PROPOSED_REVIEWERS** variable is not set until the **select-signoff-team** subtask is initiated.   Without the **-ce** argument, the **$PROPOSED_REVIEWERS** variable is assigned the values of **$PROPOSED_REVIEWERS** that existed at process initiation.

> **Note**
>
> These dynamic variables can change value throughout a process, so care needs to be taken to ensure the desired functionality.

Argument	Values
-ce	
-assignee	**$PROPOSED_REVIEWERS**

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

Argument	Values
-type	**DPV_AIType**
-RevisionRule	**Latest Working**

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

Argument	Values
-type	**DPV_PlantAIType**
-RevisionRule	**Latest Working**

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

Argument	Values
-type	**DPV_AIType**
-RevisionRule	**Latest Working**

DOCMGT-update-document-property

DESCRIPTION

Update the datasets (for example, MSWordX dataset with docx extension) associated with the target item revisions with the latest attribute exchange data, if there are any from Teamcenter to file (docx file).

> **Note**
>
> • This handler requires Teamcenter Dispatcher for the update.
>
> • The **RenderMgtTranslator** service must be enabled.
>
> • Use the Business Modeler IDE to set up and deploy IRDC and dispatcher service configuration objects to the Teamcenter database.
>
> • Target item revisions must be valid and checked in.

The update is asynchronous. The workflow continues while the update begins and runs to completion.

> **Tip**
>
> You can use a **Do** task to wait for the update process to initiate the **Complete** action before the workflow continues. The update process sets the task state to **Completed** when the update is successful.

SYNTAX

DOCMGT-update-document-property

ARGUMENTS

None

PLACEMENT

Place on the **Start** action of a **Do** task.

> **Note**
>
> Whenever this handler is used, upon successful completion, an Active Workspace user gets notified: either the process initiator, the task responsible party, or the Dispatcher client user (**dcproxy**).
>
> When the workflow administrator sets up the workflow:
>
> • If there is only one **Do** task in the workflow to update document properties, the handler is placed on the **Start** action of the **Do** task and the workflow initiator gets a notification.
>
> • If there are several tasks in the workflow, including a **Do** task for updating document properties, and the handler is placed on the **Start**

action of the **Do** task, the notification goes to the user who completed the predecessor task.

- If a successor task invokes this same handler or the **DOCMGT-render-document-revision** handler, an administrator can add a predecessor **Do** task to ensure that the user who completes the predecessor task receives the notification. Otherwise, the Dispatcher client user receives the notification.

Caution

Do not place this handler on the **perform** action of the **perform-signoffs** task. Otherwise, this handler runs multiple times.

RESTRICTIONS

- Requires Dispatcher to update the dataset's files.

- Item revision with attached datasets like Microsoft WordX must be included as targets of the workflow process.

- Do not use this handler with a workflow that is running in the background.

DOCMGT-update-docprop-logicalobject

DESCRIPTION

Update the datasets (for example, MSWordX dataset with docx extension) associated with the target item revisions with the latest attribute exchange data, if there are any from Teamcenter to file (docx file).

> **Note**
>
> - The generic (logical object) attribute exchange currently supports **MSWordX** dataset only.
>
> - The **MSWordX** dataset must be for the generic attribute exchange to occur.
>
> - Target items revisions must be valid and checked in.
>
>   The attribute exchange process from this workflow handler bypasses the **Fnd0TriggerLOAttrExch** business object constant configuration.

The update is synchronous.

SYNTAX

DOCMGT-update-docprop-logicalobject

ARGUMENTS

None

PLACEMENT

Place on the **Start** action of a **Task**.

> **Caution**
>
> Do not place this handler on the **perform** action of the **perform-signoffs** task. Otherwise, this handler runs multiple times.

RESTRICTIONS

Item revision with attached datasets like Microsoft WordX must be included as targets of the workflow process.

DOCMGT-render-document-revision

DESCRIPTION

Translates datasets associated with target item revisions to derived visualization datasets, for example, **Full Text** datasets to PDF. Settings from the **Item Revision Definition Configuration (IRDC)** and **Dispatcher Service Configuration** determine the file formats of the input and output datasets.

> **Note**
>
> • This handler requires Teamcenter Dispatcher for the translation.
>
> • Target item revisions must be valid and checked in.

The translation is asynchronous; the workflow continues while translation begins and runs to completion. The translated files are stored in Teamcenter and may be related to the input datasets or item revisions.

> **Tip**
>
> You can use a **Do** task to wait for the update process to initiate the **Complete** action before the workflow continues. The update process sets the task state to **Completed** when the update is successful.

SYNTAX

DOCMGT-render-document-revision -existing_file=[replace | preserve]

ARGUMENTS

-existing_file

• **replace**

Replaces the existing (visualization) dataset with the new (translated) dataset.

• **preserve**

This is the default value.

o If the IRDC-specified output file is not yet associated with the item revision, translates the source dataset to a new output file.

o If the IRDC-specified output file is already associated with the item revision, translates the source dataset to a new output file without replacing the previous one.

PLACEMENT

Place on the **Start** action of a **Do** task.

> **Note**
>
> Whenever this handler is used, upon successful completion, an Active Workspace user gets notified: either the process initiator, the task responsible party, or the Dispatcher client user (**dcproxy**).
>
> When the workflow administrator sets up the workflow:
>
> •   If there is only one **Do** task in the workflow to render documents, the handler is placed on the **Start** action of the **Do** task and the workflow initiator gets a notification.
>
> •   If there are several tasks in the workflow, including a **Do** task for rendering documents, and the handler is placed on the **Start** action of the **Do** task, the notification goes to the user who completed the predecessor task.
>
> •   If a successor task invokes this same handler or the **DOCMGT-update-document-property** handler, an administrator can add a predecessor **Do** task to ensure that the user who completes the predecessor task receives the notification. Otherwise, the Dispatcher client user receives the notification.

You can use a **Do** task to wait for the translation process to initiate the **Complete** action before the workflow continues.

> **Caution**
>
> Do not place this handler on the **perform** action of the **perform-signoffs** task. Otherwise, this handler runs multiple times.

RESTRICTIONS

•   Requires Dispatcher for updating the dataset's file.

•   Item revisions with attached datasets such as Microsoft Word and Microsoft Excel must be included as targets of the workflow.

•   Do not use this handler with a workflow that is running in the background.

DOCMGTAPP-insert-pdf-cover-page

DESCRIPTION

Inserts a cover page to a PDF dataset attached to the target being sent in the workflow. The target can be an item, an item revision or its subtype, or the PDF dataset itself. The cover page is a PDF dataset that is related to the item revision by using the **Document Page Type** relation. Its **Page Type** relation property is set to **Cover Page**.

For this handler to insert a PDF cover page, the following conditions are required:

- The PDF dataset must be related to the item revision or its subtype. If it is related using the **Document Page Type** related, its **Page Type** relation property must be set to **Base Document**.

- The PDF cover page must be related to the item revision or its subtype.

SYNTAX

DOCMGTAPP-insert-pdf-cover-page [**-create_new_dataset=** *<true|false>* [**-new_dataset_suffix=** *<text>*]]

ARGUMENTS

-create_new_dataset

(Optional) If **true**, creates a new PDF dataset with the cover page inserted. If **false**, the original PDF file is modified.

-new_dataset_suffix

If **-create_new_dataset** argument is specified as true, you can enter any text string for the dataset suffix name.

PLACEMENT

Place on the **Start** action or the **Complete** action.

RESTRICTIONS

None

DOCMGTAPP-apply-pdf-control

DESCRIPTION

Applies a system stamp, watermark, logo (if attached), distribution statement text (if attached), workflow signoff table (if the target object is in a review task), and Teamcenter attributes when the logical object is related to the attached PDF dataset. A target object can be an item, an item revision or its subtype, or the PDF dataset itself.

The system stamp is an imprint comprising data such as a watermark and optional boilerplate text. In Business Modeler IDE, the data model administrator creates a system stamp configuration, associating the configuration with the XML command file that defines the watermark and text.

For this handler to apply the stamp and watermark, the following conditions are required:

- The PDF dataset must be related to the item revision or its subtype.

- The system stamp configuration must be enabled for the item revision or its subtype. The **Applies To** attribute of the system stamp configuration must be set to **PDF_Control**.

- The **PDF Control** access privilege must be granted.

SYNTAX

DOCMGTAPP-apply-pdf-control -user_stamp=*text string*

ARGUMENTS

-user_stamp

(Optional) Specifies any string for the text portion of the stamp.

PLACEMENT

Place on the **Start** action or the **Complete** action.

RESTRICTIONS

None

CSI-propagate-folder-contents

DESCRIPTION

Copies change objects in the change folders to the corresponding schedule task change folders.

SYNTAX

CSI-propagate-folder-contents -relation=_relation-name_ **[-no_condition_check= true|false][[-exclude_type=**_types-to-be-excluded_**] | [-include_type=**_types-to-be-included_**]][[-allowed_status=**_status-to-be-propagated_**] | [-disallowed_status=**_status-to-not-be-propagated_**]]**

ARGUMENTS

-relation

Propagates the change objects with the specified relation. The value can be one of the following:

- **CMHasProblemItem**

- **CMHasImpactedItem**

- **CMHasSolutionItem**

- **CMReferences**

To propagate objects that have different relations, add another instance of the handler to the task. For example, to propagate objects with the **CMHasProblemItem** and the **CMHasImpactedItem** relation, add the **CSI-propagate-folder-contents** handler with the **-relation=CMHasProblemItem** argument and value along with another **CSI-propagate-folder-contents** handler with the **-relation=CMHasImpactedItem** argument and value.

-bypass_condition_check

(Optional) Specifies whether to bypass condition checking. Valid values are **true** and **false**. If this argument is not specified, condition checking is used.

-exclude_type=_object-type_

(Optional) Does not propagate objects of the specified type.

The **-exclude_type** and **-include_type** 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_type=_object-type_

(Optional) Propagates objects of the specified type.

The **-exclude_type** and **-include_type** 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.

-allowed_status

(Optional) Propagates objects with the specified status.

The **-allowed_status** and **-disallowed_status** 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.

-disallowed_status

(Optional) Does not propagate objects with the specified status.

The **-allowed_status** and **-disallowed_status** 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.

PLACEMENT

Place on the **Start** task of the workflow process.

RESTRICTIONS

None.

EXAMPLES

- This example propagates change objects with the **CMHasProblemItem** relation.

Argument	Values
-relation	**CMHasProblemItem**

- This example propagates change objects with the **CMHasProblemItem** relation, but does not check conditions.

Argument	Values
-relation	**CMHasProblemItem**
-bypass_condition_check	**true**

- This example propagates change item revisions with the **CMHasProblemItem** relation and **Completed** status, but does not check conditions.

Argument	Values
-relation	**CMHasProblemItem**
-bypass_condition_check	**true**
-include_type	**ItemRevision**
-allowed_status	**Completed**

CPD-where-used-item-revision

DESCRIPTION

Finds all realized reuse design elements in the database for a specific revision of the source item assembly or installation assembly provided by the target in the process. If specified, the search scope is restricted to certain collaborative designs that are attached as references to the process.

All found reuse design elements are added to the references.

SYNTAX

CPD-where-used-item-revision

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action of any task.

RESTRICTIONS

None.

CPD-update-item-realization

DESCRIPTION

Updates the realization of all reuse design elements attached as references, using the source assembly item revision or installation assembly item revision provided by the target.

If the realization update fails, this handler reports the failed subordinates and corresponding error codes in the log file.

SYNTAX

CPD-update-item-realization

ARGUMENTS

None.

PLACEMENT

Place on the **Complete** action of any task.

RESTRICTIONS

None.

CPD-collect-related-items

DESCRIPTION

Collects objects related to design elements from a designated source pseudofolder in a change object. For example, this handler collects the source item revision, parent design elements (such as reuse) and their corresponding source item revisions, and adds them to designated target pseudofolder of the change object.

SYNTAX

CPD-collect-related-items
  –source_folder_relation_type=*relation-name*
   –processing_type=parent|assembly|default
   –destination_folder_relation_type=*relation-name*

ARGUMENTS

-source_folder_relation_type

Processes the design elements from the pseudofolder of the change object specified by the relation type. The value can be one of the following:

• **CMHasProblemItem**

• **CMHasImpactedItem**

• **CMHasSolutionItem**

-processing_type

Defines how the design elements from the source folder are navigated to collect the related objects. The following modes are supported:

• **parent**

  The parent design element corresponding to the input design element and its source object are retrieved and copied to the target pseudofolder of the change object.

• **assembly**

  Reuse design element for the input design element and the corresponding source object that are retrieved and copied to the target pseudofolder of the change object.

• **default**

  Reuse design element and parent design element for the input design element and their corresponding source objects that are retrieved and copied to the target pseudofolder of the change object.

-destination_folder_relation_type

The related objects collected for the objects in the source folder based on the processing type are copied to the pseudofolder of the change object. Processes the design elements from the pseudofolder of the change object specified by the relation type. The value can be one of the following:

• **CMHasProblemItem**

• **CMHasImpactedItem**

- **CMHasSolutionItem**

Requires no specific placement.

This handler is specific to design elements as the source objects.

- This example collects the reuse design element for the input design element in the **Problems** folder of an ECN, which would be a subordinate design element, and the source item revision for them. It then copies them to the **Impacted** folder of the ECN.

Argument	Values
-source_folder_relation_type	**CMHasProblemItem**
-processing_type	**assembly**
-destination_folder_relation_type	**CMHasImpactedItem**

- This example collects the immediate parent for the input design element in the **Problems** folder of an ECN and the source item revision. It then copies them to the same **Problems** folder of the ECN.

Argument	Values
-source_folder_relation_type	**CMHasProblemItem**
-processing_type	**parent**
-destination_folder_relation_type	**CMHasProblemItem**

CONTMGMTS1000D-setQAStatus

DESCRIPTION

Sets the **Quality Assurance Status** property of the data module and updates the XML of the data module to reflect the QA status.

SYNTAX

CONTMGMTS1000D-setQAStatus -verification=*status***-vertype=***type*

ARGUMENTS

-verification

Sets the QA verification status for the data module. You can use one of the following three values:

- **unverified**

- **firstVerification**

- **secondVerification**

-vertype

Sets the verification type of the QA status on the data module. You can use one of the following three values:

- **tabtop**

  The content was verified without the physical presence of the equipment or system, such as with design documentation.

- **onobject**

  The content was verified by practical demonstration of the procedure on the product.

- **ttandoo**

  Both table top and on object verifications have been performed.

This argument is ignored if the **-verification** argument is set to **unverified**:

PLACEMENT

Place on the **Start** or **Perform** action of a **Do** task.

RESTRICTIONS

This handler can be used only with **Civ0DM4Revision** objects.

CONTMGMTS1000D-increment

DESCRIPTION

Sets properties depending on whether the **Civ0DM4Revision** object in a workflow is rejected or released.

- If the **Civ0DM4Revision** object is *rejected*, the **inWork** number is incremented.

- If the **Civ0DM4Revision** object is *released*, the following properties are set:

  o The **issueNum** property is incremented.

  o The **inWork** number is reset to **00**.

  o The **issue_day**, **issue_month** and **issue_year** properties are set to the current date.

SYNTAX

CONTMGMTS1000D-increment {-incInWork | -incIssueNum}

ARGUMENTS

-incInWork

Increments only the **inWork** number. Use this argument for this handler on tasks after reviewers rejections.

-incIssueNum

Increments **issueNum**, resets **inWork** to **00**, and sets **issue_day**, **issue_month** and **issue_year** to the current date. Use this argument for this handler on a task after the document gets final approval.

PLACEMENT

Place on the **Start** or **Perform** action of a normal task.

RESTRICTIONS

This handler can be used only with **Civ0DM4Revision** objects.

CONFMGMT-cut-back-effectivity

DESCRIPTION

Reduces the effectivity range of problem item objects attached to a change object so it does not overlap with the combined effectivity range of the solution items. This facilitates the release of solution items to replace problem items for a given effectivity range.

> **Note**
>
> This handler should be used only for 4th Generation Design (4GD) objects.

For example, a cast component C is a solution item for a forged component F, a problem item with a unit effectivity of 1 through 10 in 4G Designer on 4GD data. C is assigned the same effectivity (unit 1 through 10) because it has the same purpose. To replace C with F with unit effectivity 3 through 10, a change notice is created that tracks F as a problem item and C as a solution item. The change notice is assigned an unit effectivity of 3 and up. The handler applies the change notice effectivity to the solution item and then reduces the effectivity range of the problem item. As a result, C has an effectivity range of 3 through 10 and F's effectivity is reduced to 1 through 2. For every unit in the range of 1 through 10, either C or F is effective. The effective ranges of C and F neither overlap nor do they have a gap.

The effectivity range of the change is determined either by the release status attachment of the workflow process or by the effectivity range on the change object using **EffectivityConfigurable** behavior.

If the process does not have a release status attachment, the release statuses of the change object are used. An error occurs if multiple release statuses with effectivity data are found and handler arguments are used that require the definition of the effectivity range of the change object. By default, the system uses the effectivity range of the release statuses, unless user provides the **useECNEffectivity** argument.

If the **useECNEffectivity** argument is used, the effectivity range of the change object is determined as the effectivity of the change object using **EffectivityConfigurable** behavior. An error is returned if the change object does not have **EffectivityConfigurable** behavior

EffectivityConfigurable objects with no effectivity data behave as if they had an effectivity condition **Unit=1 OR Unit!=1** (in other words, **TRUE** unless explicitly stated otherwise). For more information, see the **defaultSolveTypePreferenceName** argument.

The effectivity range to be subtracted from a problem item attachment is the combined effectivity range of all **EffectivityConfigurable** objects in the corresponding solution item set. You can use the **designatorProperty** argument to define corresponding sets of solution and problem items. Solution item sets that do not correspond to a problem item set do not affect problem item effectivity ranges. Problem item sets that do not correspond to a solution item will be effected out permanently. Solution items without **EffectivityConfigurable** behavior (for example, datasets) are skipped in the computation of the effectivity range to be subtracted.

The handler only modifies problem item objects exposing **EffectivityConfigurable** behavior, such as **Cpd0DesignElement**. These modifications are not subject to access control rules.

CONFMGMT-cut-back-effectivity
[-engineeringChangeTypeName = { ChangeNoticeRevision | *object-type-name***}]**
[-problemItemRelationshipName = { CMHasProblemItem | *relationship-type-name* **}]**
[-solutionItemRelationshipName = { CMHasSolutionItem | *relationship-type-name* **}]**
[-verifyEffectivity = { NoAction | Compare | Validate}]
[-solutionItemEffectivity = { NoAction | ApplyCMEffectivity | MergeCMEffectivity |
ResetToCMEffectivity }]
[-designatorProperty = { "" | *property-name***}]**
[-defaultSolveTypePreferenceName = { "" | *preference-name***}]**
[-dropEndItemQualification]
[-useECNEffectivity   ]

-engineeringChangeTypeName

Sets the type of the target object managing the change. Any object type name is valid as long as there is only one such target attachment and the object type supports the relationship types specified below. The default value is **ChangeNoticeRevision**.

-problemItemRelationshipName

Sets the type name of the relationship that associates objects to be replaced by the objects specified by the **-solutionItemRelationshipName** argument with the change object. The type name must be compatible with the above change object type. The default value is **CMHasProblemItem**, but Siemens PLM Software recommends you use **CMHasImpactedItem** as the relationship name.

-solutionItemRelationshipName

Sets the type name of the relationship that associates objects, which replace the objects specified by the **-problemItemRelationshipName** argument, with the change object. The type name must be compatible with the change object type. The default value is **CMHasSolutionItem**.

-verifyEffectivity

Specifies the action to take with respect to the effectivity range of the change object and its solution item attachments. The action skips solution items for which no **EffectivityConfigurable** effectivity is saved or which do not expose **EffectivityConfigurable** behavior. Possible values are:

- **NoAction**

  Takes no action. This is the default.

- **Compare**

  Displays a separate warning for every solution item whose effectivity range does not equal the effectivity range of the change object. An error is returned if no effectivity has been saved for the change object.

- **Validate**

Returns an error if any solution item's **EffectivityConfigurable** effectivity range does not equal the effectivity range of the change object. An error is returned if no effectivity has been saved for the change object.

-solutionItemEffectivity

Specifies the action to take for solution item effectivity. Possible values are:

• **NoAction**

Takes no action. This is the default.

• **ApplyCMEffectivity**

Reduces the **EffectivityConfigurable** effectivity range of each solution item to be within the range of the change object (in other words, combines both with a logical **AND**). An error is returned if no release status effectivity is saved for the change object. The result is identical to action **ResetToCMEffectivity** for solution items, for which no **EffectivityConfigurable** effectivity has been saved, or which do not expose **EffectivityConfigurable** behavior.

• **MergeCMEffectivity**

Sets the **EffectivityConfigurable** effectivity range of each solution item to equal the range of the change object for the common effectivity intent; the effectivity range of the solution item having other intents are kept unchanged.

o If effectivity ranges of the solution item and the change object do not have a common effectivity intent, then the solution item effectivity range is extended with the effectivity range of the change object.

o An error is returned if no effectivity range has been saved for the change object or the effectivity range on the solution item or the change object has multiple effectivity intents or intent families.

> **Note**
>
> This mode is supported only with the **–useECNEffectivity** parameter.

• **ResetToCMEffectivity**

Sets the **EffectivityConfigurable** effectivity range of each solution item to equal the range of the release status effectivity of the change object. An error is returned if no release status effectivity has been saved for the change object. The result is identical to action **NoAction** for solution items, which do not expose **EffectivityConfigurable** behavior.

-designatorProperty

Specifies the property to use to group change object attachments into sets for the purpose of replacing problems items with corresponding solution items. These sets are formed by virtue of having a common value for the same property (for example, a logical designator as stored on a partition membership in the preferred partition scheme). If a property is specified, the solution item attachments of the change object

are grouped into sets formed by the value for this property. If the property name is an empty string (the default) there is one set for all solution items that corresponds to one set for all problem items.

-defaultSolveTypePreferenceName

By default, **EffectivityConfigurable** objects without effectivity condition behave as if they had an effectivity condition **Unit=1 OR Unit!=1**, that is, equivalent to the Boolean constant **TRUE**. If the value for this argument is different from the empty string (default) it is expected to specify a preference having the same semantics as defined for **TC_Default_Solve_Type** in the **confmgmt** module, which can be used to define whether or not **EffectivityConfigurable** objects without effectivity condition pass effectivity filter criteria. If the given preference is not found in the scope specified by the **defaultSolveTypePreferenceScope** argument a default solve type of **529** is assumed, that is **solveMismatch|solveFalse|solveInvert** except where explicitly otherwise stated. The effectivity range that is assumed for **EffectivityConfigurable** objects without effectivity condition can be configured to be the following:

- **Unit=1 OR Unit!=1**

    Equivalent to the Boolean constant **TRUE**, if the solve type specifies that **EffectivityConfigurable** objects without effectivity condition pass effectivity filters.

- **Unit=1 AND Unit!=1**

    Equivalent to the Boolean constant **FALSE**, if the solve type specifies that **EffectivityConfigurable** objects without effectivity condition do not pass effectivity filters.

-dropEndItemQualification

(Optional) If provided and if an end item qualification is present, it is dropped and changed to an effectivity condition when it is copied from

- the ReleaseStatus attachment of the workflow process.

- the ReleaseStatus of the attached change notice if the workflow process does *not* have a ReleaseStatus attachment.

-useECNEffectivity

(Optional) If provided, the effectivity range of the change is determined by the effectivity range on the change notice object. The change notice object should carry the effectivity range using **EffectivityConfigurable** behavior. An error is returned if this argument is provided and the change notice object does not have **EffectivityConfigurable** behavior.

PLACEMENT

A typical placement is to precede the **add-status** action handler that attaches the release status to the change object, so that the release status is not attached to the change object if this handler errors out.

RESTRICTIONS

None.

- This example illustrates the use of the handler with a change object type that is available in the Teamcenter foundation template. It configures the handler to reduce the effectivity of the solution item attachments to not be effective beyond the effective range of the change.

Argument	Values
-engineeringChangeTypeName	**ItemRevision**
-problemItemRelationshipName	**IMAN_reference**
-solutionItemRelationshipName	**IMAN_manifestation**
-verifyEffectivity	**NoAction**
-solutionItemEffectivity	**ApplyCMEffectivity**
-designatorProperty	**object_desc**
-defaultSolveTypePreferenceName	**TC_Default_Solve_Type**
-dropEndItemQualification	None
-useECNEffectivity	None

CM-promote-change-notice

DESCRIPTION

Performs the following operations within a transaction and rolls back all changes if there is a failure:

1. Applies release status to **ChangeNoticeRevision** workflow target objects.

2. Receives any change space data objects from the **ChangeNoticeRevision** POM Space and promotes or shares them to public data objects.

   > **Note**
   >
   > If specific object types contained in the POM space require pre- or post-promote validation, this can be accomplished by overriding the following methods on the respective types:
   >
   > - **fnd0ValidateBOTypePrePromote**
   >
   > - **fnd0ValidateBOTypePostPromote**

3. Applies release status to all of the solution items of the **ChangeNoticeRevision** target objects and any other targets not addressed in step 1.

   > **Note**
   >
   > If specific object types require pre-release validation, this can be accomplished by overriding the following method on the respective type:
   >
   > - **fnd0ValidateBOTypeForRelease**

   > **Note**
   >
   > The arguments and their effect on the behavior are all related to how the release status is applied to the target objects and **ChangeNoticeRevision** solution objects. The handler arguments are a copy of the arguments and processing behavior of the **EPM-set-status** handler.

SYNTAX

CM-promote-change-notice -action=append | replace | rename | delete [-status=*old_name***,]**
[-new_status=*new_name***]**
[-retain_release_date] [-set_effectivity] [-status_not_shared] [-promote=share]

ARGUMENTS

-action

  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.

If the target object has an existing status, the status object is renamed from **old_name** to **new_name**.

delete

Deletes the status **status_name** specified by the **-status** argument from the target object.

- If the **delete** argument is not used in combination with the -status 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.

-status

Used with the **-action** argument to define the status.

- If the action is **append** or **replace** and the status by the name given is not present on the root task, it will create a new status with this name and attach it to the root task.

- If the action is **delete**, it deletes the status objects from the target object but does not delete it from the root task.

- If the action is **rename**, it renames the status objects to the new value specified in **-new_status**.

The value provided should be the name of a status type already defined in the Business Modeler IDE, not the display name.

-new_status

Specifies the new name for the status object.

- The name provided should be the name of a status type already defined in the Business Modeler IDE, not the display name.

- This argument is only used in case of **rename** option for **–action** argument.

- If the status type is not already defined, a status object is not based on a status type, which means that effectivity and configuration may not work against it.

-retain_release_date

Retains the original release date on the target object if it had previously been released.

> **Note**
>
> This option is not valid when **-action=replace** is used.

-set_effectivity

When used, the system creates the open-ended date effectivity with release date as start date.

-status_not_shared

The default behavior is to share a single release status object reference for all target objects. When this argument is present, it changes that behavior and an individual copy of the release status object is added to each target object.

-promote

share

Specifies that the change space data objects from the **ChangeNoticeRevision** POM Space will be shared to public.

Any value other than **share** promotes the change space data objects from the **ChangeNoticeRevision** POM Space to public.

PLACEMENT

Place on any action. Typically attached to the **Complete** action.

RESTRICTIONS

If no argument is supplied or if an argument other than the one specified is supplied to the handler, the default behavior is to treat it as an action **append** argument.

If **replace** is used and there is more than one status object attached to the root task, the status on the target objects is replaced by the latest status on the root task.

EXAMPLES

• This example adds the status object of the root task to the target object.

Argument	Values
-action	**append**

• This example creates a new status with this name and attaches to the root task if status by the name given is not present on the root task already.

Argument	Values
-action	**append**
-status	**released**

• This example adds the status object of the root task to the target object and retains the original released date of the target object.

Argument	Values
-action	**append**
-retain_release_date	

- This example replaces all existing status objects with the status object of the root task.

Argument	Values
-action	**replace**

- 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.

Argument	Values
-action	**replace**
-set_effectivity	

- This example renames all the status objects named **pre-released** to the name of the new status object, **released**.

Argument	Values
-action	**rename**
-status	**pre-released**
-new_status	**released**

- This example deletes all status objects from the target object but does not delete it from the root task.

Argument	Values
-action	**delete**

- This example deletes a **released** status from the target object but does not delete it from the root task.

Argument	Values
-action	**delete**
-status	**released**

- This example takes the release status attached to root task and creates an individual copy of the release status object for each target object.

Argument	Values
-action	**append**
-status_not_shared	

- This example creates a new status with name **released** and attaches it to the root task if status by the name given is not present on the root task already. Also it creates an individual copy of the release status object for each target object.

Argument	Values
-action	**append**
-status_not_shared	
-status	**released**

- This example shares the change space contents to public and attaches **shared** status to the root task. Each time the **shared** operation is performed, the **shared** status is replaced and a copy of the release status object for each target object is created.

Argument	Values
-action	**replace**
-status_not_shared	
-status	**Cm0TC Shared**
-promote	**share**

CM-inactivate-edit-context

DESCRIPTION

Deactivates the change space associated with the change notice revision of the target.

SYNTAX

CM-inactivate-edit-context

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

CM-baseline-solution-item-revisions-on-change-notice

DESCRIPTION

Performs a smart baseline on the assemblies of any item revisions listed as **Solution Items** on target **ChangeNoticeRevisions**.

SYNTAX

CM-baseline-solution-item-revisions-on-change-notice
[-baseline_rev_rule=<revision rule name>]
[-baseline_process=<workflow process name>]

ARGUMENTS

-baseline_rev_rule

Defines the name of the revision rule used to configure the item revision's assembly structure for baselining.

Will use the default **Structure Manager** revision rule if omitted.

-baseline_process

Defines the name of the workflow process used to release the baseline revisions.

Will use the default baseline process **TC Default Baseline Process** if omitted.

PLACEMENT

Place on any action. Typically attached to the **Complete** action.

RESTRICTIONS

None.

CFG0-attach-rule-variability

Attaches variant option values and families that are referenced by a constraint rule. Such constraint rules may be located in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revisions of the values, families, and family groups.

> **Note**
>
> A configurator constraint rule references the option family if the family has free-form values. Otherwise, it references the option value directly.

SYNTAX

CFG0-attach-rule-variability
[attachment = {target | reference }]
[-configuration = {Latest Working | Latest Released }]
[-attachConfiguratorContext = {false | true }]
[-debug = { false | true }]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

* **target**

  Variant option families and values are attached as target objects. This is the default value.

* **reference**

  Variant option families and values are attached as reference objects.

  > **Note**
  >
  > If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revisions. Possible values are:

* **Latest Working**

  The most recently created revision that has no release status is attached. This is the default value.

* **Latest Released**

The most recently released revision is attached.

-attachConfiguratorContext

Specifies whether **Configurator Context** items that are referenced by the constraint rules in this workflow process should be attached as **reference** attachments.

> **Note**
>
> The **Configurator Context** items are always added as **reference** attachments. This behavior is not affected by the **-attachment** parameter value.

Options are:

- **true**

  Configurator Context items that are referenced by the constraint rules in this workflow process are attached as **reference** attachments. This is the default value.

- **false**

  No additional **Configurator Context** items are attached.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-rule-variability** action handler between a **CFG0-attach-constraint-rules** handler and a **CFG0-attach-families** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Working** revisions of variant option values and families that are used in the constraint rules in this workflow process as target attachments so that they are processed along with the constraint rules that are already attached to the workflow. The list of **Configurator Context** items to which the constraint rules apply are added as a reference attachments to this workflow.

Argument	Values
-attachment	**target**
-configuration	**Latest Working**

Argument	Values
-attachConfiguratorContext	**true**

CFG0-attach-familygroups

DESCRIPTION

Attaches to the workflow process variant option family groups that reference variant option families in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revisions of the variant option families.

SYNTAX

CFG0-attach-familygroups
[-attachment = {target | reference}]
[-configuration = {Latest Working | Latest Released}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

• **target**

Variant option family groups are attached as target objects. This is the default value.

• **reference**

Variant option family groups are attached as reference objects.

> **Note**
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revisions. Possible values are:

• **Latest Working**

The most recently created revision that doesn't have any release status is attached. This is the default value.

• **Latest Released**

The most recently released revision is attached.

-debug

Whether or not to log status information to the syslog file. Possible values are:

• **false**

No status information is written to the syslog file. This is the default value.

- **true**

   Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-familygroups** action handler between a **CFG0-attach-families** handler and a **CFG0-attach-allocations** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Released** revisions of variant option family groups for the variant option families in this workflow process as reference attachments so that they are processed along with the variant option families that are already attached to the workflow.

Argument	Values
-attachment	reference
-configuration	**Latest Released**

CFG0-attach-families

DESCRIPTION

Attaches to the workflow process variant option families that are referenced by variant option values in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revisions of the variant option families.

SYNTAX

CFG0-attach-families
[-attachment = {target | reference}]
[-configuration = { Latest Working | Latest Released}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

- **target**

  Variant option families are attached as target objects. This is the default value.

- **reference**

  Variant option families are attached as reference objects.

  > **Note**
  >
  > If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the Latest Working or Latest Released revisions. Possible values are:

- **Latest Working**

  The most recently created revision that doesn't have any release status is attached. This is the default value.

- **Latest Released**

  The most recently released revision is attached.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-families** action handler between a **CFG0-attach-rule-variability** handler and a **CFG0-attach-familygroups** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Released** revisions of variant option families for the variant option values in this workflow process as reference attachments so that they are processed along with the variant option values that are already attached to the workflow.

Argument	Values
-attachment	reference
-configuration	Latest Released

CFG0-attach-constraint-rules

DESCRIPTION

Attaches configurator constraint rules that reference a variant option value or variant option family. Such objects may be located in the target attachment or referenced attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revision of the constraint rules.

> **Note**
>
> A configurator constraint rule references the option family if the family has free-form values. Otherwise, it references the option value directly.

SYNTAX

CFG0-attach-constraint-rules
[-attachment = {target | reference}]
[-attachedConfiguratorContext = {false | true}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

- **target**

    Constraint rules are attached as target objects. This is the default value.

- **reference**

    Constraint rules are attached as reference objects.

> **Note**
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revisions. Possible values are:

- **Latest Working**

    The most recently created revision that does not have any release status is attached. This is the default value.

- **Latest Released**

    The most recently released revision is attached. Use this setting with care as there could be a large number of released constraint rules to attach.

-attachedConfiguratorContext

Specifies whether **Configurator Context** items that are attached to the workflow process should be used to filter constraint rules. Possible values are:

- **false**

  The configured revision of all constraint rules are attached, irrespective of their **Configurator Context** item scope. This is the default.

- **true**

  The configured revision of constraint rules are attached that reference a **Configurator Context** item that is attached to this workflow, for example, as a reference attachment.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases it is useful to add the **CFG0-attach-constraint-rules** action handler followed by a **CFG0-attach-rule-variability** action handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Working** revisions of constraint rules as target attachments so that they are processed along with the values and families that are already attached to the workflow. The list of constraint rules to attach is not filtered by **Configurator Context**.

Argument	Values
-attachment	**target**
-configuration	**Latest Working**
-attachedConfiguratorContext	**false**

CFG0-attach-allocations

DESCRIPTION

Attaches allocation objects that reference variant option values, families, or family groups. Such objects may be located in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the allocation's **Latest Working** or **Latest Released** revision.

SYNTAX

CFG0-attach-allocations
[-attachment = {target | reference}]
[-configuration = {Latest Working | Latest Released}]
[-attachedConfiguratorContext = {false | true}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

•   **target**

Allocation revisions are attached as target objects. This is the default value.

•   **reference**

Allocation revisions are attached as reference objects.

> **Note**
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revision. Possible values are:

•   **Latest Working**

The most recently created revision with no release status is attached. This is the default value.

•   **Latest Released**

The most recently released revision is attached.

-attachedConfiguratorContext

Specifies whether relevant **Configurator Context** items for which allocation objects are to be added are attached to this workflow process. Possible values are:

•   **false**

Configured revisions for allocations to all **Configurator Context** items will be attached. This is the default value.

- **true**

  The configured allocation revisions to attach are filtered by the **Configurator Context** items attached to this workflow.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-allocations** handler below a **CFG0-attach-familygroups** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Working** revisions of variant option value, family, and family group allocations for variant option values, families, and family groups in this workflow process as target attachments so that they are processed along with the variability that is already attached to the workflow. The list of allocations to add is filtered by the **Configurator Context** items attached to this workflow.

Argument	Values
-attachment	**target**
-configuration	**Latest Working**
-attachedConfiguratorContext	**true**

CAE-simulation-process-launch-handler

DESCRIPTION

Launches the specified simulation tool.

SYNTAX

CAE-simulation-process-launch-handler -tool=*tool_ID*
-launch=*LOCAL_OR_SERVER_OR_REMOTE* **-nosync -continue -noref -param::**

ARGUMENTS

-tool

The ID of the simulation tool to launch.

> **Note**
>
> The simulation tool ID you specify here must match the simulation tool ID defined in the **Simulation Tool Configuration** dialog box in CAE Manager.

The **–tool** argument is mandatory and requires the simulation tool ID value. The rest of the arguments are optional and can be specified without any values.

Tool names and revisions are no longer supported. The tool is now launched with the latest released revision. If you have an existing action handler with a tool name and revision values, you must modify them and use only the tool ID value.

-launch

This argument is mandatory if you select the **Remote Launch** option 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 run 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.

> **Note**
>
> This argument is valid for local launch only. Remote launch is always run 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 run 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 *paramValue* 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 *paramValue* value can be an empty string, in which case the default value of the corresponding *paramName* is overridden with an empty value.

RESTRICTIONS

None.

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.

BC-perform-export

DESCRIPTION

Performs a Briefcase/PDX export using a workflow process.

SYNTAX

BC-perform-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=*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 email IDs of users to be notified when the transaction is completed. This argument is applicable only when **-usegs=True** and when the **-notify=True**.

Separate the email IDs with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

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.

Argument	Values
-site	**Supplier-site-1**
-optionset	**CustomOptionSet1**
-usegs	**False**

ASMAINTAINED-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 runs.

SYNTAX

ASMAINTAINED-release-asmaintained-structure -release status [-depth=*level* **| all] [-owned_by_initiator] [-owned_by_initiator_group] [-initiator_has_write_prev] {[-exclude_released] [-traverse_released_component]} [-exclude_types=***types* **] [-add_excluded_as_ref] [-include_missing]**

ARGUMENTS

-release status

Applies the specified release status to each of the physical parts.

-depth

Defines the depth to which the traversal should take place.

For example, specify **1** to traverse one level deep or **all** to traverse all levels.

If not specified, the handler traverses all levels.

-owned_by_initiator

Adds the components owned by the initiator as targets to the workflow process.

-owned_by_initiator_group

Adds all components owned by the initiator's group as targets to the workflow process.

-initiator_has_write_prev

Adds all component item revisions where the initiator has write access as targets to the workflow process.

-exclude_released

Excludes released components 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.

-traverse_released_component

Traverses the structure of the released component and adds the components as targets to the workflow process.

This argument can only be used in conjunction with the **-exclude_released** argument.

If the **-depth** argument is set to **1**, **-traverse_released_component** only traverses one level deep. If the **-depth** argument is set to **all**, **-traverse_released_component** traverses all levels of the subassembly.

-exclude_types

Defines the types to be excluded from being added as targets.

-add_excluded_as_ref

Adds components that are not included as targets to the workflow process as references.

-include_missing

Includes missing components as targets.

If this is not specified, an error is displayed for structures that contain missing components.

PLACEMENT

Requires no specific placement, but preferably after review/approval completion, if any.

RESTRICTIONS

None.

ASMAINTAINED-attach-physical-components

DESCRIPTION

Traverses the as-maintained structure and attaches as-built physical parts as targets to the workflow.

SYNTAX

ASMAINTAINED-attach-physical-components [-depth=*level* **| all]**
   [-owned_by_initiator] [-owned_by_initiator_group]
   [initiator_has_write_prev]
   {[-exclude_released] [-traverse_released_component]}
   [-exclude_types=types]
   [-add_excluded_as_ref][-include_missing]

ARGUMENTS

-depth

Defines the depth to which the traversal should take place.

- For example, specify **1** to traverse one level deep or **all** to traverse all levels.

- If not specified, the handler traverses all levels.

-owned_by_initiator

Adds the components owned by the initiator as targets to the workflow process.

-owned_by_initiator_group

Adds all components owned by the initiator's group as targets to the workflow process.

-initiator_has_write_prev

Adds all component item revisions where the initiator has write access as targets to the workflow process.

-exclude_released

Excludes released components 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.

-traverse_released_component

> **Note**
>
> This argument can only be used in conjunction with the **-exclude_released** argument.

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**, **-traverse_released_component** traverses all levels of the subassembly.

-exclude_types

Defines the types to be excluded from being added as targets.

-add_excluded_as_ref

Adds components that are not included as targets to the workflow process as references.

-include_missing

Includes missing components as targets.

If this is not specified, an error is displayed for structures that contain missing components.

PLACEMENT

Requires no specific placement, but preferably after review/approval completion, if any.

RESTRICTIONS

None.

ASBUILT-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 runs.

SYNTAX

ASBUILT-release-asbuilt-structure -release status [-depth=*level* | **all]** **[-owned_by_initiator] [-owned_by_initiator_group] [-initiator_has_write_prev]** **{[-exclude_released] [-traverse_released_component]} [-exclude_types=***types* **] [-add_excluded_as_ref] [-include_missing]**

ARGUMENTS

-release status

Applies the specified release status to each of the physical parts.

-depth

Defines the depth to which the traversal should take place.

For example, specify **1** to traverse one level deep or **all** to traverse all levels.

If not specified, the handler traverses all levels.

-owned_by_initiator

Adds the components owned by the initiator as targets to the workflow process.

-owned_by_initiator_group

Adds all components owned by the initiator's group as targets to the workflow process.

-initiator_has_write_prev

Adds all component item revisions where the initiator has write access as targets to the workflow process.

-exclude_released

Excludes released components 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.

-traverse_released_component

Traverses the structure of the released component and adds the components as targets to the workflow process.

This argument can only be used in conjunction with the **-exclude_released** argument.

If the **-depth** argument is set to **1**, **-traverse_released_component** only traverses one level deep. If the **-depth** argument is set to **all**, **-traverse_released_component** traverses all levels of the subassembly.

-exclude_types

Defines the types to be excluded from being added as targets.

-add_excluded_as_ref

Adds components that are not included as targets to the workflow process as references.

-include_missing

Includes missing components as targets.

If this is not specified, an error is displayed for structures that contain missing components.

PLACEMENT

Requires no specific placement, but preferably after review/approval completion, if any.

RESTRICTIONS

None.

ASBUILT-attach-physical-components

DESCRIPTION

Traverses the as-built structure and attaches as-built physical parts as targets to the workflow.

SYNTAX

ASBUILT-attach-physical-components [-depth=_level_ **| all]**
  [-owned_by_initiator] [-owned_by_initiator_group]
  [initiator_has_write_prev]
  {[-exclude_released] [-traverse_released_component]}
  [-exclude_types=types]
  [-add_excluded_as_ref][-include_missing]

ARGUMENTS

-depth

Defines the depth to which the traversal should take place.

- For example, specify **1** to traverse one level deep or **all** to traverse all levels.

- If not specified, the handler traverses all levels.

-owned_by_initiator

Adds the components owned by the initiator as targets to the workflow process.

-owned_by_initiator_group

Adds all components owned by the initiator's group as targets to the workflow process.

-initiator_has_write_prev

Adds all component item revisions where the initiator has write access as targets to the workflow process.

-exclude_released

Excludes released components 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.

-traverse_released_component

> **Note**
>
> This argument can only be used in conjunction with the **-exclude_released** argument.

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**, **-traverse_released_component** traverses all levels of the subassembly.

-exclude_types

Defines the types to be excluded from being added as targets.

-add_excluded_as_ref

Adds components that are not included as targets to the workflow process as references.

-include_missing

Includes missing components as targets.

If this is not specified, an error is displayed for structures that contain missing components.

PLACEMENT

Requires no specific placement, but preferably after review/approval completion, if any.

RESTRICTIONS

None.

AR-mark-archive

DESCRIPTION

> **Note**
>
> This handler is deprecated and will be obsolete in a future release. Do not add this handler to new workflow processes.

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

AR-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 **AR-mark-archive** handler with the following arguments is added to the **Complete** action of the root task.

Argument	Values
-exclude_related	**IMAN_manifestation::UGPART**
-status_to_keep	**R::3, X::2**

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.

All datasets except those:

*   With manifestation relation
*   Of type **UGPART**

of the item revisions, 10-11 and 15-17, are marked for archive.

AI-process-export

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*).

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.

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.

Rule handlers

ASBUILT-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

ASBUILT-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 service lifecycle management Service Manager or As-Built Manager is licensed and installed.

VAL-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.

SYNTAX

VAL-check-validation-result-with-rules
-rule_item_revision=*item-rev-id* **[-current_event=***event-value***]**
[-pass_item_revision_only] [-ref_log]

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.

-ref_log

If this argument is present and the validation fails, the validation results log is created, a warning message is displayed, and the log is attached.

If this argument is not present and the validation fails, the validation results log is created, a warning message is displayed, but the log is *not* attached.

If the validation passes, the validation results log is not created and no message is displayed.

PLACEMENT

Do not place this handler on the root task. Place it on the **Start** action of a subsequent task after a target is attached.

> **Note**
>
> If the handler is placed on the root task, and the handler fails to complete, the workflow process itself is not created. No log file under the **Newstuff** folder is created.

RESTRICTIONS

-rule_item_revision cannot be NULL.

VAL-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 all results are outdated or failed, the handler reports the corresponding error message and returns an **EPM_nogo** and the workflow is cancelled. If at least one validation result is 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.

SYNTAX

VAL-check-validation-result [-each_validation_target]

ARGUMENTS

-each_validation_target

(Optional) At least one validation result must exist for each NX dataset for the workflow to proceed.

If this argument is not used, the workflow proceeds if there is a successful result on one NX dataset.

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.

TCRS-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

TCRS-has-target-drawing

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

TCRS-check-status

DESCRIPTION

Initiates a workflow process if the current and the previous revisions have a valid release status.

SYNTAX

TCRS-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

Parameter	Description
-previous_status	Status on the predecessor revision to be tested (last release status of the predecessor revision).
-previous_check	For the previous revision, check either the last release status or the entire list for valid status.
-current_status	Status on the target revision to be checked.
-current_check	For the target revision, check either the last release status or the entire list for valid status.
-stop	Set to **Y** to stop the process or **N** to continue with warning.

PLACEMENT

Place on the **Start** action of a root task.

RESTRICTIONS

None.

EXAMPLES

• In this example, the handler ensures that previous revisions of the target item revision have any status type. If not, the handler stops and an error message is displayed.

Argument	Values
-previous_status	**any**
-stop	**Y**

• In this example, the handler ensures that previous revisions of the target item revision have no status type. If any status type is found, the handler stops and an error message is displayed.

Argument	Values
-previous_status	**none**
-stop	**Y**

• In this example, the handler ensures that previous revisions of the target item revision have **TCM Released** as their last status type. If not, the handler stops and an error message is displayed.

Argument	Values
-previous_status	**TCM Released**
-previous_check	**last**
-stop	**Y**

• In this example, the handler ensures that previous revisions of the target item revision have had **TCM Released** as their status type at any time. If not, the handler stops and an error message is displayed.

Argument	Values
-previous_status	**TCM Released**
-previous_check	**all**
-stop	**Y**

• In this example, the handler ensures that target item revisions have any status type. If not, the handler stops and an error message is displayed.

Argument	Values
-current_status	**any**
-stop	**Y**

• In this example, the handler ensures that target item revisions have no status type. If any status type is found, the handler stops and an error message is displayed.

Argument	Values
-current_status	**none**
-stop	**Y**

• In this example, the handler ensures that target item revisions have **TCM Released** as their last status type. If not, the handler stops and an error message is displayed.

Argument	Values
-current_status	**TCM Released**
-current_check	**last**
-stop	**Y**

TCRS-check-signoff

DESCRIPTION

Checks the signoff users against signoffs from other task.

SYNTAX

TCRS-check-signoff -task=$PREVIOUS|$NEXT

ARGUMENTS

-task =$PREVIOUS | $NEXT

When the argument is set to **$PREVIOUS**, the handler checks the task before the current task. If set to **$NEXT**, the handler checks the task after the current task. The default is **$PREVIOUS**.

PLACEMENT

Place on the **Complete** action of the **select-signoff-team** task.

RESTRICTIONS

None.

EXAMPLES

In this example:

- The workflow uses two **Review** tasks, **Task 1** and **Task 2**.

- The **TCRS-check-signoff** handler is placed on the **Complete** action of the **select-signoff-team** task for **Task 2**, with the following argument:

Argument	Values
-task	**$PREVIOUS**

The handler compares the **Task 2** signoff team with that of **Task 1**, and displays an error message if it finds a reviewer who is a member of both teams.

TCRS-check-prev-itemrev-status

DESCRIPTION

This handler checks whether a release process is valid for an item revision, based on its current status, and checks whether the target item revisions are released with specified status. You can also check whether the target item revision is unreleased and whether it currently has no status.

SYNTAX

TCRS-check-prev-itemrev-status -status=status_name | -unreleased | -latest

ARGUMENTS

-status

A valid status name to be checked.

-unreleased

Ensures that all target item revisions have no status.

-latest

- When this parameter is specified, the handler validates the most current status of the target item revision and ensures the latest status on target item revision is same as status specified in **-status** parameter.

- When this parameter is not specified, the handler validates all statuses attached to target item revisions.

PLACEMENT

Place on the **Start** action of the root task of a release process. If an error occurs, this rule handler prevents the start of the release process.

RESTRICTIONS

The **latest**, **unreleased** and **targetstatus** parameters are mutually exclusive.

EXAMPLES

- In this example the handler ensures that target item revisions are released with status **Approved**.

Argument	Value
-status	**Approved**

- Use the **-unreleased** argument to ensure that all target item revisions are not yet released and have no attached status.

Argument	Value
-unreleased	

- In this example, the handler ensures that all target item revisions have **90** as their last status.

Argument	Value
-status	**90**
-latest	

- In this example, the workflow stops if any of the target item revisions are already released with status **60**.

Argument	Value
-status	**60**
-targetstatus	

TCRS-check-jobowner

DESCRIPTION

Checks that the owner of a certain stage (task) of a release process cannot delegate approval.

SYNTAX

TCRS-check-jobowner [-who=jobowner] [-task=this]

ARGUMENTS

Parameter	Description
-who	User ID to examine.
	Currently, **jobowner** is the only valid value. This parameter is reserved for possible future extensions.
-task	Task to examine.
	Currently, **this** is the only valid value. This parameter is reserved for possible future extensions.

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.

TCRS-check-itemrev-status

DESCRIPTION

Checks the status of target item revisions.

SYNTAX

TCRS-check-itemrev-status [-status=status-type **[-unreleased]
[-latest] [-targetstatus]**

ARGUMENTS

Parameter	Description	Required
-status	All target objects must be released with this status type.	No
-unreleased	All target objects should be without a release status.	No
-latest	Validates that the target item revision is the latest released item revision.	No
-targetstatus	Stops the handler if any of the target objects are released	No

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

TCRS-check-datasets

DESCRIPTION

Checks that datasets attached to the target item revision are of the specified type. This handler also checks that the dataset name matches the specified pattern.

SYNTAX

TCRS-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

Parameter	Description	Default	Required
-type	The type of dataset for the item revision.		Yes
-search_dataset_name	The names of the datasets to be searched. Separate multiple names with a semicolon (**;**).		Yes
-check_include_dataset_name	Identifies the dataset. This argument accepts a single value only.		Yes
-stop	Determines whether or not to stop the workflow when the attached dataset names do not match the name in the **-search_dataset_name** argument.	**y**	No
-dataset_to	Defines the location of the text dataset which contains the errors in case of failure. This text dataset is stored at the location specified in this argument. Valid values are target object (**target**), an attachment to the item revision (**item**), or an attachment to the job (\).	**target**	No

PLACEMENT

Requires no specific placement.

RESTRICTIONS

All item revisions must have write privileges at the level that the handler is used.

- The following example checks all UGPART datasets with an **EZ** or **GZ** prefix in their names and **ANT** as postfix.

Argument	Values
-type	**UGPART**
-search_dataset_name	**EZ;GZ**
-check_include_dataset_name	**EZ**
-dataset_to	**item**

- In the following example, the handler checks whether a PDF dataset named **testDoc** is attached to the target item revision. Also, the error log is attached to the item if the dataset is not found.

Argument	Values
-type	**PDF**
-search_dataset_name	**testDoc**
-check_include_dataset_name	**testDoc**
-stop	**y**
-dataset_to	**item**

TCRS-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

TCRS-check-comps-against-pattern -mode=[list|check_only] -file=_dataset-name_

ARGUMENTS

Parameter	Description
-mode	Defines how the check should be performed. Valid values are:
	• **list**
	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.
	• **check_only**
	The dataset named reference is replaced with the latest information.
-file	Specifies the name of the dataset that should be attached as a reference to the workflow process.

PLACEMENT

Must be set in the **Start** action.

RESTRICTIONS

Handler should not be put after the **Complete** action.

TCRS-check-bomchild-statuslist

DESCRIPTION

Checks all components of a target assembly in a BOM view revision for a valid status.

SYNTAX

TCRS-check-bomchild-statuslist -rule=*configurationrule*

-statelist=*status***[,***status***] [-check_job=[y|n]]. [-log=[error<all]]**
[-stop=[y|n]] [-maxdepth=*depth***]**

ARGUMENTS

Parameter	Description	Value
-rule	Configuration rule.	
-statelist	List of valid status names. Separate multiple names with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.	
-check_job	Defines the terms of the component status.	• **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.
-log	Log data record.	• **error** = Record incorrect components only. • **all** = Record all component examinations.
-stop	Warning in the event of an error (=**n**) or Workflow with error stop (=**y**)	Warning in the event of an error (=**n**) or Workflow with error stop (=**y**)
-maxdepth	Level in the assembly to be checked.	• **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.

TCRS-check-bom-precise

DESCRIPTION

Checks whether all BOM view revisions are precise.

SYNTAX

TCRS-check-bom-precise [-stop=[y|n]] [-maxdepth=*depth*]

ARGUMENTS

Parameter	Description
-stop	Valid type of form.
-maxdepth	Levels to be checked. The value 0 corresponds to all levels.

PLACEMENT

Must be set in the **Complete** action of the **perform-signoffs** task.

RESTRICTIONS

None.

TCRS-check-approver

DESCRIPTION

Compares the lists of assigned users for two specified tasks. If the same user is assigned to both tasks, the handler displays a warning message or stops the task, depending on the value you enter for the **-stop** argument. You specify the tasks with the **-a_task** and **-b_task** arguments, and the user with the **-a_user** and **-b_user** arguments.

SYNTAX

TCRS-check-approver -a_task =[*task-name***|$PREVIOUS|$NEXT] -a_user=[***userid***|$USER] -a_jobowner -b_task =[***task-name***|$PREVIOUS| $NEXT] -b_user=[***userid***|$USER] -b_jobowner -stop=[Y|N]**

ARGUMENTS

Arguments	Values	Definition
-a_task	[*task-name*|$PREVIOUS|$NEXT]	All signoffs of this task are assigned to set **A**.
-a_user	[*userid*|$USER]	The user specified or the current user is assigned to set **A**.
-a_jobowner	n/a	The owner of the job is added the quantity of **A**.
-b_task	[*task-name*|$PREVIOUS|$NEXT]	All signoffs of this task are assigned to set **B**.
-b_user	[*userid*|$USER]	The user specified or the current user is assigned to set **B**.
-b_jobowner	n/a	The owner of the job is added the quantity of **B**.
-stop	[Y|N]	The job stops if a signoff is found. Default is **Y**.

PLACEMENT

Place on the **Complete** action of the **select-signoff-team** task.

> **Tip**
>
> You can also place the handler on the **Complete** action of the **perform-signoffs** task.

RESTRICTIONS

None

EXAMPLES

• If the same user is assigned to task **R1** and task **R2**, the handler returns **EPM_nogo** and stops the task.

Argument	Value
-a_task	**R1**
-b_task	**R2**
-stop	**Y**

- If the user who is currently logged on is assigned to the **R2** task, the handler displays a warning message but does not stop the task.

Argument	Value
-a_user	**$USER**
-b_task	**R2**
-stop	**N**

- If the same user is assigned to both task **R1** and task **R2**, or if the **R2** signoff list includes the job owner, the handler returns **EPM_nogo** and stops the task.

Argument	Value
-a_task	**R1**
-a_jobowner	n/a
-b_task	**R2**
-stop	**Y**

TCRS-generate-pdf

Description

Converts the attached dataset of a specified type and relation to the PDF format and reattaches the generated PDF as a dataset with a specified relation.

Syntax

TCRS-generate-pdf-exe=_PDF convertor tool executable path_**-input_key**
=_Input parameter name_ **-output_key**
=_Output parameter name_ **-additional_args=**Additional parameters which are expected
by the tool**-input_dataset_relation=**_Relation of Dataset which is to be converted to PDF_
-output_relation=_Relation with which the newly generated PDF will be attached_

Arguments

Arguments	Values	Required
-exe	Defines the revision rule to be applied for BOM traversal.	Value

Arguments	Values	Required
-input_key	Specifies the input parameter name that accepts the input file path of the PDF converter tool. If the tool does not accept the input key, this will be an optional argument. For example, **–in_file**.	No
-output_key	Specifies the output parameter name which accepts the output file path of the PDF converter tool where the new file will be placed. If the tool does not accept the output file path then this will be an optional argument. For example,**–out_file**	No
-additional_args	Specifies any additional parameters expected by the tool with the name value pair. If the tool does not accept any additional parameters then this will be an optional argument.	No
-input_dataset_type	Specifies the type of dataset that must be converted to PDF. You can specify multiple dataset types using comma-separated values.	Value
-input_dataset_relation	Specifies the relation of dataset that must be converted to PDF. You can specify multiple dataset types using comma-separated values.	Value
-output_relation	Specifies the relation with which the newly generated PDF is attached.	Value

Placement

Requires no specific placement.

Restrictions

User must install a third party PDF convertor tool on the server as a prerequisite for this handler. All item revisions must have *write* privileges at the level that the handler is used.

Notes

This handler accepts comma separated input dataset types and relations. For exmaple, **-input_dataset_type=MSWord,Bitmap**, or **-input_dataset_relation=IMAN_specification,IMAN_reference**. Consider that the above parameters are provided with comma-separated values to the handler and an item revision has 6 datasets of following types and relations attached:

•   MSWord Dataset1 (IMAN_specification)

•   MSWord Dataset2 (IMAN_ reference)

- Bitmap Dataset3 (IMAN_specification)

- Bitmap Dataset4 (IMAN_ reference)

- MSWord dataset5 (IMAN_rendering)

- Text Dataset6 (IMAN_specification)

In this case, the handler converts the datasets that are matched with the specified types and relations. Therefore only **Dataset1**, **Dataset2**, **Dataset3** and **Dataset4** will be converted to PDF. These four new PDF datasets will be attached as a datasets to the item revision while retaining the old ones as they are.

Therefore, **Dataset5** and **Dataset6** are not converted to the PDF format (as the **IMAN_rendering** relation and **Text** type are not specified as an input).

Example

Consider that **PDF Editor**, a third-party PDF generation tool is installed on the server at **C:\Program Files (x86)\PlotSoft\PDFill\PDFill.exe**.To convert an image file of the type Bitmap (.bmp) to PDF and save it at **C:\out.pdf**, enter the following command:

```
C:\Program Files (x86)\PlotSoft\PDFill\PDFill.exe OCRC:\image.bmp C:\out.pdf
```

In this example, the tool accepts **OCR** as an input parameter key. The following table lists the arguments that you must provide.

Argument	Value
-exe	C:\Program Files (x86)\PlotSoft\PDFill\PDFill.exe
-input_key	OCR
-input_dataset_type	bitmap
-input_dataset_relation	IMAN_specificaiton
-output_relation	IMAN_Rendering

The **input_key**, **output_key**, and **additional_args** parameters vary depending on the PDF generation tool. In this example, the **output_key** and **additional_args** parameters are not provided as these are not required by the **PDF Editor** tool.

Once the handler with the above parameters is executed, the input dataset of type **Bitmap** with the relation **Specifications** is converted to PDF. This generated PDF is reattached as a dataset with the **Rendering** relation.

TCRS-bom-plmxml-export

Description

Exports targets and references information to an XML file. Use this handler to export targets and references data to an XML file.

Syntax

TCRS-bom-plmxml-export-context=_transfer-mode_
-viewtype=_view-type-to-export_**-attach=[target|**
reference]-revrule=_revision-rule_**-path=**_data-export-path_**-prefix=**_filename-prefix_**-postfix=**_filename-prefix_

Arguments

Arguments	Definition	Optional/ Mandatory	Value/Default Value
-context	Defines the context string, which specifies the transfer mode used for export.	Mandatory	**transfer_mode**
-viewtype	Specifies the view type that you want to export.	Optional	**view**
-attach	Specifies which workflow process attachments are exported. If not specified, only targets are exported.	Optional	**target**
-revrule	Specifies the revision rule to be applied for the BOM lines while exporting the structure. If you do not specify a value, the latest revision is used.	Optional	**Latest Working**
-path	Specifies the path where you want to export the data. The exported file is saved to the server machine. If not defined, the data is exported to **TC_TMP_DIR**.	Optional	**TC_TMP_DIR**
-prefix	Specifies the prefix text for the exported file name.	Optional	None
-postfix	Specifies the post-fix text for the exported file name.	Optional	None

Placement

Requires no specific placement.

Restrictions

None.

Example

This example shows how to export the targets in the workflow to an XML file using the
ConfiguredDataExportDefault transfer mode and the **Latest Working** revision rule.

Argument	Value
-context	ConfiguredDataExportDefault
-attach	target
-revrule	Latest Working

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.

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 list of the relation types used to relate ERP forms to item revisions.

Separate multiple types with commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

> **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.

PS-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

PS-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:

Argument	Values
-note_types	**Torque,Power**

PS-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 progress, 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

PS-check-assembly-status-progression [-rev_rule=_revision-rule_**] [-saved_var_rule=**_saved-variant-rule_**] [-status=** _status-being-applied-to-the-target-object_**][-check_component_status=** _component-status-to-be-checked-against_**] [-check_unconfigured]**

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.

-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:

Argument	Values
-rev_rule	**Working**
-saved_var_rule	**GMC 300 Rule**

- 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:

Argument	Values
-check_component_status	**Design**

- 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:

Argument	Values
-status	**Design**

- 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**

o **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:

```
<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

o   Add the **UserData** blocks between the **<PLMXML>** and **</PLMXML>** tags.

o   Ensure you increment the **UserData id** value when you add a new entry.

o   After adding a new **UserData** block, change the value for **Type** to a type you are defining.

o   You can modify the value of the release status to meet your requirements.

MROCORE-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

MROCORE-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 service lifecycle management Service Manager or As-Built Manager is licensed and installed.

MFG-invoke-customized-validations

DESCRIPTION

Performs customized validation checks for Manufacturing Execution System Integration. This handler does the following:

- Takes the **CC** object and create BOP windows.

- Configure all windows with the configuration rule.

- Calls the validation checks for any BOP window.

If a validation check fails or there is an error or warning, it is returned within the **validationError** structure and added to the log in the handler or in the user interface.

SYNTAX

MFG-invoke-customized-validations -Type = *callback-type-1, callback-type-2, ...* **-Name =***callback-name-1, callback-name-2, ...* **[-ContinueOnFail = True|False, True|False, ...]**

ARGUMENTS

-Type

The callback type; for example, **MFG_ValidationChecksCallback** or **MESINTEG_ValidationChecksCallback**. Each **-Type** value is paired with the **-Name** value, separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. You can have more than one type/name pair.

-Name

The callback name; for example, **ValidationCheck1**. Each **-Type** value is paired with the **-Name** value, separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. You can have more than one type/name pair.

-ContinueOnFail

(Optional) Whether or not to continue checking if the previous check failed. The default is **False**. You can use multiple values, separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. There should be one less value than the number of type/name pairs, because if the last check fails, there is not another check to continue to.

PLACEMENT

Place this handler on any workflow that transfers a **CC** object to a BOP window.

RESTRICTIONS

None.

EXAMPLES

- This example runs three different validation checks, **ValidationCheck1**, **ValidationCheck2**, and **ValidationCheck3**. If **ValidationCheck1** fails, the handler runs **ValidationCheck2** anyway. If **ValidationCheck2** fails, the handler does not run **ValidationCheck3**.

Argument	Values
-Type	**MFG_ValidationChecksCallback, MFG_ValidationChecksCallback, MFG_ValidationChecksCallback**
-Name	**ValidationCheck1, ValidationCheck2, ValidationCheck3**
-ContinueOnFail	**True, False**

MESINTEG_ValidateReleaseAndExport

DESCRIPTION

Performs customized validation checks for Manufacturing Execution System Integration. This handler does the following:

- Takes the **CC** object and creates BOP windows.

- Configures all windows with the configuration rule.

- Calls the validation checks for any BOP window.

If a validation check fails or there is an error or warning, it is returned within the **validationError** structure and added to the log in the handler or in the user interface.

SYNTAX

MESINTEG_ValidateReleaseAndExport -Type = *callback-type-1, callback-type-2, ...*

ARGUMENTS

-Type

Specifies the callback type, for example, **MFG_ValidationChecksCallback** or **MESINTEG_ValidationChecksCallback**. Each **-Type** value is paired with the **-Name** value, separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. You can have more than one type/name pair.

-Name

Specifies the callback name, for example, **ValidationCheck1**.

Each **-Type** value is paired with the **-Name** value, separated by commas or the character specified in the **EPM_ARG_target_user_group_list_separator** preference. You can have more than one type/name pair.

-perform

Specifies the list of operations to be performed by the action handler.

Values include **Validate**, **Release**, **GenerateMESWIRep**, **Export**, and **modifyscope**.

> **Note**
>
> Specify these values without spaces and separated by commas or the character specified in the **EPM_ARG_target_user_group_list_separator** preference

-fullexport

Indicates whether it is a full export or a delta export.

-ContinueOnFail

(Optional) Specifies whether to continue checking if the previous check fails. The default value is **False**. You can use multiple values, separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. Specify one value less than the number of type/name pairs, because if the last check fails, there is no check to continue.

-export_as_fai

Specifies whether to consider the work package as part of the the **Send to MES** command.

If set to **True**, the work package is considered as a part of the **Send to MES** command.

PLACEMENT

Place this handler on any workflow that eventually creates a BOP window from the **VisStructureContext**, exports the data, and updates the release status.

RESTRICTIONS

None.

EXAMPLES

Arguments used in the **ReleaseToMES**, **Send**, and **ReleaseUpdateToMES** workflows.

> **Note**
>
> Specify values without spaces and separated by commas or the character specified in the **EPM_ARG_target_user_group_list_separator** preference

Argument	Values
-Type	**MFG_ValidationChecksCallback**, **MFG_ValidationChecksCallback**, **MFG_ValidationChecksCallback**
-callback_name	**Release Status Validation**, **Workarea Assigned Validation**, **Process Hierarchy Validation**, **Workarea Name Validation**
-perform	**Validate**, **Release**, **GenerateMESWIRep**, **Export**
-fullexport	**True**
-ContinueOnFail	**True** or **False**
-export_as_fai	**True, False**
	If this property is set to **True**, the work package is considered as a part of the **Send to MES** command.

Arguments used in the **ReleaseToProduction** workflow.

> **Note**
>
> Specify values without spaces and separated by commas or the character specified in the **EPM_ARG_target_user_group_list_separator** preference

Argument	Values
-Type	**MFG_ValidationChecksCallback**
-callback_name	**Change Object Validation**
-target	**production**

Argument	Values
-perform	**Validate**, **Pending**,**Export**, **exportdelta**, **Release**, **modifyscope**
	The **modifyscope** value is specific to **ReleaseToProduction** workflow. If you want to use this value, you must register this callback using the following command:
	install_callback -u=_infodba_ **-p=**_password_ **-g=**_dba_ **-mode=**_create_ **-type=**_MFG_ModifyScopeCallback_ **-library=**_library_ **-function=**_function_ **-name=**_Modify Export Scope_
	Note Do not use **modifyscope**, **Pending**, or **exportdelta** values for MES Integration.
-fullexport	**TRUE**
-ContinueOnFail	**TRUE**
-export_as_fai	**True, False**
	If this property is set to **True**, the work package is considered as a part of the **Send to MES** command.

LDF-sync-ldf-status

Queries the remote Linked Data Framework (LDF) integrated systems, such as Polarion, for properties, and checks their values against the expected values configured.

- If the values match, the handler applies the configured status to the target(s) and allows the task to continue processing.

- If the expected values do not match, the handler does not allow a task to continue processing.

Querying a remote system like Polarion is accomplished through APIs against LDF objects attached to the root task by target or reference relations, or attached to a target or reference by a specified relation or property.

> **Note**
>
> Arguments specific to applying release status are the same as the **EPM-set-status** handler. Any added, modified, or deleted **EPM-set-status** handler arguments apply to the **LDF-sync-ldf-status** handler arguments.

SYNTAX

LDF-sync-ldf-status –property=<oslc-namespace-prefix-url>.property-name
  [-remote_user_name=user_name**]**
  [-attachment={target / reference / both**}] [-attachment_property=**property-name**]**
  [-attachment_relation=relation-name
  ] [-include_type=include-type**]**
[-include_related_type=include_related_type**] [-check_first_object_only]**
[-[action={append/rename/replace/delete**}]**
[-status=name**]**
[-new_status=new-status**]**
[-retain_release_date] [-set_effectivity]

ARGUMENTS

Parameter	Description	Default	Req.
-property:: <oslc-namespace-prefix-url> . property-name	Specifies the remote property or properties check. Requires a fully qualified property name with a prefix URL prepended to every property in a workflow argument, which is prepended by **–property::**. The OSLC namespace prefix URL must be contained in angle brackets, **<** and **>**, in the <oslc-namespace-prefix-url		Yes

Parameter	Description	Default	Req.
	>.property-name format as shown in the *Examples* section.		
	Enter a list separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.		
-remote_user_name	Used by the handler to connect to a remote system like Polarion for sending HTTP requests.		No
	The *Restrictions* section describes separate actions required to generate an encrypted password file.		
-attachment	Specifies the type of attachment to be checked:	**target**	No
	target		
	Checks the target attachments		
	reference		
	Checks the reference attachment		
	both		
	Checks target and reference types of attachments.		
-attachment_property	Property of the attachment to derive the linked object.		No
-attachment_relation	Specifies the relation name to expand to get the linked object from workflow attachment. Linked objects attached to targets and references of a workflow with the relation specified by **attachment_relation** are searched. Linked objects not matching the specified relation are not checked.	**Lcm0Affected ByDefect**	No

Parameter	Description	Default	Req.
-include_type	Specifies the type of workflow target and reference attachments to be checked. Workflow attachments not matching the specified type are not checked.	**target**	No
-include_related_type	Specifies the type of linked object to retrieve that is related to the workflow attachment using the **attachment_relation** value. This argument should be used in conjunction with the **attachment_relation** or **attachment_property** arguments.	**target**	No
-check_first_object_only	If specified, only the first object of the type specified by **include_type** is considered. This argument is optional.	**true**	No
-status	When the check is satisfied, a new milestone with the name specified by this argument is added to targets and references of the workflow.	**task-name**	No
-action	Specifies an action: **append** Attaches the status objects from the root task to the target objects, with no impact to 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**.	**append**	No

Parameter	Description	Default	Req.
	If a status object with the **old_name** status is not found, it renames the last status object attached to the target objects.		
	If the target object has an existing status, the status object is renamed from **old_name** to **new_name**.		
	delete		
	Deletes the status **status_name** specified by the **status** argument from the target object.		
	If the delete argument is not used in combination with the **status** 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.		
-new_status	Specifies the new name for the status object.		No
	Use in conjunction with **rename** and **replace** actions.		
-retain_release_status	Retains the original release date on the target object if it had previously been released. Not valid for **replace**.	**false**	No
-set_effectivity	If used, the system creates the open-ended date effectivity with the release date as the start date.	**false**	No

PLACEMENT

Because this is a rule handler with some action handler behavior, place it as the last rule handler in the rule handler list for the task **Complete** action.

RESTRICTIONS

Use if you are using the LDF framework for application integrations and you want Teamcenter workflows to apply status based on LDF linked property values.

You must generate an encrypted password file by following these steps in a Teamcenter command shell:

1. Run this command:

   mkdir %TC_DATA%\polarionconnector

2. Run this command:

   **%TC_ROOT%\bin\install -encryptpwf
   -f=%TC_DATA%\polarionconnector**<*user name*>

   Where <*user name*> is user name of remote system such as Polarion ALM. This user name should be configured as a value of the **–remote_user_name** handler.

- The following example checks the status property of linked objects on the remote system.

Argument	Values
-property:: *<http://polarion.plm.automation. siemens.com/oslc#>* **.priority**	**Low**, **Medium**
-attachment	**target**
-attachment_relation	**Lcm0RelatedChangeRequest**
-include_type	**ChangeRequestRevision**
-status	**Synced**
-action	**append**
-remote_user_name	**admin**

ICS-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

ICS-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 (**DocumentRevison**, 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:

Argument	Values
-allowed_type	**ItemRevision**

This handler is very useful in restricting unclassified items and item revisions from being released.

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:

  o   If the attribute parameter is required, the field must have a value.

  o   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 an overview of using LOVs in handlers, see *Lists of values as argument values*.

  o   For string attributes, the length of string entered must be no more than that defined in the schema.

  o   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:

  o   Check a corresponding BOMView revision of the correct type exists, as described for the **SAP-check-forms-attached-RH** handler.

  o   Check all components with the same item ID have the same attribute values (for those attributes specified in the mapping schema, except quantity).

  o   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.

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-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.

EPM-verify-digital-signature

DESCRIPTION

Verifies if the target objects and, optionally, the schedule task have a valid digital signature.

SYNTAX

EPM-verify-digital-signature [-include_schedule_task] [-quorum=*size*] [-no_void]

ARGUMENTS

-include_schedule_task

(Optional) Verifies the digital signature on the schedule task and all target objects of the workflow. If this argument is not provided, the digital signature is verified only on the target objects of the workflow.

-quorum

(Optional) Specifies the minimum number of valid digital signatures each target must have, where *size* is a positive integer specifying the quorum. If this argument is not specified, all digital signatures on all targets must be valid.

-no_void

(Optional) Checks each target object in the workflow for a void digital signature. If the target object has one or more void digital signatures, the handler fails with an error indicating the failure, even if the quorum in the **-quorum** argument for valid digital signatures is met.

PLACEMENT

Place on any action on any task.

RESTRICTIONS

None.

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.

SYNTAX

EPM-validate-target-objects
[-include_type =*type-of-workspace-object[, type-of-workspace-object2,..]***]**
[-exclude_type =*type-of-workspace-object[, type-of-workspace-object2,..]***]**
[-latest_rev]

ARGUMENTS

-include_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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference between the types. This argument is optional.

Accepts valid Teamcenter object types, such as **ItemRevision**, **UGMASTER**, and **UGPART**.

When you add any object type or class as a target, all its subtypes are also included. To explicitly exclude any subtypes, use the **-exclude_type** argument.

For example, if this argument is specified as **ItemRevision**, any type of item revision (for example, **DocumentRevison**, and so on, and any custom item revision types) is allowed.

Does not accept bracketed () class notation to distinguish between classes and types.

-exclude_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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference 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, **DocumentRevison**, 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
-include_type	**ItemRevision**

- This example allows **MEOPRevision** objects as the targets and disallows **MENCMachining Revision** and **METurningRevision** objects:

Argument	Values
-include_type	**MEOPRevision**
-exclude_type	**MENCMachining Revision**, **METurningRevision**

> **Note**
>
> **MEOPRevision** is the parent type (class) for **MENCMachining Revision** and **METurningRevision**. In this example, all **MEOPRevision** subtypes are allowed as targets except for **MENCMachining Revision** and **METurningRevision**.

- This example allows only the latest item revisions as targets:

Argument	Values
-include_type	**ItemRevision**
-latest_rev	

EPM-signoff-team-validation

DESCRIPTION

Checks to ensure the minimum number of reviewers specified by the **-num_reviewers** argument is assigned to the **select-signoff-team** task. If no argument is provided, the handler checks for at least one reviewer.

If the number of reviewers assigned to the **select-signoff-team** task is less than the minimum reviewers required, then **EPM_nogo** is returned.

SYNTAX

EPM-signoff-team-validation [-num_reviewers= *minimum-number*]

ARGUMENTS

-num_reviewers

(Optional) Minimum number of reviewers required for the **select-signoff-team** task.

PLACEMENT

Place *only* on the **Complete** action of the **select-signoff-team** task.

RESTRICTIONS

None.

EXAMPLES

This example checks to see if at least 2 reviewers are assigned to the **select-signoff-team** task.

Argument	Values
-num_reviewers	**2**

EPM-invoke-system-rule

DESCRIPTION

Runs an external command (specified with the **-command** 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 **-command** 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

EPM-invoke-system-rule -command=*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] [-comment=*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***]**
[-depth=*maximum-recursion-depth***] [-debug]**

ARGUMENTS

-command

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 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 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.

-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

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, a comma, or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

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.

-expand

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 on the **Start** or **Complete** action of any task. If this handler is configured to set the signoff decisions on a **perform-signoffs** task (for example, if the **-comment** argument is specified), then place on the **Complete** action of the **perform-signoffs** task.

RESTRICTIONS

Do not add to a workflow process containing *any* handler using resource pools.

EXAMPLES

This example shows how to run the **iman_check_renderings_pl** script using the **-command** 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.

> **Note**
>
> The script should be placed in the *TC_ROOT*/**bin** directory.

Argument	Values
-command	**iman_check_renderings_pl**

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.

This handler checks the **task_result** property of the task to which it is attached. If this property is not set to **Completed**, this handler pauses the task. If the value is set to **Completed**, the task progresses normally.

In addition, in case of **Notify** tasks that are sub-tasks of **Route** tasks, this handler checks whether the reviewers are completely assigned to the **Route** task. If the reviewers' assignment is complete, then it allows the **Notify** task to proceed even if the value of **task_result** property of the **Notify** task is not set to **Completed**.

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

ARGUMENTS

None.

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-disallow-reviewers

DESCRIPTION

Prevents specified users, the workflow process owner, reviewers for a specified task, reviewers from all tasks, or a combination of them from being added to a signoff team in a **Review** task.

SYNTAX

EPM-disallow-reviewers -assignee=user:[*user-name-1*] [,**user:***user-name-2*,...] | [**user:$PROCESS_OWNER**] **-task=[***parent-task-name***:***sub-task-name* | **ALL]**

ARGUMENTS

-assignee

Specifies the user IDs and/or the workflow process owner that are not allowed as reviewers.

Any Teamcenter users or **$PROCESS_OWNER** are specified in the following format:

user:*user-name-1*, **user:***user-name-2*, ...

You must use either the **-assignee** or the **-task** argument. You can optionally use both.

-task

Specifies the parent task and subtask names, separated by a colon (:), for an existing **select-signoff-team** task in the workflow process. Reviewers for this task are not allowed as reviewers for the task with this handler. You can specify all tasks in the workflow process with the **ALL** keyword.

You must use either the **-assignee** or the **-task** argument. You can optionally use both.

PLACEMENT

Place *only* on the **Complete** action of the **select-signoff-team** task.

RESTRICTIONS

None.

EXAMPLES

• This example prevents the user **Smith** from being a reviewer:

Argument	Values
-assignee	**user:Smith**

• This example prevents the workflow process owner and user **Smith** from being reviewers:

Argument	Values
-assignee	**user:$PROCESS_OWNER**, **user:Smith**

• This example prevents the existing reviewers on the **Review1:SST1** task from being reviewers:

Argument	Values
-task	**Review1:SST1**

• This example prevents the existing reviewers on all other **select-signoff-team** tasks within the workflow process from being the reviewers:

Argument	Values
-task	**ALL**

• This example prevents the process owner and existing reviewers on the **Review1:SST1** task from being reviewers:

Argument	Values
-assignee	**user:$PROCESS_OWNER**
-task	**Review1:SST1**

EPM-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

EPM-disallow-removing-targets

ARGUMENTS

None.

PLACEMENT

Place on the **Perform** action of the root task.

RESTRICTIONS

None.

EPM-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 **PS-attach-assembly-components** handlers are dependent on this handler.

SYNTAX

EPM-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.

Argument	Values
-exclude_type	BOMView Revision

EPM-debug-rule

DESCRIPTION

Notifies a user that an action is executing. Attaching **EPM-debug-rule** to any EPM action notifies the user when that task action runs by printing that action name to the standard output device.

SYNTAX

EPM-debug-rule -comment=_string_

ARGUMENTS

-comment

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 runs by printing **Complete**, **action is executing** to the standard output device.

Argument	Values
-comment	**action is executing**

Note

This example assumes you have attached this handler to a **Complete** action.

EPM-check-target-object

DESCRIPTION

Checks the status of the object to determine whether to allow the action.

> **Note**
>
> Enable debugging functionality for this handler with the
> **TC_HANDLERS_DEBUG** environment variable.

SYNTAX

EPM-check-target-object -allowed_status=
*status-name***| -disallowed_status=***status-name*

ARGUMENTS

-allowed_status

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

Indicate *in process* status:

IN_PROCESS

-disallowed_status

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:

*|all|ALL|any|ANY

Indicate *no* status with one of the following:

null|NULL|none|NONE

Indicate *in process* status:

IN_PROCESS

PLACEMENT

Place on the **Perform** action of the root task.

RESTRICTIONS

None.

- This example allows any target to be attached with a status of **Pending** or with no status (work in progress):

Argument	Values
-allowed_status	**Pending, NONE**

- This example disallows any targets from being attached with a status of **Released** or **Obsolete**:

Argument	Values
-disallowed_status	**Released,Obsolete**

EPM-check-target-attachments

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. For an overview of using LOVs in handlers, see *Lists of values as argument values*.

> **Note**
>
> Enable debugging functionality for this handler with the
> **TC_HANDLERS_DEBUG** environment variable.

SYNTAX

EPM-check-target-attachments { {-include_type=target-object-type
 -include_related_type=attached-object-type
 -relation=relation-type**} | -lov=**lov-name**}**
 [-allowed_status=valid-status-names **| ANY | NONE]**

ARGUMENTS

-include_type
Defines the type of target object to be checked.

-include_related_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 impacted item of a change object with **CMHasImpactedItem**.

- Specify a solution item of a change object with **CMHasSolutionItem**.

- Specify a problem item of a change object with **CMHasProblemItem**.

- Specify a reference item of a change object with **CMReferences**.

- Specify a change object that implements another change object with **CMImplements**.

-allowed_status

Specifies the required status of the attachment. Multiple statuses can be checked by listing valid Teamcenter statuses separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

ANY checks for any status. **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 **-include_type**, **-include_related_type**, and **-relation** arguments. It can be used with the **-allowed_status** argument to check relation status.

See the LOV row, for the required LOV format.

LOV

For an overview of using LOVs in handlers, see *Lists of values as argument values*.

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.

For an overview of using multilevel object paths in handlers, see *Defining multilevel object paths*.

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.

[$TARGET.]*target-(class)-or-type-1*

*relation1***.***sec-obj-(class)-or-type-in-target-1*

*relation2***.***sec-obj-(class)-or-type-in-target-1*

[$TARGET.]*target-(class)-or-type-2*

*relation1***.***sec-obj-(class)-or-type-in-target-2*

*relation2***.***sec-obj-(class)-or-type-in-target-2*

…

> **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.

[$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 **$TARGET** or identified by their lack of dots (.).

[(Class)[!Type1][,(Class2)[,Type1[,…]]]]

For example, to specify that all item revisions are checked except software revision:

(ItemRevision)!Software Revision

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

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:

Argument	Values
-include_type	ChangeItemRevision
-include_related_type	ItemRevision
-relation	CMHasProblemItem
-allowed_status	ANY

- This example checks the targeted change revision for an item revision with no status in the **Impacted Items** folder:

Argument	Values
-include_type	ChangeItemRevision
-include_related_type	ItemRevision
-relation	CMHasImpactedItem
-allowed_status	NONE

- This example checks the targeted change revision for the **CORP_Part** revision with a released status in the **Solution Items** folder:

Argument	Values
-include_type	ChangeItemRevision
-include_related_type	CORP_PartRevision
-relation	CMHasSolutionItem
-allowed_status	Released

Alternatively, you can use these LOV settings:

Argument	Values
-lov	SYS_EPM_check_target_attachments
-allowed_status	Released

where the **SYS_EPM_check_target_attachments** LOV contains this data:

> **$TARGET.ChangeItemRevision**
> **CMHasSolutionItem.CORP_PartRevision**

- 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:

Argument	Values
-include_type	ChangeItemRevision
-include_related_type	ItemRevision
-relation	CMHasSolutionItem
-allowed_status	Concept Approval,Funding Approval,Design Approval

- This example checks the targeted change revision for an item revision in the **Solution Items** folder, irrespective of status:

Argument	Values
-include_type	ChangeItemRevision
-include_related_type	ItemRevision
-relation	CMHasSolutionItem

- This example performs specific relation checks for particular revision type targets and other relation checks for the remaining revision types all with no status:

Argument	Values
-lov	SYS_EPM_check_target_attachments
-allowed_status	NONE

where the **SYS_EPM_check_target_attachments** LOV contains this data:

Value	Description
Software Revision, DocumentRevision **IMAN_specification.Text**	Check that any software and document revision targets have a text dataset attached in the **IMAN_specification** relation.
DocumentRevision **IMAN_specification.Word, Excel, PowerPoint**	Check that any **DocumentRevision** targets also have a Word, Excel OR PowerPoint dataset attached in the **IMAN_specification** relation.
(ItemRevision)!Software Revision! DocumentRevision **IMAN_specification.UGMASTER** **IMAN_specification.UGPART**	Check that any other targets of class **ItemRevision**, (in other words, that are not **SoftwareRevision** or **DocumentRevision**) have a **UGMASTER** and **UGPART** attached in the **IMAN_specification** relation.
(ItemRevision) **Proj.Project**	Check that any revision targets also have a project item attached to the custom **Proj** relation.

Note

The relation lines are indented for clarity.

EPM-check-status-progression

DESCRIPTION

Checks the complete release status progression of a specific object. For example, this handler identifies the last status added on any item revision because the handler considers that the latest status for that item revision.

- This handler 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_released_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**.

  For example: An object has revisions **A**, **B**, and **C**. Revision **A** is released later than revision **B**, and **C** is not released. The **latest_rev** option returns **A**.

- The latest release status of the greatest release status of the target object, use **greatest_released_rev**.

  For example: An object has revisions **A**, **B**, and **C**. Revision **A** is released later than revision **B**, and **C** is not released. The **greatest_released_rev** option returns **B**.

> **Note**
>
> The **EPM-check-status-progression** rule handler first identifies the last status added on an item revision. The handler considers that the latest status for that item revision. Then this handler looks at the various **-rev** arguments to determine which revision to use.
>
> When checking the last status added to each revision, status maturity is established by the release status order in the **ProgressionPath.plmxml** file.

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:

Argument	Values
-status	Design

- In this example, consider the scenario:

  o Progression path: **Quote**, **Experimental**, **Development**, **Design**, **Manufacturing**, **Production**

  o IR ABC123

  o IR ABC123/001 has **Experimental** status

  o IR ABC123/002 in **Working** state

  o 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.

Argument	Values
-rev	current_rev

- 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.

Argument	Values
-rev	**previous_rev**

• 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.

Argument	Values
-rev	**latest_rev**

• In this example, consider the progression path and values:

  o Progression path: **Quote**, **Experimental**, **Development**, **Design**, **Manufacturing**, **Production**.

  o IR XYZ123

  o IR XYZ123/001 has **Design** status

  o IR XYZ123/002 has **Experimental** status

  o IR XYZ123/003 has **Development** status

  o IR XYZ123/004 status not yet applied

  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**.

Argument	Values
-rev	**greatest_released_rev**

ADDITIONAL INFORMATION

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:

```
<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-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

EPM-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.

EPM-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

EPM-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.

Argument	Values
-responsible	**User:george, Group:dba**

EPM-check-related-objects

DESCRIPTION

Checks whether the specified target object contains the required secondary related objects, and whether those objects are in process or have achieved a valid status. 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.

> **Note**
>
> If this handler is checking multiple objects, all objects must meet the criteria to satisfy this handler.

SYNTAX

EPM-check-related-objects [-include_type=*type-of-target-object***]**
{-primary_type=*type-of-target-object***| -secondary_type=***secondary-object-type***}**
-relation=*relation-type* **[-allowed_status=***status-names* **| ANY | NONE |**
IN_PROCESS]
[-check_first_object_only]

ARGUMENTS

-include_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 impacted item of a change object with **CMHasImpactedItem**.

- Specify verification of a solution item of a change object with **CMHasSolutionItem**.

- Specify verification of a problem item of a change object with **CMHasProblemItem**.

- Specify verification of a reference item of a change object with **CMReferences**.

- Specify verification of a change object that implements another change object with **CMImplements**.

-allowed_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 **-include_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:

Argument	Values
-include_type	**ItemRevision**
-secondary_type	**xyz**
-relation	**IMAN_specification**
-allowed_status	**Released**

- This example checks for a primary attachment that is a **ChangeItemRevision**, currently in process, and attached to the target item revision with a **CMHasImpactedItem** relation:

Argument	Values
-include_type	**ItemRevision**
-primary_type	**ChangeItemRevision**
-relation	**CMHasImpactedItem**
-allowed_status	**IN_PROCESS**

- This example checks for a primary **ChangeItemRevision** attachment that is either a change request (ECR) or change notification (ECN), that is in process, and attached to the target item revision with a **CMHasImpactedItem** relation. This checks for both **ChangeRequestRevision** and **ChangeNoticeRevision ChangeItemRevisions**, whether in process or not:

Argument	Values
-include_type	**ItemRevision**
-primary_type	**ChangeItemRevision:: ChangeRequestRevision~ ChangeNoticeRevision**
-relation	**CMHasImpactedItem**
-allowed_status	**IN_PROCESS**

- This example checks for any released secondary **xyz** attachment with an **IMAN_specification** relation to the **type1** target object:

Argument	Values
-include_type	**type1**
-secondary_type	**xyz**
-relation	**IMAN_specification**
-allowed_status	**ANY**

- This example checks for a secondary **xyz** attachment with no status in the **Impacted Items** folder of the target change object revision:

Argument	Values
-include_type	**ChangeItemRevision**
-secondary_type	**xyz**
-relation	**CMHasImpactedItem**
-allowed_status	**NONE**

- This example checks for a secondary dataset attachment with a working status attached to the target item revision. Defining the **secondary_type** as **Dataset** checks for all dataset types of the defined relation:

Argument	Values
-include_type	**ItemRevision**
-secondary_type	**Dataset**
-relation	**IMAN_specification**
-allowed_status	**NONE**

- 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:

Argument	Values
-include_type	ItemRevision
-secondary_type	xyz
-relation	IMAN_specification
-allowed_status	Released
-check_first_object_only	

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 message lists those properties.

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 -include_type=_object-type_
-property=_property-names_
[-value=_required-values_**]**
 [-attachment=_attachment-type_**]**
[-check_first_object_only] [-include_replica]

ARGUMENTS

> **Note**
>
> To check for a single property value that is not null, omit the **-value** argument.

-include_type

Specifies the type of the workflow target/reference attachments to be checked. Workflow attachments not matching the specified type are not checked.

> **Caution**
>
> This argument is required.

This argument is used in cases where the check is used only on a specific type subset of workflow attachments, particularly if that property is specific to that type and not found on others.

> **Note**
>
> Multiple values can be added to **-include_type** by using a comma-separated list.

> **Note**
>
> An error does not occur if target/reference objects do not match the **-include_type** value.

-property

Specifies the properties to be checked. Enter a list separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

> **Note**
>
> If the handler uses a property that references a group member and its value is being checked, then the value should be specified as: **group/role/person name (user id)**.

> **Caution**
>
> If you specify a property of the **Reference** type, the handler checks the referenced object, not the workflow attachment.

-value

Specifies the required real values to be checked. Enter real values as defined in Business Modeler IDE.

> **Caution**
>
> Do not enter localized values.

Enter a list separated by commas or the character specified by the **EPM_ARG_target_user_group_list_separator** preference. The order of these values must match the order of properties listed in the **-property** argument.

This argument is optional.

> **Note**
>
> If **-value** is not specified, then any populated value will be accepted.

-attachment

Specifies the type of attachment to be checked.

- **target**
  Checks the targets attachment.

- **reference**
  Checks the reference attachment.

- **schedule_task**
  Checks the schedule task attachment.

- **both**
  Checks **target** and **reference** 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.

-include_replica

(Optional) Checks the **Replica Proposed Targets** as well as the target objects if the **-attachment=target** argument is also specified.

If the **-attachment=schedule_task** argument is specified with this argument, it ignores the attached schedule object if it is a proxy link of schedule task.

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:

Argument	Values
-include_type	**CMII CR Form**
-property	**cr_priority,prop_soln**
-attachment	**target**

- 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:

Argument	Values
-include_type	**CMII CR Form**
-property	**cr_priority,cr_type**
-value	**1 = High,Corrective Action**
-attachment	**target**

- 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:

Argument	Values
-property	**cr_priority,prop_soln,cr_type**
-value	**1 = High,,Corrective Action**
-include_type	**CMII CR Form**
-attachment	**target**

> **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:

Argument	Values
-include_type	CMII CR Form
-property	cr_priority,cr_type,prop_soln
-value	1 = High,Corrective Action
-attachment	target

> **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:

Argument	Values
-include_type	CMII CR Form, CMII CN Form
-property	cr_priority,prop_soln,cr_type
-value	1 = High,,Corrective Action
-attachment	both
-check_first_object_only	

EPM-check-item-status

DESCRIPTION

Verifies that all secondary relations connected by **ImanRelations** of a target item or item revision have been released or that these secondary objects 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

EPM-check-item-status [**-relation=***relation-name*]
[**-include_related_type=***object-type*]
[**-allowed_status=***status-name-to-check*]

ARGUMENTS

-relation

Relation name.

-include_related_type

Object type.

-allowed_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**:

Argument	Values
-relation	**IMAN_requirement**
-include_related_type	**Text**
-allowed_status	**X**

• 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:

Argument	Values
-include_related_type	**UGPART**

EPM-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 **-source_task** argument to specify the name of the preceding **Condition** task and the **-decision** argument to specify the result (**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

EPM-check-condition -source_task= *task-name***-decision= {true |
false |***custom-result* **| ANY |** *error-code*}

ARGUMENTS

-source_task

Specifies the name of the preceding **Condition** task. 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.

-decision

Specifies the result that must be met. Use this argument in conjunction with a **Condition** task, placing this handler on a successor task. Valid values are the following:

• *custom-result*

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 or the character specified by the **EPM_ARG_target_user_group_list_separator** preference.

> **Note**
>
> This value is automatically set when you use the **Set Custom Result**
> option to configure the flow path from the **Condition** task to the
> successor task.

- **ANY**

  Use this value 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 value 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.

- *error-code*

  Use this value 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 value is automatically set when you use the **Set Error Codes** option
  > to configure a failure path from the **Validate** task to the successor task.

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**
>
> 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.

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 -responsible=[owner|$OWNER] | [group|$GROUP] | [$RESPONSIBLE_PARTY] | [privileged | $PRIVILEGED] | [*group*::{*|*role*}] | [*role*]

ARGUMENTS

-responsible

Checks if the user matches the specified value. Valid values are:

- **owner | $OWNER**

  Specifies the owner of the task.

- **group | $GROUP**

  Specifies that the current user's logged-on group be the same as one of the groups of the task's responsible party.

- **$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.

Argument	Values
-responsible	**privileged**

- This example allows any member of the **engineering** group to trigger the action.

Argument	Values
-responsible	**engineering::***

• This example allows any user with the role of **manager** to trigger the action.

Argument	Values
-responsible	**manager**

• This example allows any user with the role of **designer** in the **engineering** group or the **Project Administrator** role in the **Project Administration** to trigger the action.

Argument	Values
-responsible	**Project Administration::Project Administrator, engineering::designer**

• This example allows any user with the role of **designer** in the **structure** subgroup of the **engineering** group to trigger the action.

Argument	Values
-responsible	**structure.engineering::designer**

EPM-assert-targets-checked-in

DESCRIPTION

Verifies that all target objects in this workflow process are checked in.

SYNTAX

EPM-assert-targets-checked-in

ARGUMENTS

None.

PLACEMENT

Requires no specific placement.

RESTRICTIONS

None.

EPM-assert-signoffs-target-read-access

DESCRIPTION

Checks if all the selected reviewers have read access to the attached target attachments.

SYNTAX

EPM-assert-signoffs-target-read-access [-check_assignee=$RESOURCE_ POOL_ALL]

ARGUMENTS

-check_assignee

If the selected reviewer is a resource pool, checks if all members of the resource pool have read access to the attached targets.

The only valid value is **$RESOURCE_POOL_ALL**.

PLACEMENT

Place on the **Complete** action of a **select-signoff-team** task.

RESTRICTIONS

None.

AUTOSCHEDULING-person-reassign-validate

DESCRIPTION

Verifies that when a workflow task with an attached job card or job task is reassigned to another user, that user has the discipline (skill) and qualifications specified on the job card or job task.

SYNTAX

AUTOSCHEDULING-person-reassign-validate

ARGUMENTS

None.

PLACEMENT

Place on the **Start** action of the **perform-signoffs** task.

RESTRICTIONS

None.

ASMAINTAINED-validate-missing-asmaintained-structure

DESCRIPTION

Validates the as-maintained structure does not contain any missing or unidentified physical parts.

SYNTAX

ASMAINTAINED-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 service lifecycle management Service Manager is licensed and installed.

ASMAINTAINED-validate-for-unserviceable-physicalpartrevision

DESCRIPTION

Checks the as-maintained structure for any unserviceable physical parts.

SYNTAX

ASMAINTAINED-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 service lifecycle management Service Manager is licensed and installed.

ASMAINTAINED-validate-for-latest-asmphysicalpartrevision

DESCRIPTION

Checks if the target physical part revision is the latest revision.

SYNTAX

ASMAINTAINED-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 service lifecycle management Service Manager is licensed and installed.

ASMAINTAINED-validate-for-checkedout-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

ASMAINTAINED-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 parts.

RESTRICTIONS

This handler is available only when Teamcenter service lifecycle management Service Manager is licensed and installed.

ASBUILT-validate-missing-structure

DESCRIPTION

Validates the as-built structure does not contain any missing or unidentified physical parts.

SYNTAX

ASBUILT-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 service lifecycle management Service Manager or As-Built Manager is licensed and installed.

ASBUILT-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

ASBUILT-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 service lifecycle management Service Manager or As-Built Manager is licensed and installed.

Siemens Industry Software

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

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

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

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

About Siemens PLM Software

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