TEAMCENTER

Active Workspace Customization

Teamcenter 2412

Unpublished work. © 2025 Siemens

This Documentation contains trade secrets or otherwise confidential information owned by Siemens Industry Software Inc. or its affiliates (collectively, “Siemens”), or its licensors. Access to and use of this Documentation is strictly limited as set forth in Customer’s applicable agreement(s) with Siemens. This Documentation may not be copied, distributed, or otherwise disclosed by Customer without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This Documentation is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this Documentation without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made.

No representation or other affirmation of fact contained in this Documentation shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

If you have a signed license agreement with Siemens for the product with which this Documentation will be used, your use of this Documentation is subject to the scope of license and the software protection and security provisions of that agreement. If you do not have such a signed license agreement, your use is subject to the Siemens Universal Customer Agreement, which may be viewed at **https://www.sw.siemens.com/en-US/sw-terms/base/uca/**, as supplemented by the product specific terms which may be viewed at **https://www.sw.siemens.com/en-US/sw-terms/supplements/**.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS DOCUMENTATION INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS DOCUMENTATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TRADEMARKS: The trademarks, logos, and service marks (collectively, "Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens’ Marks may be viewed at: **www.plm.automation.siemens.com/global/en/legal/trademarks.html**. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

About Siemens Digital Industries Software

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

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

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

Contents

How can I customize Active Workspace? 1-1

Configuring the user interface

User interface configuration for Active Workspace 2-1
Configuring the universal viewer 2-1
Modifying commands 2-6
Changing the default thumbnail 2-7
Display environment information 2-12
Display a banner 2-13
Organizing your users' common destinations 2-14
- Tile reference 2-17
- Tile collection reference 2-20
- Tile relation reference 2-21
- Tile template reference 2-23
- Create a new tile using the rich client 2-25

Dynamic compound properties 2-26
- Learn about dynamic compound properties 2-26
- Dynamic compound property column behavior 2-28
- Using dynamic compound properties with XRT 2-32
- Using dynamic compound properties with column configuration 2-33
- Dynamic compound property syntax 2-33

Configuring tables 2-43
- Two types of tables in Active Workspace 2-43
- Configuring declarative tables 2-47
- Configuring object set tables 2-66
- Table properties (are not objectSets) 2-76

Configuring page layout 2-78
- XRT information specific to Active Workspace 2-78
- Working with HTML panels in XRT 2-88

Changing Active Workspace behavior

The Active Workspace customization process 3-1
The Active Workspace gateway architecture 3-3
- The Active Workspace gateway architecture 3-3
- Using the production logon 3-5
- Using developer mode 3-6
- Using the development server 3-7
- Work with a declarative module 3-9
- Commit your UI Builder changes to your module 3-10
- Build your module 3-11
- Publish your local site 3-12

The Active Architect workspace 3-13

© 2025 Siemens

What is Active Architect? 3-13
Command Builder 3-14
Panel Builder 3-23
Configuration Builder 3-32
Active Architect workshop 3-46
The Active Workspace development environment 3-56
The stage directory 3-56
The declarative customization project 3-57
Development scripts 3-60
Verify your customizations using the development server 3-63
Active Workspace development examples 3-65

Working with platform customizations

Integrating Teamcenter platform customizations 4-1
Enable a custom business object in Active Workspace 4-1
Adding custom type icons 4-2
Registering icons using conditions 4-5
Enabling your custom preferences in Active Workspace 4-7
Enable your custom workflow handler in Active Workspace 4-8

Active Workspace Customization Reference

Troubleshooting 5-1
Open source software attributions 5-1
Examine the application context 5-1
Use logical objects to consolidate properties 5-4
Logical Objects 5-4
Logical object configuration 5-5
Destination criteria 5-8
Compound logical objects 5-9
Workspaces 5-11
Learn about workspaces 5-11
Create a custom workspace definition 5-13
Modify your custom workspace definition 5-16
Modify an OOTB workspace definition 5-18
Create or update workspace mappings 5-19
Assign style sheets to a workspace 5-22
Assign a tile collection to a workspace 5-23
Create or modify column configuration for a workspace 5-27
Verifying your new workspace 5-28
Remove a workspace 5-29
XRT element reference 5-30
Active Workspace style sheet elements 5-30
Modifying style sheets using the XRT Editor 5-32
nameValueProperty 5-54

Architecture concepts

Learning Active Workspace architecture 6-1

© 2025 Siemens

How Active Workspace saves changes to properties 6-2
Learn declarative contributions 6-2
Declarative action: navigate 6-2
Declarative conditions 6-5
Declarative panel walk-through 6-10
What are intermediary objects? 6-13
Using visual indicators to quickly recognize a property 6-16
Using a sublocation to display a custom page 6-16
Declarative user interface 6-18
Declarative UI introduction 6-18
Declarative kit and module 6-22
Declarative control of sublocation visibility 6-25
Declarative view 6-27
Declarative view model 6-30
Declarative messages 6-33
Learn the declarative command architecture 6-35
Declarative command object - commands 6-35
Controlling command visibility 6-36
Declarative command object - commandHandlers 6-39
Declarative command object - commandPlacements 6-41
Declarative command object - activeWhen 6-44
Declarative command object - visibleWhen 6-46
Declarative command object - actions 6-48
Data providers 6-49
Learn about data providers 6-49
Use an existing server data provider 6-50
Creating a custom server data provider 6-54
Learn client-side data providers 6-55

© 2025 Siemens

© 2025 Siemens

1. How can I customize Active Workspace?

There are many ways to customize Active Workspace. Some examples are as follows:

- Improve your users' efficiency by hiding the commands which they do not need using *Command Builder* and *workspaces*.
- Decrease your users' cognitive load by arranging properties in meaningful ways using *style sheets*.
- Improve your users' capabilities by creating new *locations* and *sublocations*.

© 2025 Siemens

© 2025 Siemens

2. Configuring the user interface

User interface configuration for Active Workspace

How do I change the user interface for Active Workspace?

You generally change the user interface for Active Workspace using customization methods. However, as an administrator, you can perform simple configurations that only involve changing certain configuration files, such as style sheets.

Why configure the user interface?

You can emphasize information specific to your company's processes, such as displaying custom properties on tiles or exposing custom business object types.

What can I configure?

In Active Workspace, you can:

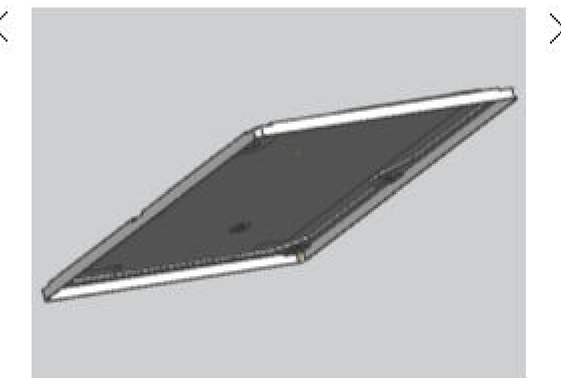
- **Configure the home page**.
- **Configure tables**.
- Change which properties appear on object cells.
- **Change the layout of the main work area**.
- **Add commands to the user interface**.
- **Enable your custom business object**.
- **Enable your custom workflow handler**.
- Define the revision rule list.

Configuring the universal viewer

What is the universal viewer?

The *universal viewer* is a gallery-style viewer element — it has left and right chevrons so your users can cycle through the available thumbnails.

© 2025 Siemens



It consists of a collection of individual viewers, each designed to handle specific single-file document or image formats in the client, such as Word, PDF, and PNG. The universal viewer renders single-file JTs, but not multi-file JT assemblies. To render the JTs in assemblies, use the **3D** viewer which requires installation of the **Active Content** feature.

It uses several third-party libraries to render the various file types. Consult the documentation for those libraries for possible restrictions and limitations.

Why would I use the universal viewer?

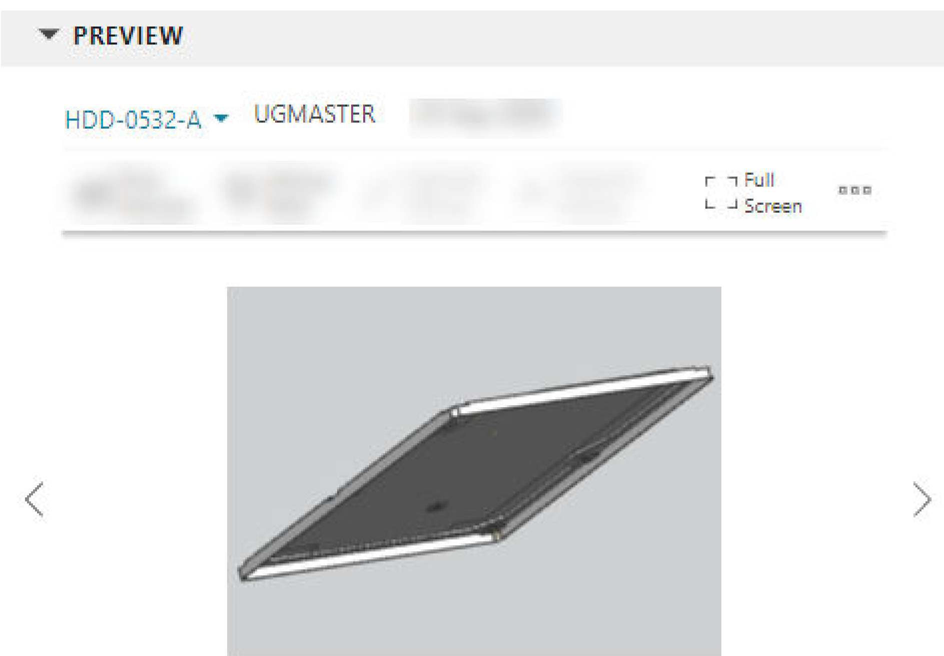
You can add a single viewer widget that can display many kinds of information without needing to know beforehand which kinds will be present or desired. The user can browse through all available information seamlessly.

How do I add the universal viewer to my style sheet?

The following XRT code snippet shows the OOTB **Preview** section. You can inject this source into your custom style sheets to use the universal viewer.

```
<section titleKey="tc_xrt_Preview">
   <inject type="dataset" src="Awp0GalleryViewer"/>
</section>
```


© 2025 Siemens



How does it work?

When the user selects an object, the universal viewer builds a list of available files and their associated viewers from the **AWC_defaultViewerConfig.VIEWERCONFIG** preference values.

This is a multiple-value (array) preference. Each entry is one of two types:

- **SEARCHORDER**.**objectType** = **relationType**, **relationType**, ...
- **datasetType**.**viewerName** = **namedReference**

Active Workspace recursively searches the **SEARCHORDER** values of the preference to build a list of datasets related to the selected object. These datasets form the list of available datasets for the gallery. When each dataset is displayed, the universal viewer searches the preference values to find the viewer associated with that dataset type. There can be *only one* line for each dataset type.

If you create custom relations, custom dataset types, or custom named references, you will need to add them to this preference if you want the viewer to recognize them.

If the gallery does not find any content to be displayed, it displays the icon for the object type.

> Example:
>
> If you select a folder, and it contains an item revision, and the revision has a GIF dataset as reference, then the values of the preference that are considered are as follows (spaces added for readability):

© 2025 Siemens

```
SEARCHORDER.Folder = contents
SEARCHORDER.ItemRevision = ...,IMAN_reference,...
GIF.Awp0ImageViewer = GIF_Reference
```

The final result is the **Awp0ImageViewer** showing the ***.gif** file contained in the dataset.

Tip:

In some cases, the user selects an **intermediary object**, which represents another object in the UI. This is common in structures and table rows. You must register the intermediary object for the viewer, not the underlying object.

Which viewers are available in the universal viewer?

Available viewers vary depending on your installed options and the software release you have. Following are *some* of the more common viewers and what they display:

Note:

Your list of available viewers may differ from the ones listed here.

- **Awp0CodeViewer**

  Various source code files, such as JavaScript, XML, JSON, and so on rendered with the Monaco editor.

- **Awp0HTMLViewer**

  Web-style content rendered using your browser.

- **Awp0ImageViewer**

  Renders 2D files using your browser. This displays **png**, **jpg**, **gif**, and anything else your browser supports.

- **Awp0JTViewer**

  Single or monolithic JT files. Not for assemblies of JT files.

- **Awp0PDFViewer**

  PDF files – uses a 3rd party PDF renderer.

- **Awp0TextViewer**

  Text files.

© 2025 Siemens

Additional Active Workspace features may include additional viewers specialized for that functionality. For example:

The Lifecycle Visualization server provides a 2D viewer for file types that your browser can not normally render.

- **Awp02dViewer**

The Requirements Manager functionality adds two additional viewers:

- **Arm0RequirementPreview**
- **Arm0RequirementDocumentationACE**

Set client-side rendering for the universal viewer

Depending on your needs, you can choose to render 3D data on the client or the server (on Windows or Linux). To set the rendering method for the universal viewer, use the **AWV0ViewerRenderOption** preference.

Configuring the universal viewer to view Microsoft Office files

You can view and edit Microsoft Office files in the universal viewer if Teamcenter Office Online is installed.

Configuring the universal viewer to view custom datasets or named references

If you create a custom dataset or named reference, you must create new values in the **AWC_defaultViewerConfig.VIEWERCONFIG** preference. For example, if you created a custom **C9PDF** dataset with a **C9PDF_Reference**, you must also create the following value (copied from PDF and modified) for the viewer to display your files:

C9PDF`.Awp0PDFViewer=`**C9PDF_Reference**

Configuring the universal viewer to view your assemblies

Normally, you would *not* use the universal viewer to view your assemblies. Instead, use the **Active Content Structure** feature's **3D** viewer tab.

However, there are certain cases when you might want to use the universal viewer (displayed in the **Overview** tab) for your assemblies instead. These special cases would be when your assemblies contain:

- Monolithic JT data instead of an individual JT at each part.
- No JT data at all.

© 2025 Siemens

- Your assemblies are non-CAD, consisting of other types of data, such as Microsoft Word documents, PDF files.

To use the universal viewer as the default for assembly data, there are a few things to be aware of:

- You must add your business object type and relations on a new **SEARCHORDER** line.
- A selected BOM line object is actually the runtime business object named **Awb0DesignElement**, an **intermediary object**.
- The relation that the design element object has to its target object is named **awb0UnderlyingObject**.

Example:

Add the following line to the preference to process any objects referenced by the design element object.

```
SEARCHORDER.Awb0DesignElement=awb0UnderlyingObject
```

If you have an assembly of **DocumentRevision** objects that have datasets attached with the **TC_Attaches** relation, Active Workspace does two things. It processes the new **Awb0DesignElement** line, finding the underlying **DocumentRevisions**, and then because of the OOTB line:

```
SEARCHORDER.DocumentRevision=TC_Attaches
```

It checks the **DocumentRevisions** for anything attached with the **TC_Attaches** relation.

Modifying commands

To help make your users' interface more streamlined and efficient, you can change various features of commands using the Active Architect workspace.

How can I change commands?

Icon

What the command *looks like* is specified in its **basic definition**. Command icons can be changed in the **Icon Preview** section on the **Commands** page of the **Command Builder**.

Placement

Where a command appears is part of its **commandPlacements** definition. Command placement can be changed in the **Placements** section of the **Command Builder**.

Visibility

© 2025 Siemens

When a command appears is part of its **commandHandlers** definition. Command visibility can be changed in the **Handlers** section of the **Command Builder**. **Command visibility** is determined by a combination of server-side and client-side conditions.

Action

What the command *does* is determined by its **commandHandlers** and its **action** definition. Command actions can be changed in the **Handlers** section and the **Actions** page of the **Command Builder**.

Labels

The default setting of whether command names are displayed is controlled by the **AW_show_command_labels** preference. If the user changes the **Labels** setting in the UI, a user preference instance is created for that user.

You can also set the default at the workspace level by adding the **showCommandLabel** setting in the workspace definition file. This can only be overridden by a user preference instance.

Example:

Contribute the setting to the **Author** workspace.

.../stage/src/workspace_contribution_*{myContribFile}.json*

```
{
    "schemaVersion": "1.0.0",
    "workspaceId": "TcAuthorWorkspace",
    "settings": {
        "showCommandLabel": false
    }
}
```

Example:

Adding the setting to your custom workspace.

.../stage/src/workspace_*{myWorkspace}.json*

```
...
"workspaceId": "...",
"workspaceType": "Exclusive",
...
"settings":{
    "showCommandLabel": false
},
...
```

Changing the default thumbnail

Active Workspace uses the following preferences to find a thumbnail image to be displayed. Each preference contains a prioritized list, and the preferences themselves follow the priority shown.

© 2025 Siemens

- **TC__thumbnail_relation_order**
- **TC__thumbnail_dataset_type_priority_order**
- **TC__thumbnail_dataset_type_order**

How does Active Workspace find a thumbnail image?

1. If the current object is not a dataset, follow any existing relations listed in **TC__thumbnail_relation_order**.
2. For each relation found, look for all datasets of the type listed in **TC__thumbnail_dataset_type_priority_order**.
3. From the list of datasets located, or if the original target was a dataset, examine the referenced files to locate a *named reference* listed in **TC__thumbnail_dataset_type_order**.
4. Display the first file on the list as the thumbnail.

   Note:

   If there are multiple files with the **UG-QuickAccess-Binary** reference (as with most NX datasets), Active Workspace displays the one named **images_preview.qaf** first.

5. If no file was located, display the icon for the business object as the thumbnail.

Examples

For these examples, assume your preferences are set as follows:

TC__thumbnail_relation_order

```
...
IMAN_specification
IMAN_manifestation
...
```

TC__thumbnail_dataset_type_priority_order

```
UGMASTER
UGPART
...
```

TC__thumbnail_dataset_type_order

```
Image
...
UG-QuickAccess-Binary
...
```


© 2025 Siemens

Each of the following examples show two files and which one is chosen.

Example:

This example highlights an OOTB situation where you might want the *Image* reference to be displayed instead of the *UG-QuickAccessBinary*, but it does not because *UGMASTER* has priority over *UPGART*.

1. IMAN_specification / **UGMASTER** / UG-QuickAccess-Binary

2. IMAN_specification / **UGPART** / Image

Example:

This example of a non-typical use case highlights how even though *UGMASTER* has priority over *UPGART*, and *Image* has priority over *UG-QuickAccessBinary*, the *relation order* preference ultimately has first priority.

1. **IMAN_specification** / UGPART / UG-QuickAccess-Binary

2. **IMAN_manifestation** / UGMASTER / Image

How can I add my own image dataset for thumbnails?

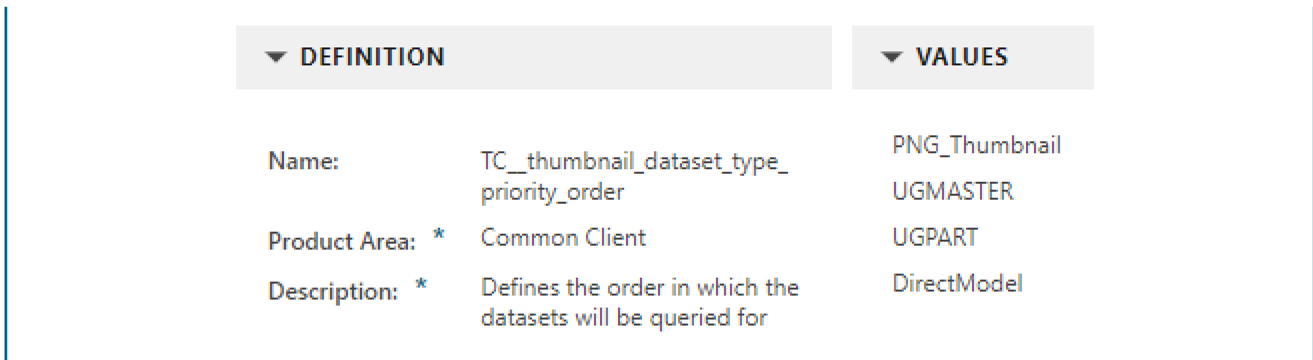
A common configuration is to use a special dataset exclusively for thumbnails. This assumes you already have a process to create and populate them. To configure your site to behave this way, you must modify the **TC__tumbnail** preferences. There are many existing datasets already defined in Teamcenter that you can use, or you can create your own.

Example:

In this example, your company decides to use the existing **PNG_Thumbnail** dataset to hold your thumbnail images. As **TC__thumbnail_dataset_type_priority_order** does not check this type of dataset, you must add it to the list. You decide to use the existing **Thumbnail** relation, as it is not only already on the list of relations but also has the highest priority. The same is true for the **Image** reference.

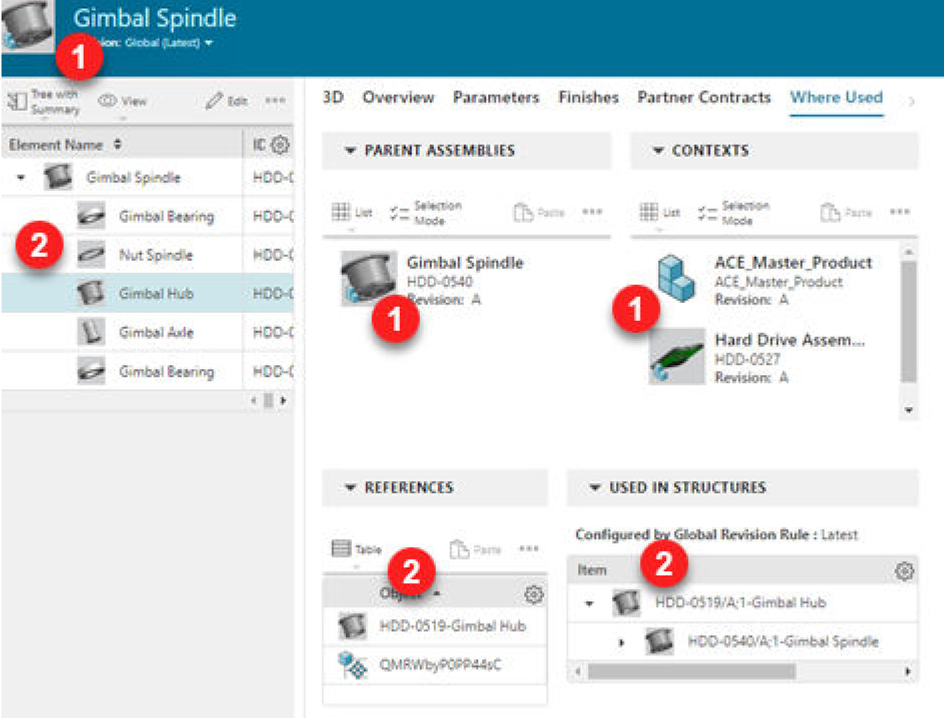
The only change you must make is to add the dataset type to the top of the *type priority order*.

© 2025 Siemens



Where do thumbnails appear?

You can find thumbnail images in many places throughout the UI.



1. **Location headers and lists**

   These also show an overlay of the business object type icon.

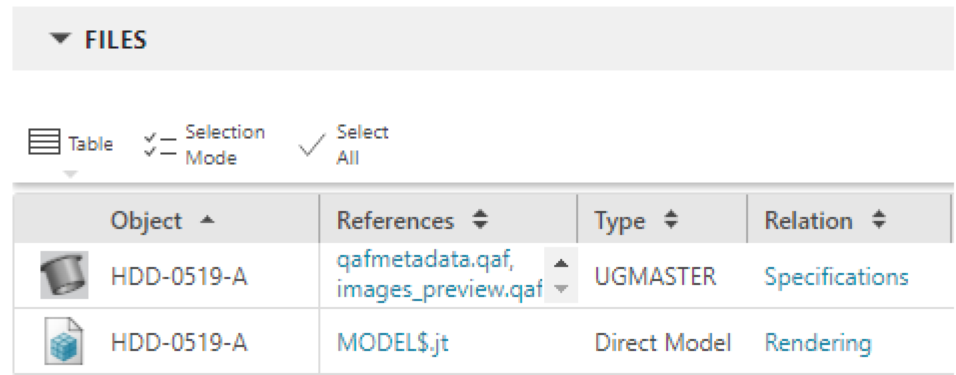
© 2025 Siemens

2. **Tables and trees**

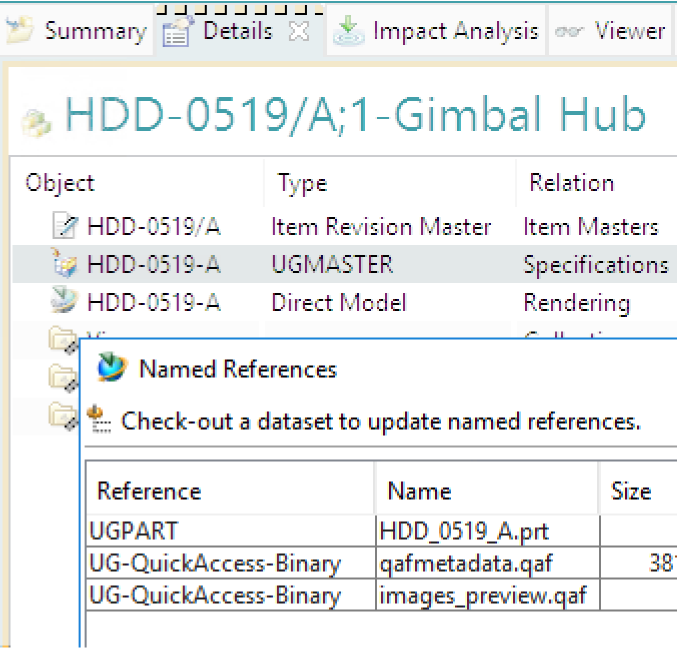
   These do not show the business object type icon because of their smaller size.

How can I view file references?

In Active Workspace, you can view a list of the files tracked by a dataset from the **FILES** section of the **Attachments** tab.



To view the actual **Reference** value for the files within a dataset, you must use the **Named References** dialog box in the rich client.



© 2025 Siemens

Display environment information

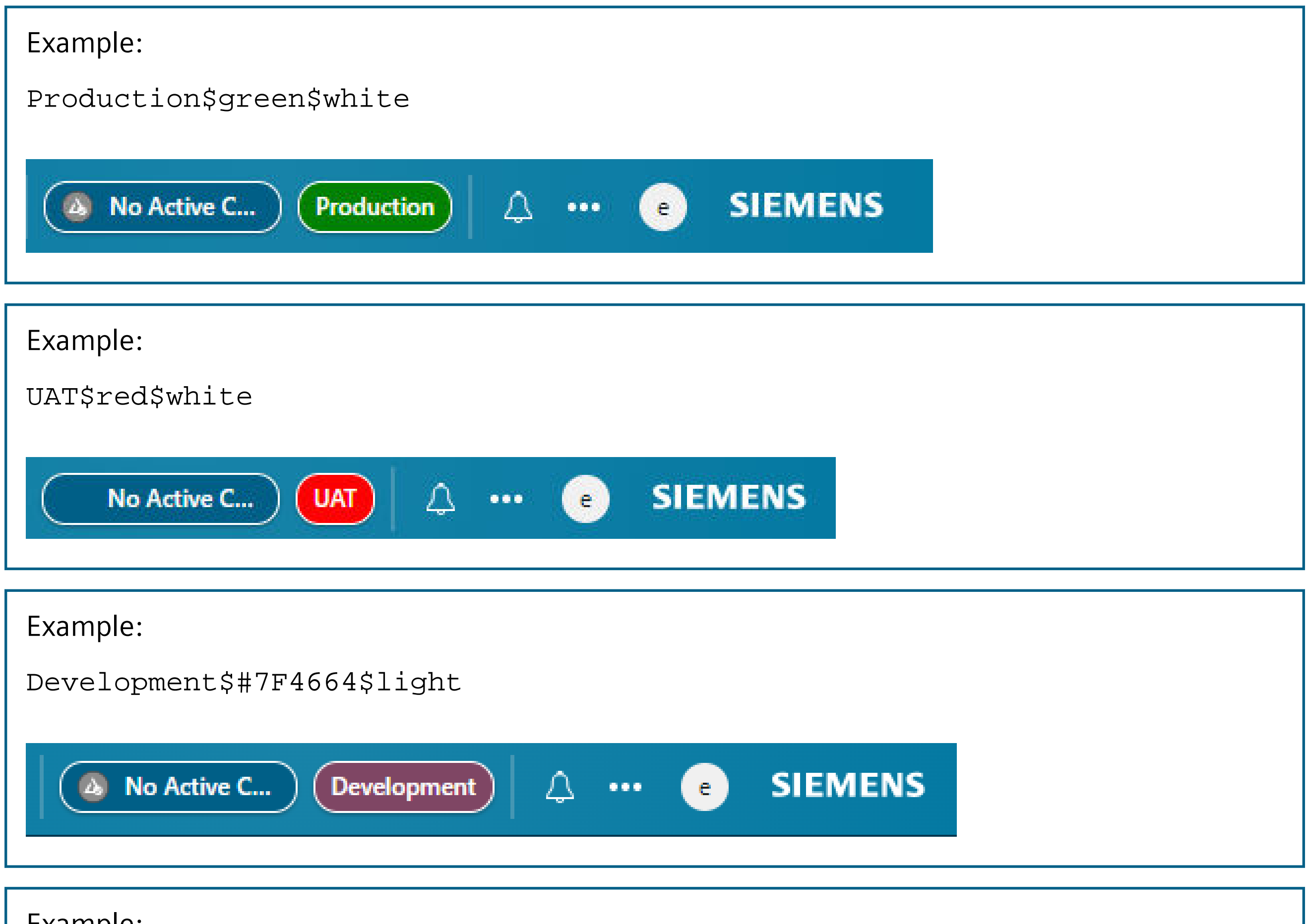
If your users connect to multiple Teamcenter environments, you can add information to the page so they can easily see which environment they are connected to.

To display the environment information, you must set the value of the **AWC_Display_Environment_Info** site preference. The value consists of three parts separated by dollar signs **$**.

```
label$background_color$text_color
```

The label is a non-localized literal string. You can label your environments by usage type, by content, or by any other method you want. It is recommended that you use labels that make sense to your users. You may also use the **$siteName** variable as your text, which substitutes Teamcenter's **Site Name**. This is a great option if you have already named your databases using custom **Site Name**.

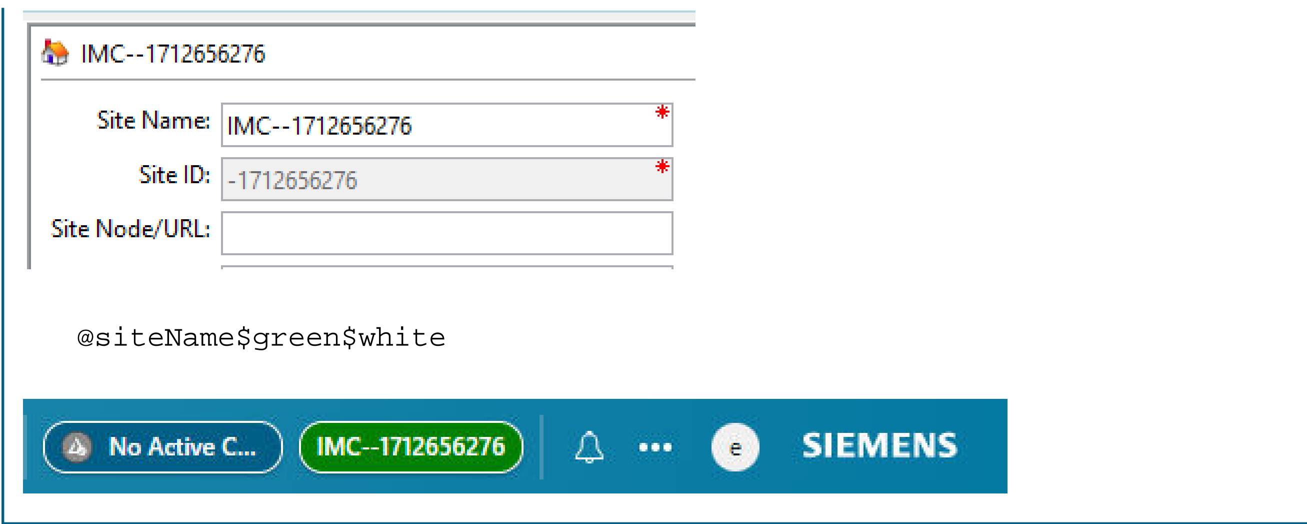
Background color and text color use hexadecimal RGB values like **#4C7F79**, or CSS standard names like **red**, **green**, **blue**, and so on.



Example:

You can display the existing **Site Name**, **IMC--1712656276** in this example.

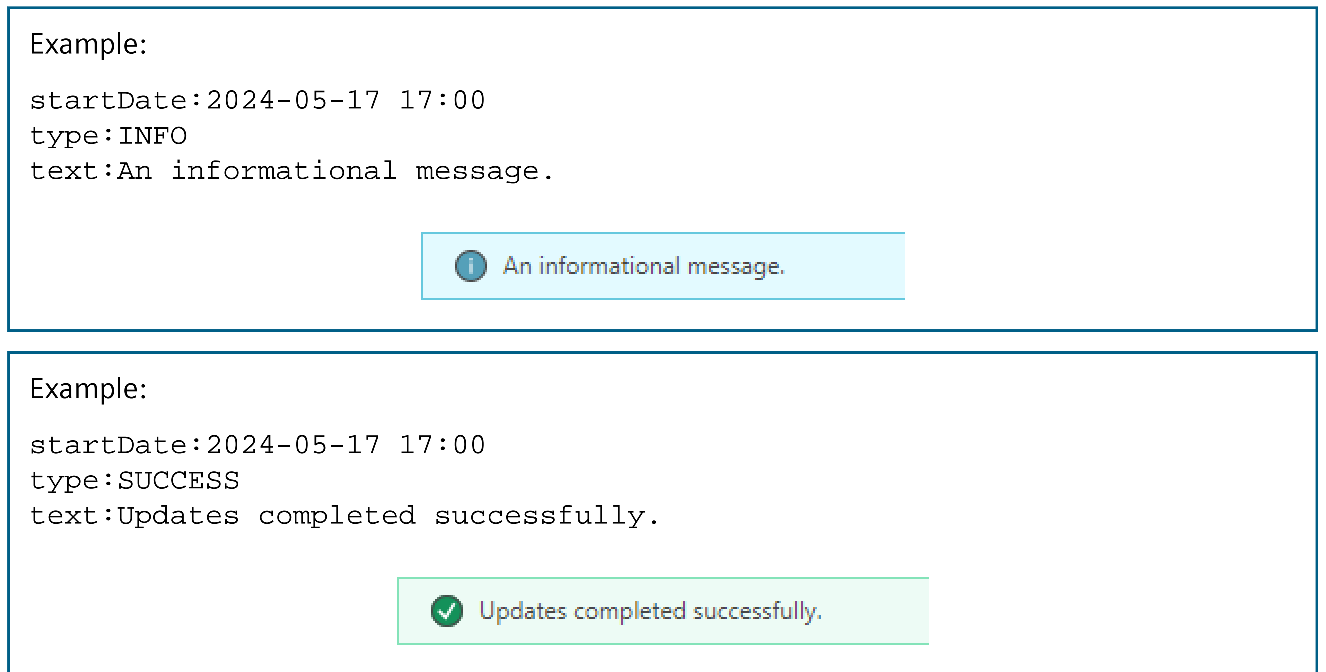
© 2025 Siemens



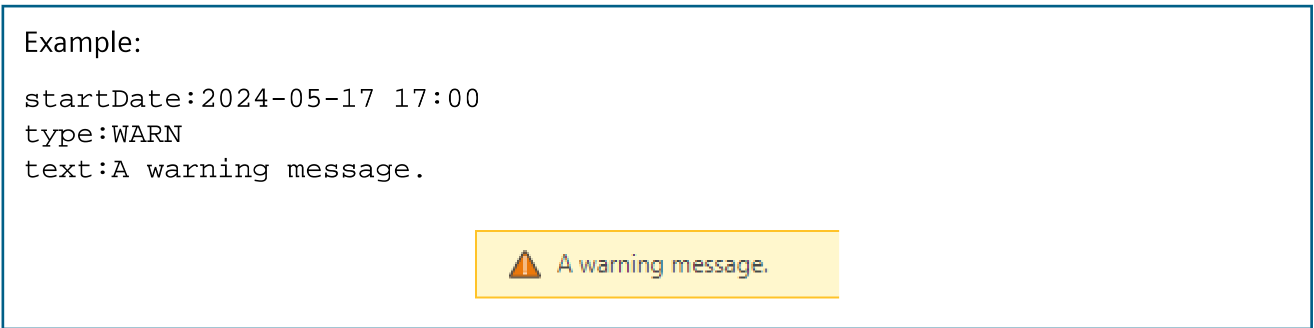
Display a banner

Create a banner which is displayed in the Active Workspace user interface to communicate messages to your users.

To display the banner, you must set values of the **AW_Banner** site preference. You must specify the message text, type, and beginning date and time. You can optionally set an ending time or a duration.



© 2025 Siemens


Example:

```
startDate:2024-05-17 17:00
type:WARN
text:A warning message.
```


Organizing your users' common destinations

What is the home page?

You can use Active Workspace home page to provide tiles for your users' most commonly used pages, objects, and saved searches.

The home page displays the same content for a given user regardless of their device type, whether it is a workstation, a laptop, or a mobile device.

© 2025 Siemens

How does it work?

The tiles are displayed on the page according to *tile collections*. Each tile collection is associated with a scope and can have multiple tiles related to it. The tiles shown to a user are based on a combination of all of the valid tile collections for their session context. All tiles from each valid collection are combined onto the display. As a result, the user might see duplicate tiles if the same tile is defined in more than one of their collections. To avoid duplicate tiles, Siemens Digital Industries Software recommends that you plan your tile collections based on the desired visibility. For example, if everyone needs to see a tile, place it in the *Site* collection. If you want only certain groups to see a tile, place it in those group collections, and the same with roles, workspaces, and projects. You may also hide tiles in certain collections, overriding a tile shown in another collection.

Users may have only one tile collection for each *scope*. All users get the tiles from the site collection. Then the appropriate group, role, workspace, and project collections are added based on their session context. Finally tiles in their user collection are added. If the user changes their session context, the home page is rebuilt using the collections appropriate to their new session context

What can I do with home page tiles?

The following tasks require that you are familiar with the object model (following) of the various tile objects.

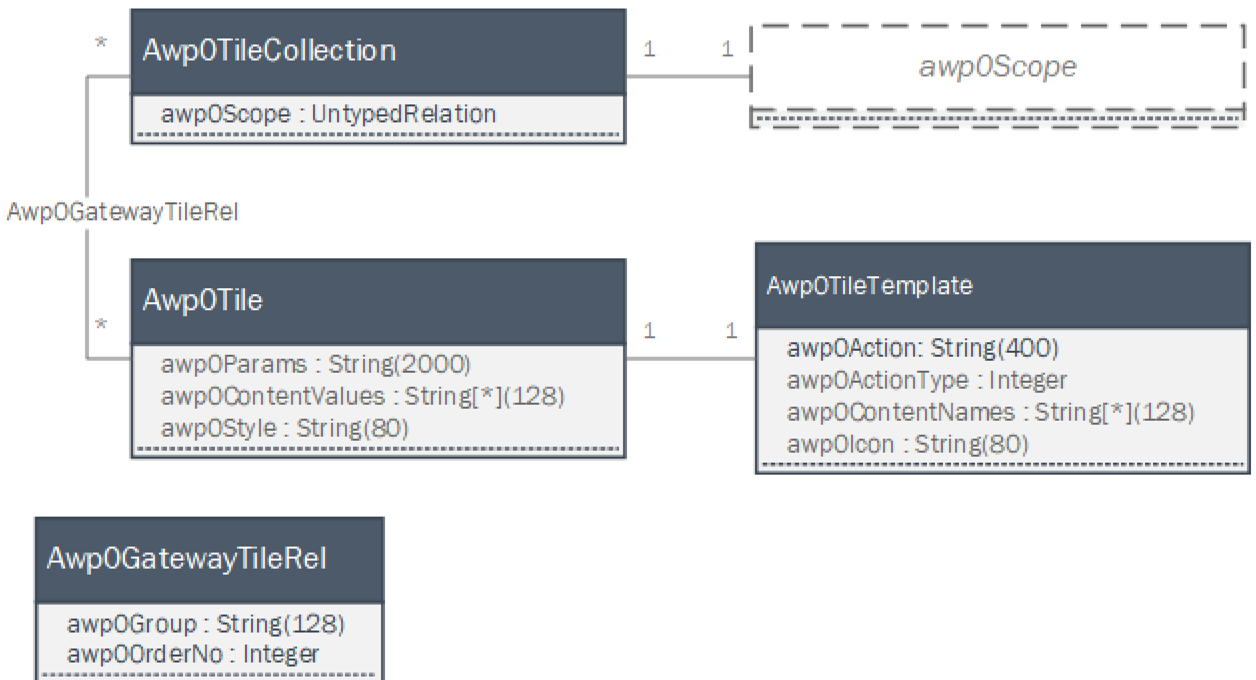
- **Reset a user's home page changes.**
- **Pin (add) a tile to a collection.**
- **Protect a tile from being unpinned.**
- **Create a new tile.**
- **Create a new tile collection.**
- **Create a new type of tile.**

Object model

A tile collection object maintains relations to a number of tile objects using **Awp0GatewayTileRel** relations, and they have a reference property that points to a single scope object. Each tile object is created using a tile template object.

All of these things together specify the behavior of the tile.

© 2025 Siemens



- **Awp0TileCollection**

  This object is a container for the tiles assigned to a given scope. The scope is attached to the untyped **awp0Scope** reference property. The tiles are attached using the untyped **Awp0GatewayTileRel** relation property array.

- **Awp0Tile**

  These objects are the tiles that appear on the home page. They contain any information that is specific to the instance, such as parameters and data for live tiles.

- **Awp0GatewayTileRel** (relation object)

  This relation object links a tile to a collection. Use its properties to determine the tile group and the order in which the tile will appear when in this collection.

- **Awp0TileTemplate**

  A tile template stores the overall configuration for a specific tile type. When you create a new tile instance, you must specify which tile template it is based upon. The templates store information about the tile's action, action type, icon, and content names for live tiles.

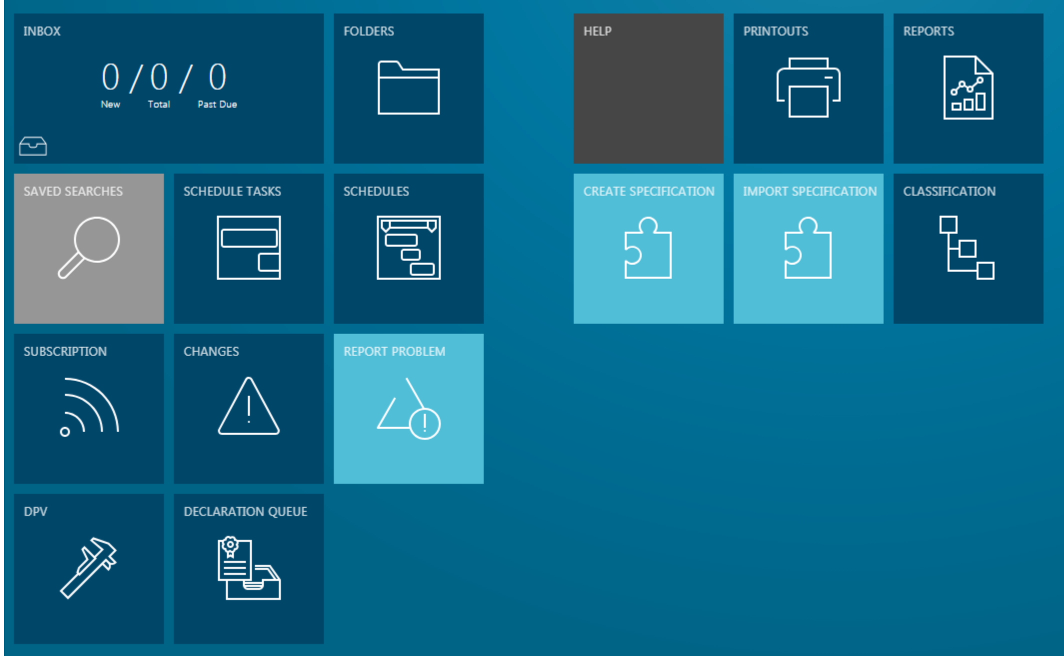
- **awp0Scope** (relation)

  This relation must point to an object that is a valid scope object. This scope determines whether a user can see the tiles in the collection based on their current session context.

© 2025 Siemens

Tile reference

The tile object (**Awp0Tile**) is the actual tile that a user sees on their home page. Each tile must reference a single tile template. Together they make up the total functionality of the tile.



Tasks you can do with the tile

- **Provide arguments for command tile types**
- **Change the tile name**

awp0TileId

This is the tile's unique internal identifier. When creating your own tiles, use your custom prefix

awp0DisplayName

This is the tile name that is displayed to the user for non-object tiles. (Tiles not based on **Awp0PinnedObjectTemplate**) Almost all tiles provided OOTB will use this value as their display name.

However, for a tile that represents a business object (when the user uses the **Pin to Home** command, for example) the tile will instead display the **object_string** property of the pinned object.

© 2025 Siemens

awp0TileTemplate

This is a reference to the tile template that this tile is associated with. This represents the type of tile — its core functionality. The other tile properties allow tiles that share a tile template to have slightly different behavior.

awp0Params

If the referenced tile template is a command (action type = 3), provide your parameters to that command using this property.

Example:

If the tile template's command is to show a *create object panel* to the user, you can specify which object types are displayed on the list. Only the types specified (not their subtypes) are displayed.

- Show all item revision types.

  ```
  cmdArg=ItemRevision
  ```

- Show folder and document types.

  ```
  cmdArg=Folder&Document
  ```

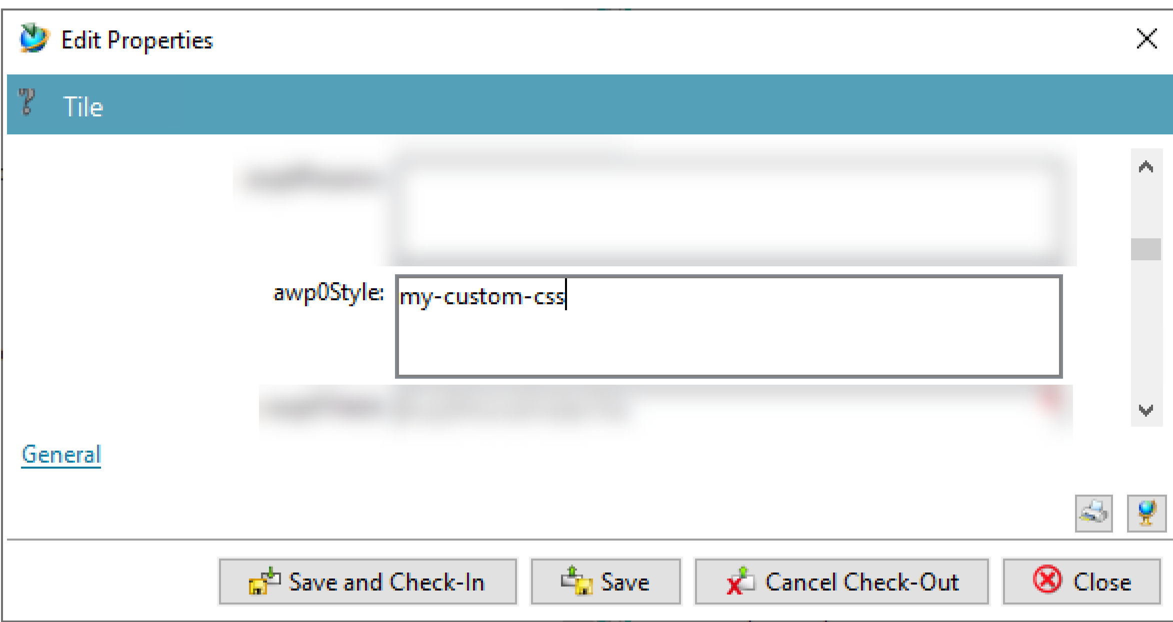
- Show 3 specific dataset types.

  ```
  cmdArg=Text&MSWordx&MSExcel
  ```

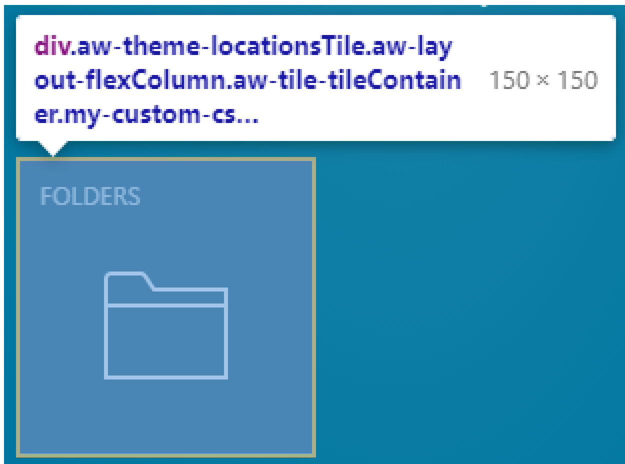
awp0Style

A custom cascading style sheet (CSS) class that is added to the tile's **<div>** class list on the **HOME** page. For example, add **my-custom-css** to the **awp0Style** property.

© 2025 Siemens



That class appears in the HTML code of the UI.




```
▼<div class="aw-theme-locationsTile aw-layout-flexColumn aw-tile-tileContainer my-custom-css aw-tile-smallSize" ng-class="{'aw-tile-tileEdit': tile.editing, 'aw-tile-smallSiz
```


© 2025 Siemens

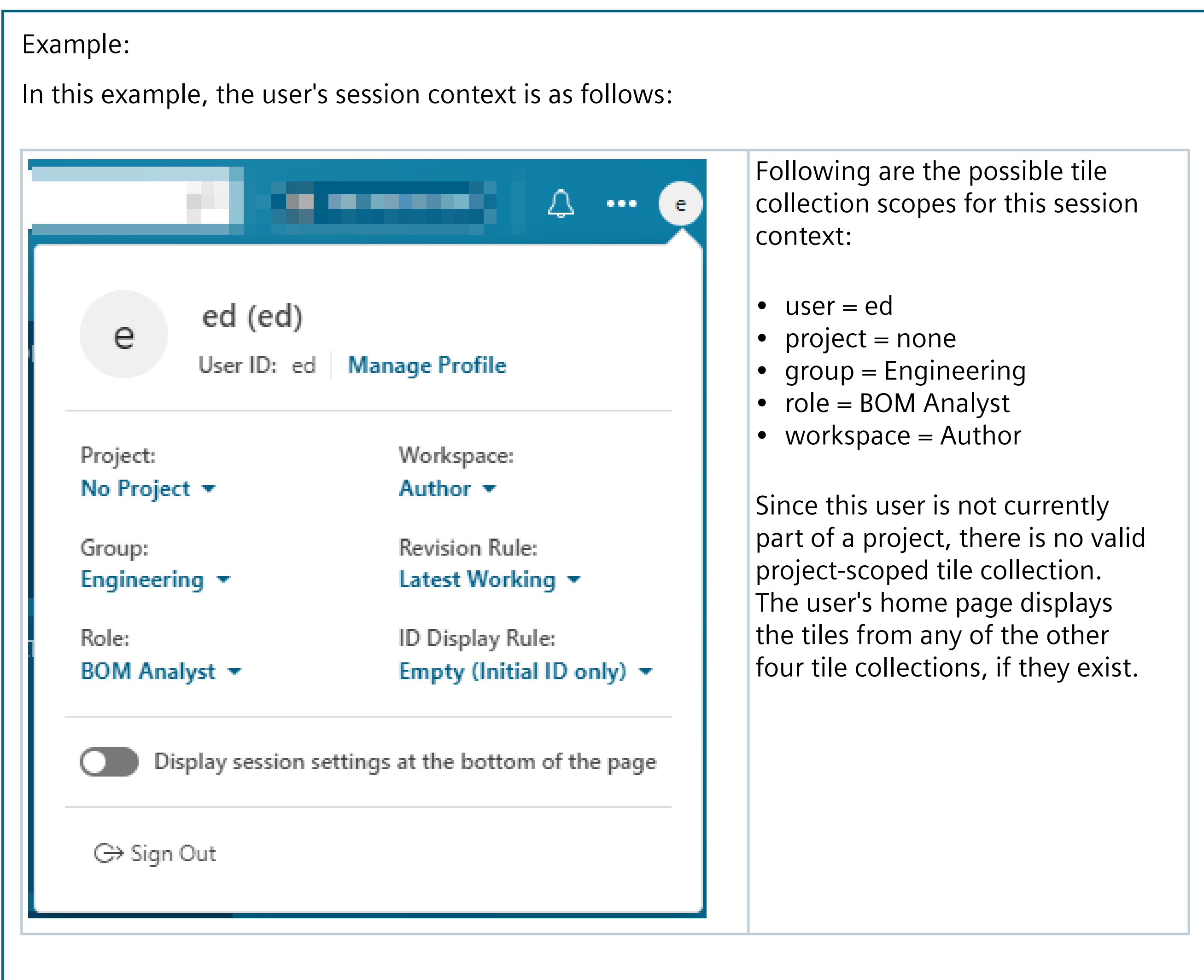
Tile collection reference

You can create a tile collection for a specific scope. You can then relate tiles to the collection so that users whose session context include the appropriate scope can see the tiles. Each user's session context consists of many pieces of information. Of these, the tile collection scopes are concerned with the user and their current group, role, project, and workspace. The user's home page consists of all the tiles from up to five tile collections, one from each scope.

The user's active tile collections have no order of precedence. The tiles from each active tile collection are displayed. If a tile is hidden by an active collection, it is not displayed to the user regardless of how many other active collections contain that tile.

Example:

In this example, the user's session context is as follows:



Following are the possible tile collection scopes for this session context:

- user = ed
- project = none
- group = Engineering
- role = BOM Analyst
- workspace = Author

Since this user is not currently part of a project, there is no valid project-scoped tile collection. The user's home page displays the tiles from any of the other four tile collections, if they exist.

© 2025 Siemens

> Note:
>
> If the user has only a single workspace available to them based on their current group and role, the UI does not display the **Workspace** option in the context control panel, since they have no choice. Regardless, every user is always part of a workspace, even if they don't know about it.

Tasks you can do with the tile collection

- **Reset a user's home page changes**

  All of the user's changes to their home page are stored in their user-scoped tile collection. To reset their home page, delete their tile collection.

  User tile collections are automatically named as follows: **username-TileCollection**.

  > Example:
  >
  > The user **ed** logs on and makes a change to their home page. Active Workspace creates an **awp0TileCollection** object named **ed-TileCollection**, and the **awp0Scope** property references ed's user object.

- **Pin (add) a tile to a collection**

- **Assign a scope to a collection**

awp0Scope — Scope

In this property, you must add a reference to an object. The type must be one of the following:

User (POM_system_class)

Role (POM_system_class)

Group (POM_system_class)

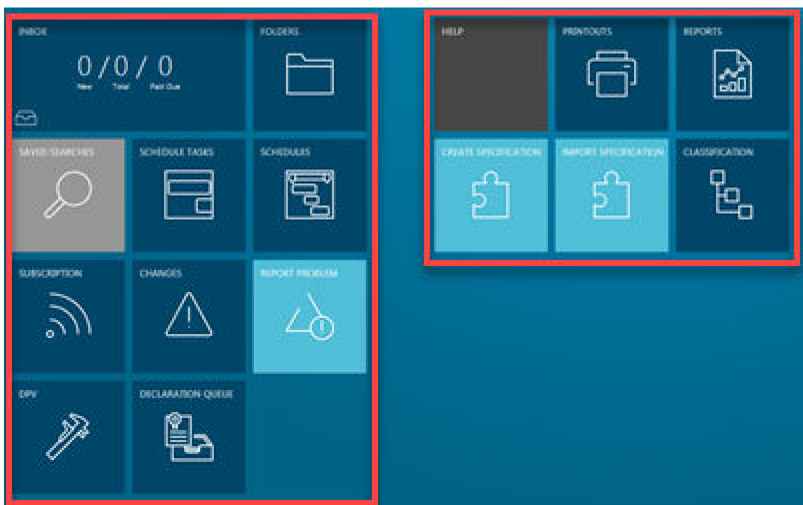
TC_Project (POM_application_object)

Awp0Workspace (Fnd0UIConfigurationObject)

Tile relation reference

The gateway tile relation (**Awp0GatewayTileRel**) is the relation type that attaches a tile to a tile collection. It includes the properties used for determining the tile's layout in this specific collection.

© 2025 Siemens



Tasks you can do with the tile relation

- **Determine how tiles are grouped**
- **Determine the order of tiles in a group**
- **Choose the tile size**
- **Hide a tile**
- **Protect a tile from being unpinned**

awp0Group — Tile Group

You can select the tile group in which this tile appears. There are a few groups provided, but you can type in your own.

- Main
- Quick Links
- Favorites

awp0OrderNo — Order Number

The tiles are ordered within the group according to this integer. Leave a gap between the numbers to make it easier to insert and rearrange the tiles later on.

The lowest order number within each group determines the order of the groups themselves.

© 2025 Siemens

awp0Size — Tile Size

This integer allows you to choose the size of the tile. You must choose from the list of available sizes as defined in the tile template.

If you choose	By entering	The tile is
small	0	1 x 1
tall	1	1 x 2
wide	2	2 x 1
large	3	2 x 2

awp0Removed — Hidden

When you set this Boolean property to **true**, the tile does not appear on the user's home page. This can be done to override a tile that is included in a collection from another scope. For example, consider that the workspace-scoped tile collection contains a certain tile, and the user's current group-scoped tile collection contains that same tile, but it is hidden. In such a case, the user will not see that tile. If they change to a different group and this group's tile collection does *not* hide the tile, the user will see the tile.

awp0Protected — Protected

If you set this Boolean property to **true**, the tile cannot be unpinned through the UI.

Tile template reference

Before you can create a home page tile, you must have an appropriate tile template configured.

Once this is available, you can use it to **create a new tile**.

Tasks you can do with the tile template

- **Determine the tile's action type**
- **Choose the tile icon**
- **Choose the tile theme**
- **List available tile sizes**

© 2025 Siemens

awp0TemplateId — Template ID

The template ID of the tile.

awp0Icon — Icon

awp0IconSource — Icon Source

awp0ActionType — Action Type

Tile templates are used to provide actions to tiles. Following are the available values to use in the **awp0ActionType** property.

awp0ActionType	awp0Action	On click
0 — Default	Active Workspace history token	Go to the Active Workspace history token.
1 — External link	URL	Open the URL in a new tab.
(2 — Static resource)		*This action style is no longer supported.* Instead, store your file in the Teamcenter database and pin it to the home page.
3 — Command	**page;cmdId=commandId**	Go to the page and then run the command. Command arguments are provided by the tile.

Example:

If you want the tiles created from this template to send the user to their home folder (**Awp0ShowHomeFolder**) and then run the *show create object* command (**Awp0ShowCreateObject**):

```
awp0actionType=3
awp0Action=Awp0ShowHomeFolder;cmdId=Awp0ShowCreateObject;
```


© 2025 Siemens

The types of objects that can be created are defined on the tile.

awp0ThemeIndex — Theme Index

A tile's theme index is categorized by its function, which also determines the tile's color scheme.

This theme index	Is for
0	Admin locations
1	Pinned objects
2	Locations
3	Commands and actions
4	Saved searches

awp0Sizes — Tile Supported Sizes

There is an array of size options available for the tile.

If you choose	By entering	The tile is
small	0	1 x 1
tall	1	1 x 2
wide	2	2 x 1
large	3	2 x 2

Create a new tile using the rich client

To create a new tile using the rich client, follow these general steps:

1. **Create a new tile template**. (optional)

2. **Create a new tile based on a tile template**.

3. **Relate the new tile to a tile collection**.

Create a new tile template

1. Create a new business object of the type Tile Template (**Awp0TileTemplate**).

2. Provide the required properties.

© 2025 Siemens

- **awp0ActionType**: Specify the action type.
- **awp0Icon**: Choose an icon.
- **awp0Action**: Specify an action. (optional, based on action type)
- **awp0ThemeIndex**: Choose a theme index.
- **awp0Sizes**: Pick a tile size.

Create a new tile based on a tile template

1. Search for and copy a *tile template* (**Awp0TileTemplate**) object to the rich client clipboard.

2. Create a new business object of the type Tile (**Awp0Tile**).

3. Provide the required properties.

   - **awp0TileId**: Provide a unique tile ID.
   - **awp0DisplayName**: Provide the tile's display name.
   - **awp0TileTemplate**: Paste the tile template from the clipboard.
   - **awp0Params**: (optional) Provide parameters if the tile template is *action type* 3, *command*.
   - **awp0Style**: (optional) Provide the name of a cascading style sheet (CSS) class to assign to the tile.

Relate the new tile to a tile collection

1. Search for and copy a *tile* (**Awp0Tile**) object to the rich client clipboard.

2. Search for a *tile collection* (**Awp0TileCollection**) object.

3. Paste the *tile* onto the *tile collection* using the **Gateway Tile** (**Awp0GateWayTileRel**) relation.

4. Provide the properties on the relation.

   - **awp0Group**: Provide the name of the tile group.
   - **awp0OrderNo**: Provide the order within the tile group.
   - **awp0Size**: Pick the size from the available sizes.
   - **awp0Removed**: (optional) Set this to **true** to hide the tile.
   - **awp0Protected**: (optional) Set this to **true** to prevent the tile from being unpinned.

Dynamic compound properties

Learn about dynamic compound properties

What are dynamic compound properties?

Compound properties are properties that are not defined on the selected object, called the primary object, but instead defined on a related object, called the secondary object, or on a relation between the objects.

© 2025 Siemens

- Traditional compound properties are static, defined in the Business Modeler IDE, and require a schema change and deploy.
- *Dynamic* compound properties are defined using XML files, and are created and modified quickly and easily.

What are the benefits?

You can:

- Create and modify them with no deployment or downtime required.
- Override the display name of the target property to make column titles unique.
- View, edit, filter, sort, and arrange columns in tables.
- Traverse to **n-levels** and **n-cardinality**, as well as both the **primary-to-secondary** and **secondary-to-primary** directions.
- Experience equal or better performance compared to traditional compound properties.

Where can you use them?

You can use dynamic compound properties in tables. Tables that use *data providers* support dynamic compound properties. If a table has its own data population mechanism, the dynamic compound properties are not automatically supported. In addition to tables, you may use dynamic compound properties in place of single properties in the following places:

- **As a property in an XRT style sheet**, including object set tables, table properties, and name-value properties.
- **As a property in a column configuration**.
- **As a source of an object set table**.

What about the compound properties created using the Business Modeler IDE?

The compound properties created in the Business Modeler IDE are *static* and require a TEM or Deployment Center redeploy because of the change in the data model. *Dynamic* compound properties can be used instead of static compound properties in nearly every case.

> Note:
>
> However, because dynamic compound properties cannot be indexed by **Solr**, they cannot be used for searching, although they will appear in the results.

© 2025 Siemens

What else do I need to know?

- When retrieving a string property, a string is displayed.
- When retrieving a reference property, a hyperlink is displayed.
- When the retrieved property contains multiple values, a comma separated list is displayed.
- When multiple objects match the traversal rule, Active Workspace displays each object's property on a separate row in the table. If the dynamic compound property it is not displayed in a table, such as in a summary view, then only the first object's property is displayed.

Dynamic compound property column behavior

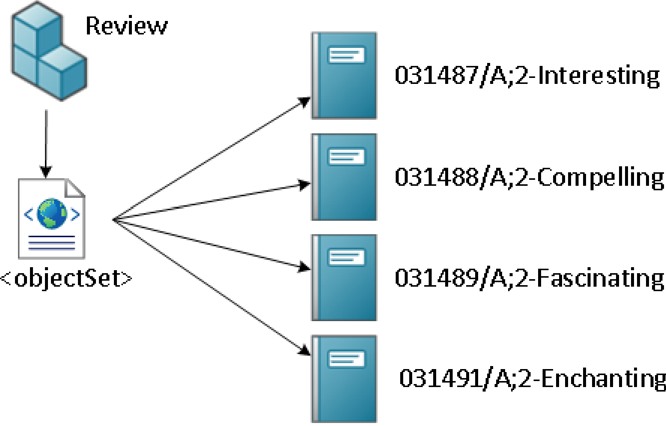
When you use Dynamic Compound Properties (DCP) in a table, they do not repeat information that has already been presented.

> Tip:
>
> The following examples use style sheet object sets, but the same principle applies to declarative tables.

Basic object set behavior

In the following example, there are four document revisions attached to a review object.

An object set lists all **DocumentRevision** objects attached with the **TC_Attaches** relation to the review object.

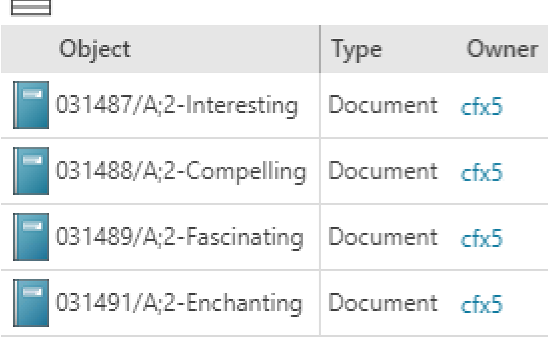


The object set can easily retrieve properties directly from the table source objects, like **object_string**, **object_type**, and **owning_user**.

© 2025 Siemens

```
<section titleKey="tc_xrt_Documents">
    <objectSet source="IMAN_specification.DocumentRevision"
                  sortdirection="ascending"
                  sortby="object_string"
                  defaultdisplay="tableDisplay">
        <tableDisplay>
            <property name="object_string"/>
            <property name="object_type" />
            <property name="owning_user"/>
        </tableDisplay>
        ...
    </objectSet>
</section>
```

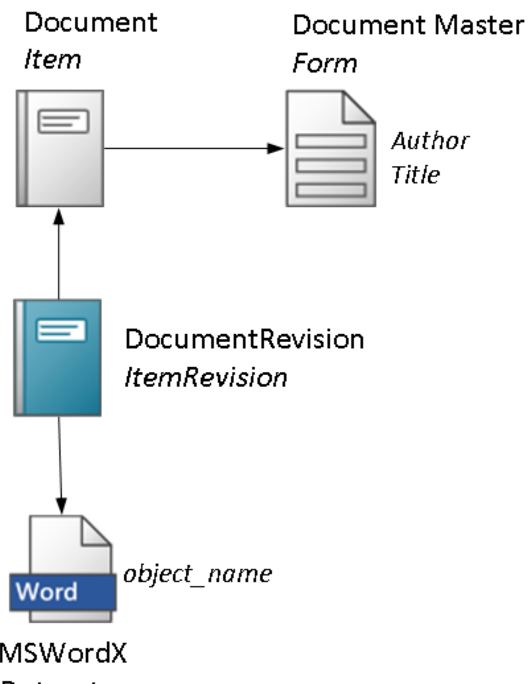
The object set shows the objects and their properties.



Basic object set behavior using DCP

The document revision objects have other objects related to them which contain properties that you may also want to display on your table, such as the author and title from the document item's master form, and the name of the word dataset.

© 2025 Siemens

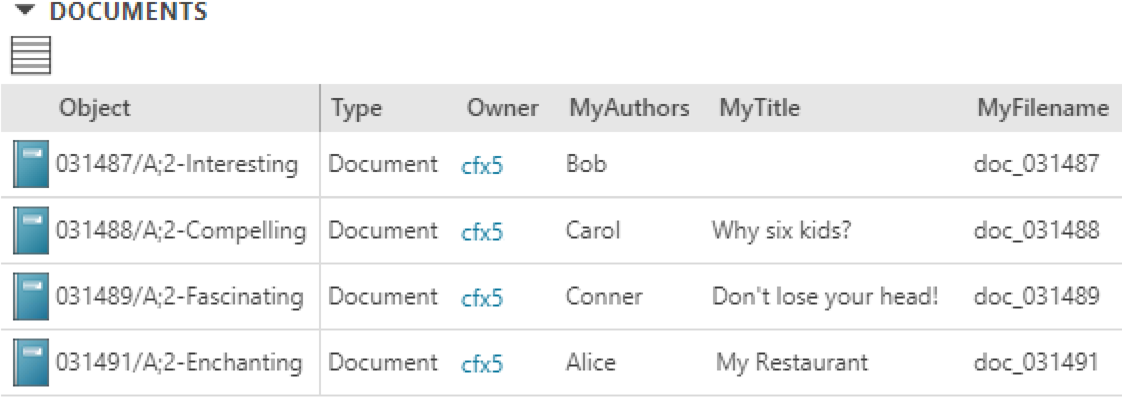


You can use dynamic compound properties to easily retrieve those properties. For example, follow the **item_tag** reference property to the **Document** item, then the **item_master_tag** reference property to the **Document Master** form to retrieve the **Author** property.

```
<tableDisplay>
   <property name="object_string"/>
   <property name="object_type" />
   <property name="owning_user"/>
   <property name="REF(items_tag,Document).REF(item_master_tag,Document Master).Author"
             titleKey="MyAuthors" />
   <property name="REF(items_tag,Document).REF(item_master_tag,Document Master).Title"
             titleKey="MyTitle" />
   <property name="GRM(TC_Attaches,MSWordX).object_name"
             titleKey="MyFilename" />
</tableDisplay>
```

The other properties can be retrieved in the same manner. Notice that the two properties retrieved from the form and the single property retrieved from the dataset are listed together in the same row. This is a convenience feature, but remember that the DCP columns from separate objects (relation paths) don't have any correlation to each other, except that they were located from the same source object.

© 2025 Siemens



DOCUMENTS

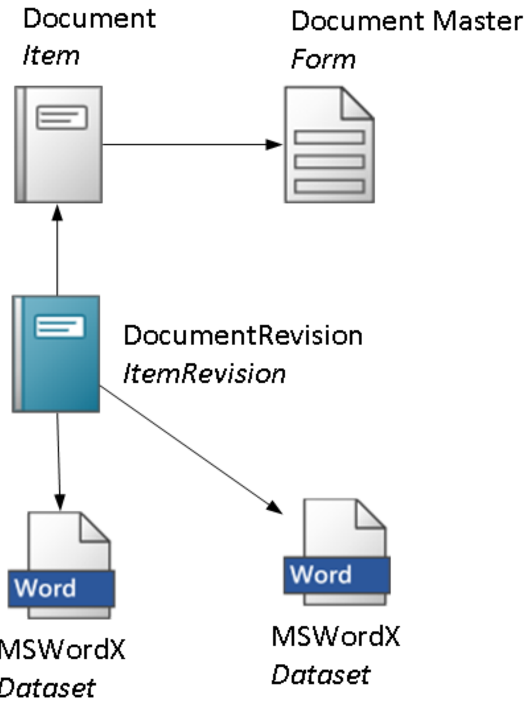
Object	Type	Owner	MyAuthors	MyTitle	MyFilename
031487/A;2-Interesting	Document	cfx5	Bob		doc_031487
031488/A;2-Compelling	Document	cfx5	Carol	Why six kids?	doc_031488
031489/A;2-Fascinating	Document	cfx5	Conner	Don't lose your head!	doc_031489
031491/A;2-Enchanting	Document	cfx5	Alice	My Restaurant	doc_031491

Bob has not yet given his document a title.

Object set behavior using DCP with more than one target

When there is more than a single destination object related to the table source object, the table displays the results, but it does not repeat information from the other DCP columns.

Consider that a document revision may have more than a single word dataset attached.



Suppose that a second dataset, doc_031491-b, is added to the **031491/A;2-Enchanting** document revision. Given the same column setup as the previous example, the table now looks like this.

© 2025 Siemens

DOCUMENTS

Object	Type	Owner	MyAuthors	MyTitle	MyFilename
031487/A;2-Interesting	Document	cfx5	Bob		doc_031487
031488/A;2-Compelling	Document	cfx5	Carol	Why six kids?	doc_031488
031489/A;2-Fascinating	Document	cfx5	Conner	Don't lose your head!	doc_031489
031491/A;2-Enchanting	Document	cfx5	Alice	My Restaurant	doc_031491-b
031491/A;2-Enchanting	Document	cfx5			doc_031491

A fifth row has been added to account for the fifth dataset. However, there are still only 4 objects that are the source of the object set. Notice that the last row of the table repeats the data from the original object set source object it is based from in the first three columns, but does not repeat the information from the fourth and fifth columns, since they are not part of the same relation path.

Using dynamic compound properties with XRT

Use dynamic compound properties (DCP) in your *properties*, *tables*, and *table properties* to display property values from related objects.

Examples of use

Property

```
<property name="REF(att0AttrDefRev,Att0AttributeDefRevision)
                .att0AttrType"/>
```

Object set

```
<objectSet source="GRM(IMAN_specification,UGMASTER)
                   .GRMREL(IMAN_manifestation,NX0TDP)
                   .secondary_object" >
```

Table property

```
<tableProperty source="GRM(IMAN_specification,UGMASTER)"
               name="nx0MaterialsTable">
```

Name-value property

```
<nameValueProperty source="REF(IMAN_specification,AW2_NameValueRevision)"
                   name="aw2_NameValue_Pair">
```

Review the dynamic compound property syntax for more examples of how to use DCP.

Use the titleKey to override a property name

Property names are unique on an object, but when displaying properties from several objects, you could have duplicate property names shown on the table. To avoid user confusion, you may override property names that you retrieve from related objects.

© 2025 Siemens

Example:

You create a table of datasets and you want to display current name of both the dataset and the dataset's parent revision. This will create two columns with the title "**Current Name**".

To avoid confusion, you change the name of the parent revision's **current_name** property using **titleKey**.

```
<property name="current_name"/>
<property name="GRMS2P(IMAN_specification,ItemRevision).current_name"
          titleKey="tc_xrt_dsir_name"
/>
```

Then define a corresponding entry in the **TextServer**.

```
<key id="tc_xrt_dsir_name">Parent's Name</key>
```

Using dynamic compound properties with column configuration

To use dynamic compound properties in column configuration definitions,

- Use the **dynamic compound property syntax** within the **propertyName** attribute.
- Use the **columnName** attribute to override the display name, if desired.

For example, when viewing a dataset in a table, you can traverse the specification relation to find the parent revision's **current_name** property, and then override the **Current Name** column title with something you define in the **TextServer**.

```
<ColumnDef objectType="Dataset"
           propertyName="GRMS2P(IMAN_specification,ItemRevision).current_name"
           columnName="tc_xrt_ds_name"
           width="300"
/>
```

If the **columnName** string is not found in the **TextServer** definitions, it will be presented as-is.

Note:

Use **columnName** only when traversing to other objects. Do not use **columnName** on regular properties.

Dynamic compound property syntax

To use dynamic compound properties, you need to know four things.

© 2025 Siemens

- The traversal method.
- The relation type or reference property type.
- The target object type.
- The target property name.

DCP Syntax

Following is a list of the traversal methods, the syntax required, and an example for each.

REF

Traversal using the reference property. If the reference property is an array property, there will be multiple source objects and therefore multiple property values.

REF(referencePropertyName,typeName).propertyName

Example: Traverse from an **ItemRevision** to an **item** using the **items_tag** typed reference property and return **item_id** property.

```
REF(items_tag,Item).item_id
```

REFBY

Traversal using reference property in reverse. If multiple objects are referenced by the reference property, there will be multiple source objects and therefore multiple property values.

REFBY(referencePropertyName.typeName).propertyName

Example: Traverse from an **Item** to an **ItemRevision** whose **items_tag** refers to it, and then return the **item_reivision_id** property.

```
REFBY(items_tag,ItemRevision).item_revision_id
```

GRM

Traversal using generic relationship management (GRM) rules, primary-to-secondary. If the relation has multiple secondary objects, there will be multiple source objects and therefore multiple property values.

GRM(relationName,typeName).propertyName

Example: Traverse from an **ItemRevision** to a **Dataset** using the **IMAN_specification** relation and retrieve the **object_name** property.

```
GRM(IMAN_specification,Dataset).object_name
```

GRMS2P

Traversal using generic relationship management (GRM) rules, secondary-to-primary. If there are multiple primary objects of the relation, there are multiple source objects and therefore multiple property values.

© 2025 Siemens

GRMS2P(relationName,TypeName).propertyName

Example: Traverse from a **Dataset** to an **ItemRevision** using the **IMAN_specification** relation and retrieve the **item_reivision_id** property.

```
GRMS2P(IMAN_specification,ItemRevision).item_revision_id
```

GRMREL Traversal using generic relationship management (GRM) rules, primary-to-secondary, but stopping on the relation instead of the other object.

GRMREL(relationName, SecondaryObject Type Name).propertyName

Example: Find the related object and display it *as a link*. Compare this to **GRM** above which displays the related object *as a string*.

```
GRMREL(IMAN_specification,Dataset).secondary_object
```

GRMS2PREL Traversal using generic relationship management (GRM) rules, secondary-to-primary, but stopping on the relation instead of the other object.

GRMS2PREL(relationName, PrimaryObject Type Name).propertyName

Example: Find the

```
GRMS2PREL(IMAN_specification,ItemRevision).relation_type
```

Advanced traversal

Multiple-level traversal To perform a multiple-level traversal, concatenate several single-level traversals together separated by periods.

Example: Traverse from an **ItemRevision** up to its **Item**, and from there down to the **ItemMaster** form, and then retrieve the **user_data_1** property.

```
REF(items_tag,Item).REF(item_master_tag,ItemMaster).user_dat
a_1
```

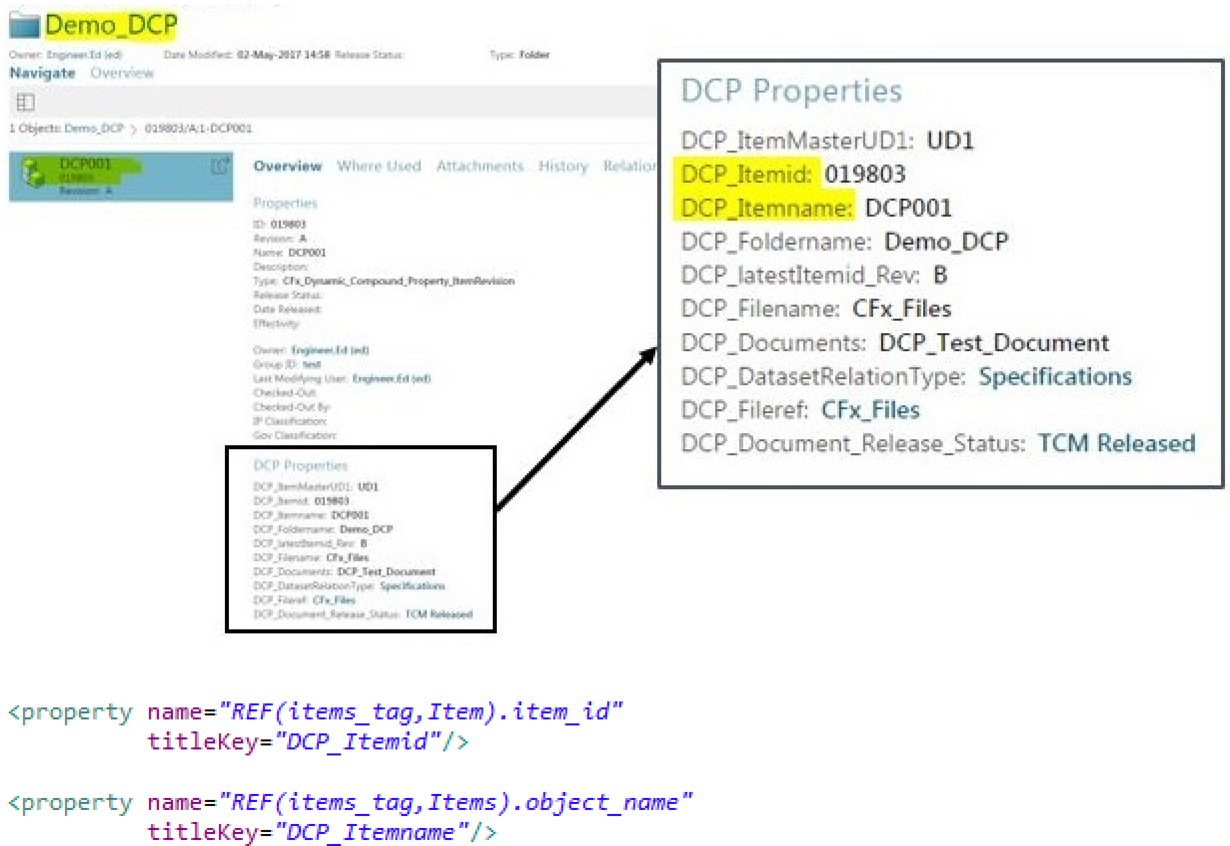
Example REF

Original object: ItemRevision

Traverse to the:

- Parent item, and then retrieve the item's ID.
- Parent item, and then retrieve the item's name.

© 2025 Siemens




```
<property name="REF(items_tag,Item).item_id"
          titleKey="DCP_Itemid"/>

<property name="REF(items_tag,Items).object_name"
          titleKey="DCP_Itemname"/>
```

Original object: VendorPart

Traverse to the:

- Vendor, and then retrieve the supplier's address.
- Vendor, and then retrieve the supplier's contact name.
- Vendor, and then retrieve the supplier's phone number.

Traverse to the:

- Company location, and then retrieve the company's name.

  Change property label to **Supplied From**.
- Company location, and then retrieve the company's street address.
- Company location, and then retrieve the company's city.
- Company location, and then retrieve the company's country.

© 2025 Siemens

Properties

ID: 031495

Vendor Part Name: Power Supply, 300W, PC

Description:
Power Supply with Fan. 120/240V Input, 5V, 1V, 12 V output. 300W max output

Type: Vendor Part

Vendor: 031431-Acme Electronics

Address: 123 S. Main St
San Jose CA 95110

Contact: John Johnson

Phone: 408-555-9600

Supplied From: Penang

Street: 402 Bala Kampur Rd.

City: Penang

Country: MY

```
<property name="REF(vm0vendor_ref,Vendor).supplier_addr"/>
<property name="REF(vm0vendor_ref,Vendor).contact_name"/>
<property name="REF(vm0vendor_ref,Vendor).supplier_phone"/>

<property name="REF(vm0location,CompanyLocation).object_name"
          titleKey="Supplied From"/>
<property name="REF(vm0location,CompanyLocation).street"/>
<property name="REF(vm0location,CompanyLocation).city"/>
<property name="REF(vm0location,CompanyLocation).country"/>
```

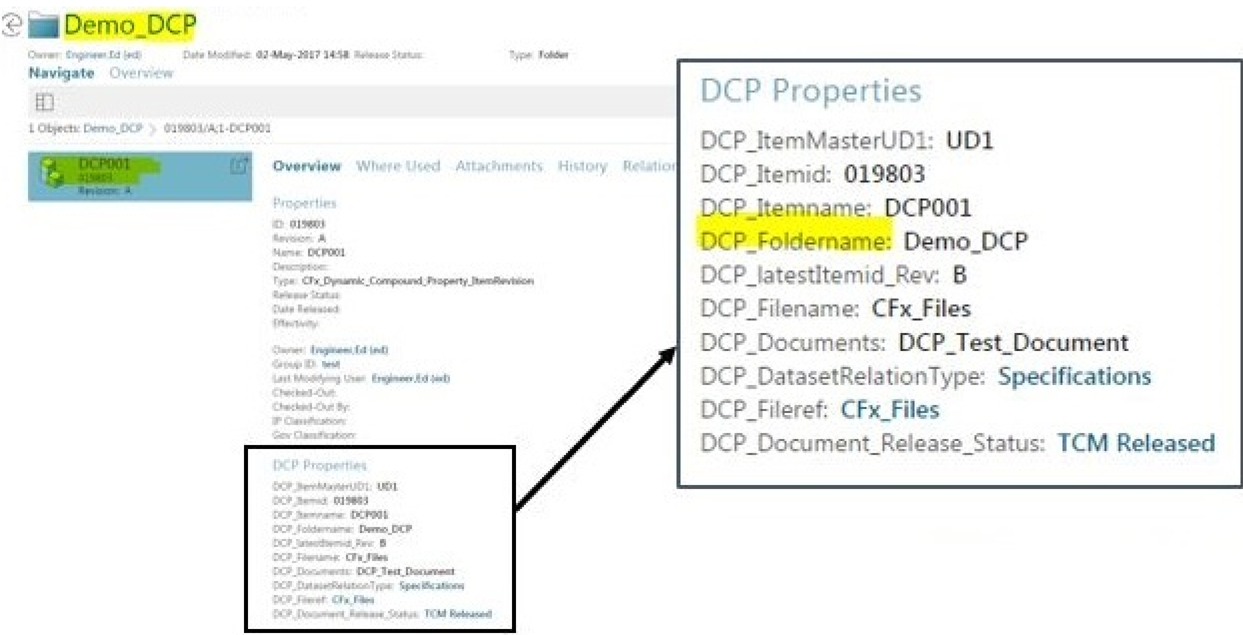
Example REFBY

Original object: ItemRevision

Traverse to:

- A containing folder, and then retrieve the folder's name.

© 2025 Siemens




```
<property name="REFBY(contents,Folder).object_name"
          titleKey="DCP_Foldername"/>
```

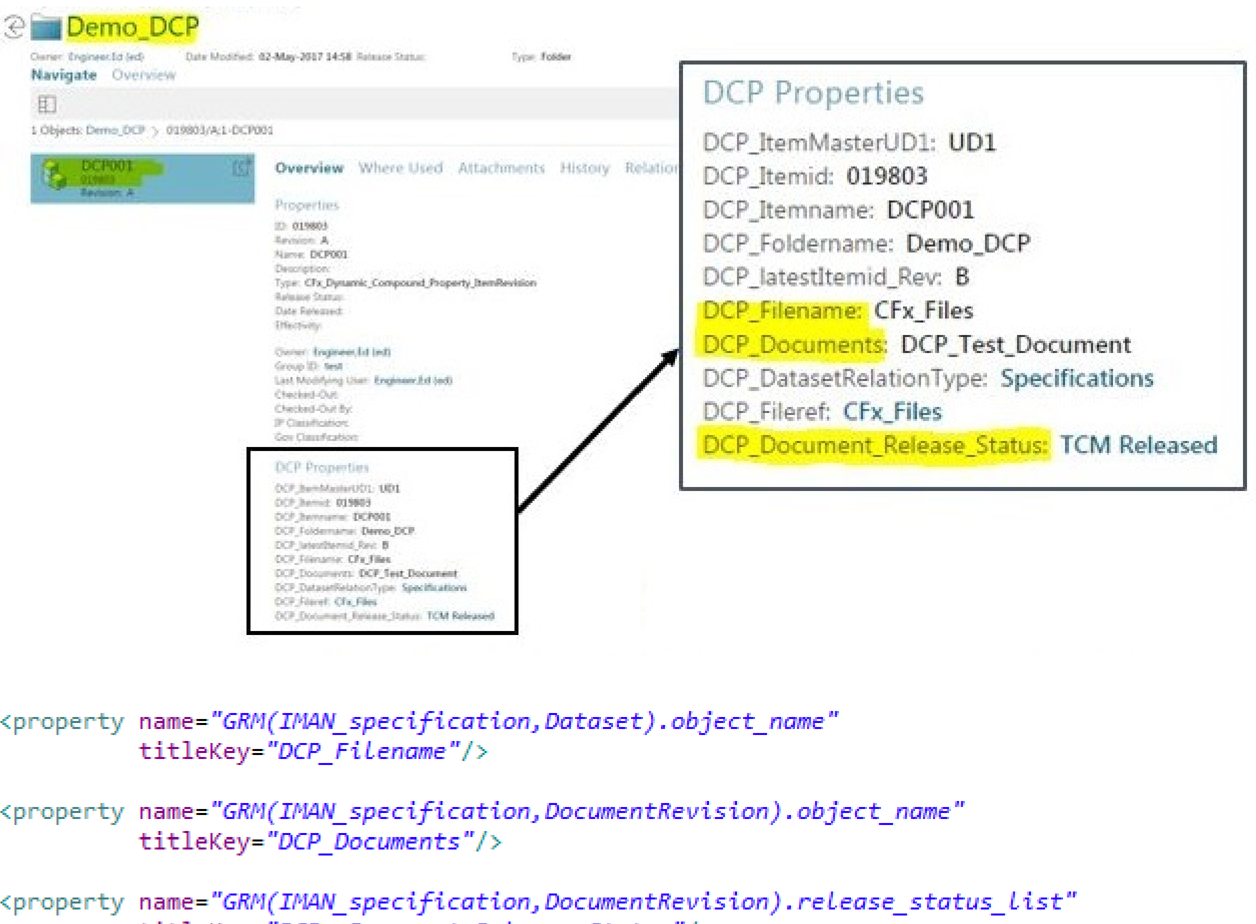
Example GRM

Original object: ItemRevision

Traverse to a:

- Dataset, and then retrieve the dataset's name.
- Document revision, and then retrieve the document revision's name.
- Document revision, and then retrieve the document revision's release status.

© 2025 Siemens




```
<property name="GRM(IMAN_specification,Dataset).object_name"
          titleKey="DCP_Filename"/>

<property name="GRM(IMAN_specification,DocumentRevision).object_name"
          titleKey="DCP_Documents"/>

<property name="GRM(IMAN_specification,DocumentRevision).release_status_list"
          titleKey="DCP__Document_Release_Status"/>
```

Example GRMS2P

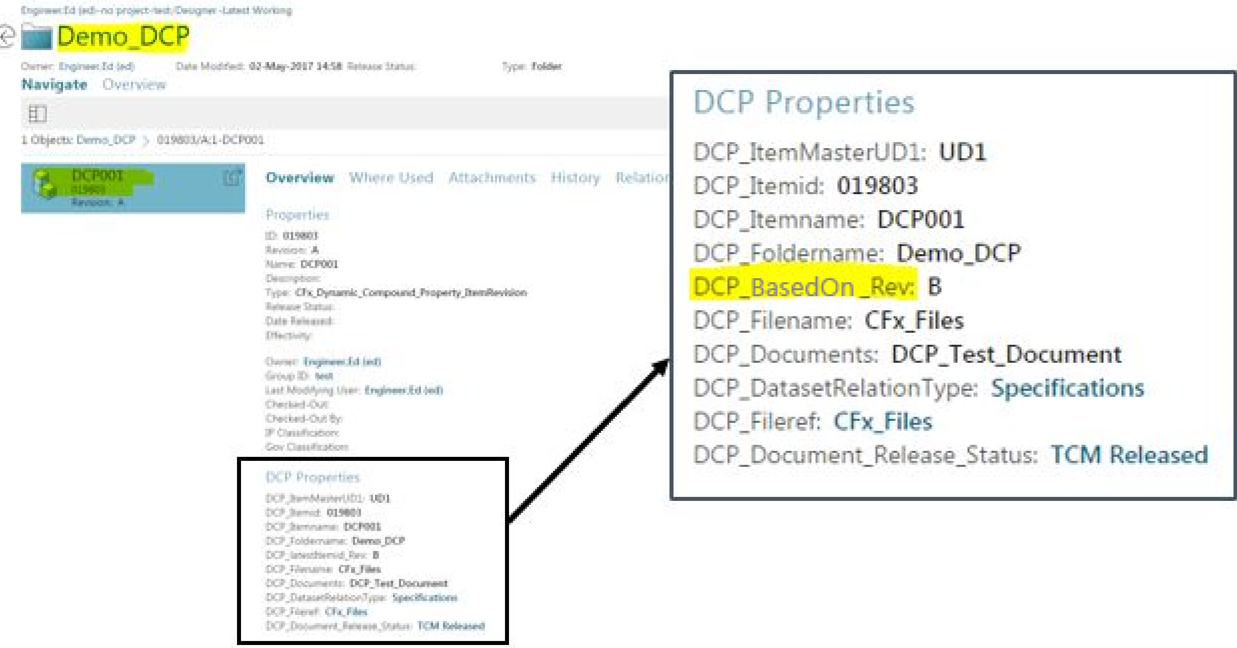
Original object: ItemRevision

Traverse to the:

- Item revision that is based upon this one, and then retrieve that revision's ID.

  Change property label to **DCP_BasedOn_Rev**.

© 2025 Siemens




```
<property name="GRMS2P(IMAN_based_on,ItemRevision).item_revision_id"
          titleKey="DCP_BasedOn_Rev" />
```

Original object: Dataset

Traverse to the:

- Parent item revision, and then retrieve the item ID.
- Parent item revision, and then retrieve the revision's ID.

If the parent item revision is a document revision, then traverse to the:

- Parent document revision, and then retrieve the document's title.
- Parent document revision, and then retrieve the document's author.
- Parent document revision, and then retrieve the document's subject.

© 2025 Siemens

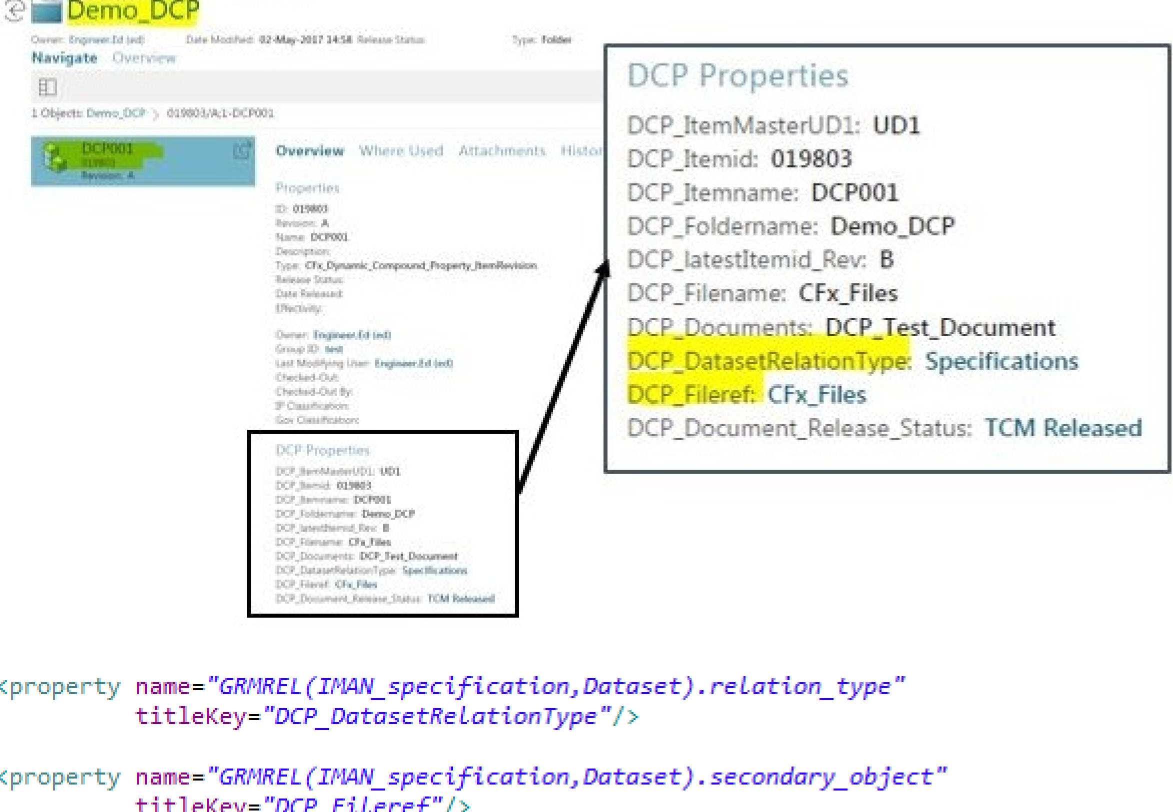
```
<property name="GRMS2P(TC_Attaches,ItemRevision).item_id"/>
<property name="GRMS2P(TC_Attaches,ItemRevision).item_revision_id"/>
<property name="GRMS2P(TC_Attaches,DocumentRevision).DocumentTitle"/>
<property name="GRMS2P(TC_Attaches,DocumentRevision).DocumentAuthor"/>
<property name="GRMS2P(TC_Attaches,DocumentRevision).DocumentSubject"/>
```

Example GRMREL

Original object: ItemRevision

Traverse to the:

- Relation between it and a dataset, and then retrieve a link to the relation type.
- Relation between it and a dataset, and then retrieve the link to the dataset.



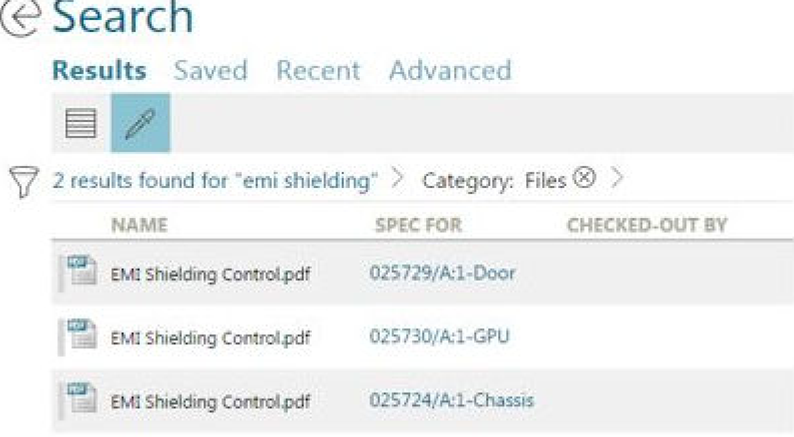

```
<property name="GRMREL(IMAN_specification,Dataset).relation_type"
          titleKey="DCP_DatasetRelationType"/>

<property name="GRMREL(IMAN_specification,Dataset).secondary_object"
          titleKey="DCP_Fileref"/>
```

Example GRMS2PREL

Column configuration

© 2025 Siemens



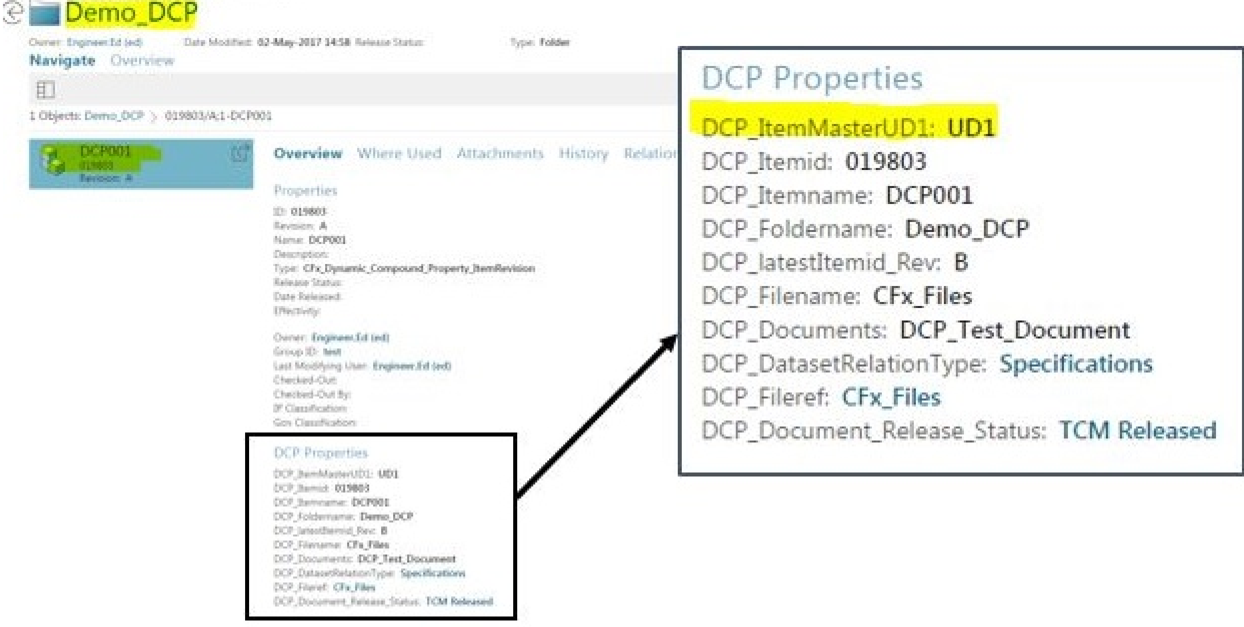

```
<ColumnDef columnName="Spec For" objectType="Dataset"
          propertyName="GRMS2PREL(IMAN_specification,ItemRevision).primary_object"/>
```

Example multilevel traversal

Original object: ItemRevision

Traverse to the:

- Parent item, and then to the
- Item Master form, and then retrieve the **User Data 1** property.




```
<property name="REF(items_tag,Item).REF(item_master_tag,Item Master).user_data_1"
        titleKey="DCP_ItemMasterUD1"/>
```


© 2025 Siemens

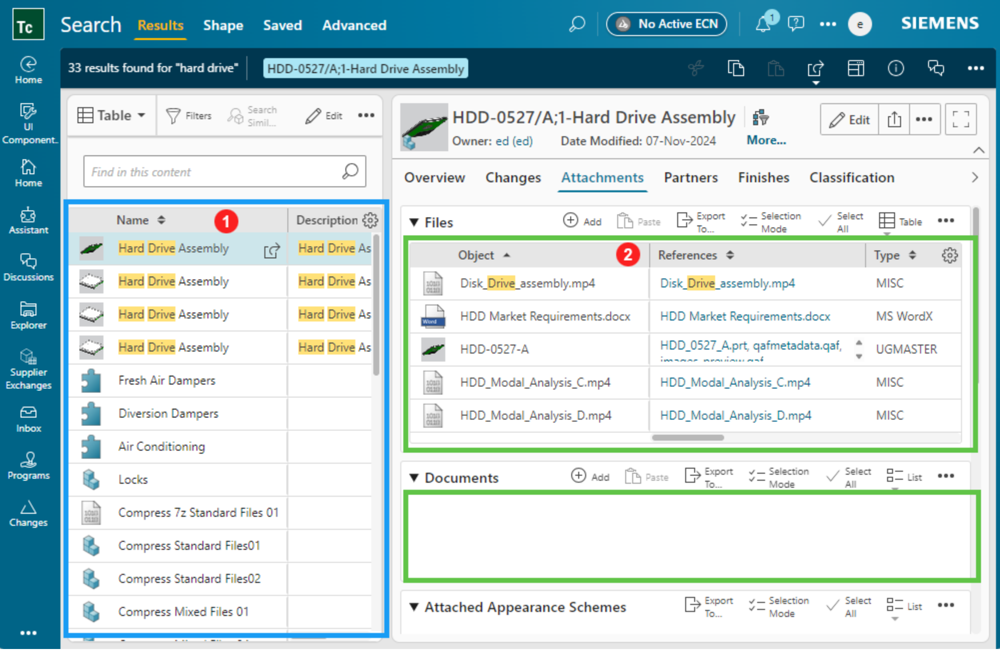
Configuring tables

Two types of tables in Active Workspace

There are two types of tables in Active Workspace.

1. *Declarative tables*, typically in the primary work area.

2. *Object set tables*, exclusively in the secondary work area.



Declarative tables

All tables in the primary work area are part of the declarative page definition, which is detailed in the **view and view model for the page**. There are some locations that are completely declarative (no XRT style sheet) and so even the secondary work area tables are declarative.

In this example, the panel on the left (primary work area) is a **Table** showing the results of the search. You may also see these tables when viewing assemblies or other lists of objects that are the focus of the page.

In declarative tables, you control which columns are available to your users and in which order they appear, by using the **Table Configurator**. You may also use **command line utilities to manage your column configurations**.

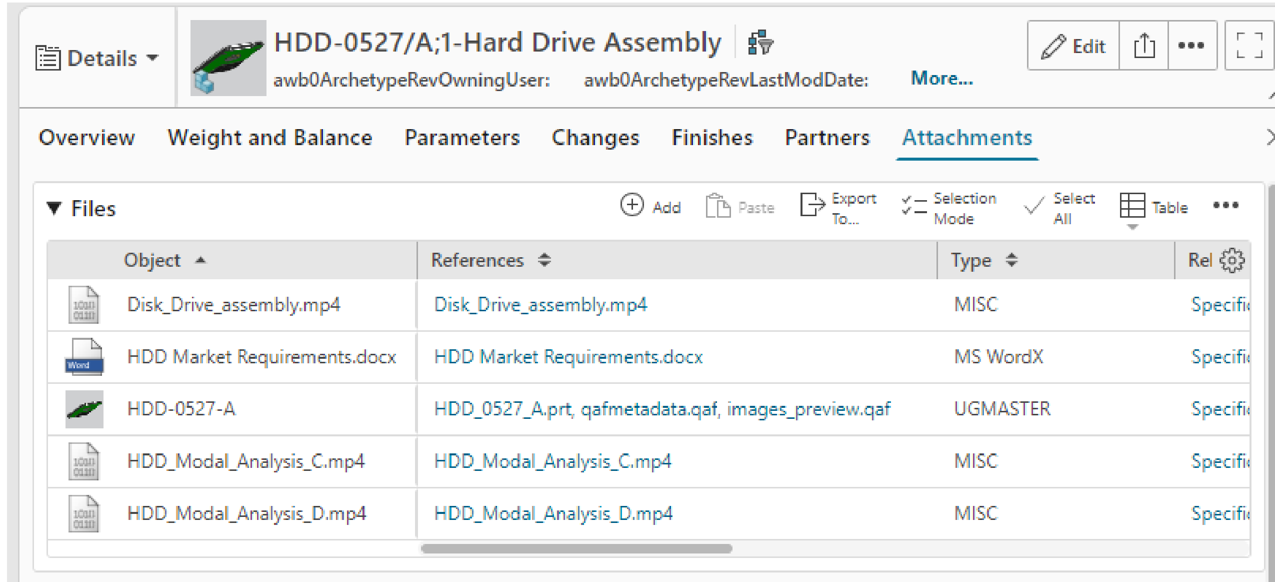
© 2025 Siemens

Style sheet tables

Many of the tables in the secondary work area are part of an *XML Rendering Template* (XRT) style sheet definition, which is detailed by the page's **objectSet** element. However, some locations are fully declarative and do not use a style sheet. Examine the HTML UI elements of the table in your browser if you are unsure of the table type.

Example:

In this example, the table is in the secondary work area and is part of the **Summary** style sheet for the object selected in the primary work area. Part of that summary is a page showing **Attachments** which are displayed in a table in the **Files** section. These types of tables will display properties of (or properties of objects related to) the object selected in the primary work area.



Use the browser's developer tools to inspect the HTML. Notice the classes **aw-walker-objectset** and **aw-xrt-objectSetContent**, these provide clues that this table is an object set table rendered using a style sheet (XRT).

```
▼<div class="sw-section sw-section-content sw-column "> flex  slot
  ▼<div class="aw-walker-objectset">
    ▼<div class="aw-xrt-objectSetContent aw-layout-flexColumn aw-base-scrol
      lPanel" style="max-height: 240px; width: 100%;"> flex
```

Your HTML may differ!

Create a style sheet table's *initial configuration* by defining it directly in the **objectSet** XRT element using **property** tags. However, if the user arranges the table (using the **Arrange** command), a new

© 2025 Siemens

configuration is automatically created to manage the table for that user in that group and role. The objectSet properties are ignored.

If the user uses the **Reset** command, then it will remove all of their user-specific configurations and return to the original objectSet definition.

> Tip:
>
> Use the XRT Editor to edit style sheets directly in the user interface.

Do both types of tables allow filtering?

Yes. Both declarative and style sheet tables are capable of using faceted filtering as long as it's enabled in the table and supported by the data provider.

If you are creating a *custom* declarative table, set **isFilteringEnabled=true** in your table definition to enable filtering. As long as the table's data provider supports it, filtering will be enabled. Get more information from the **UI Pattern Library** on Support Center.

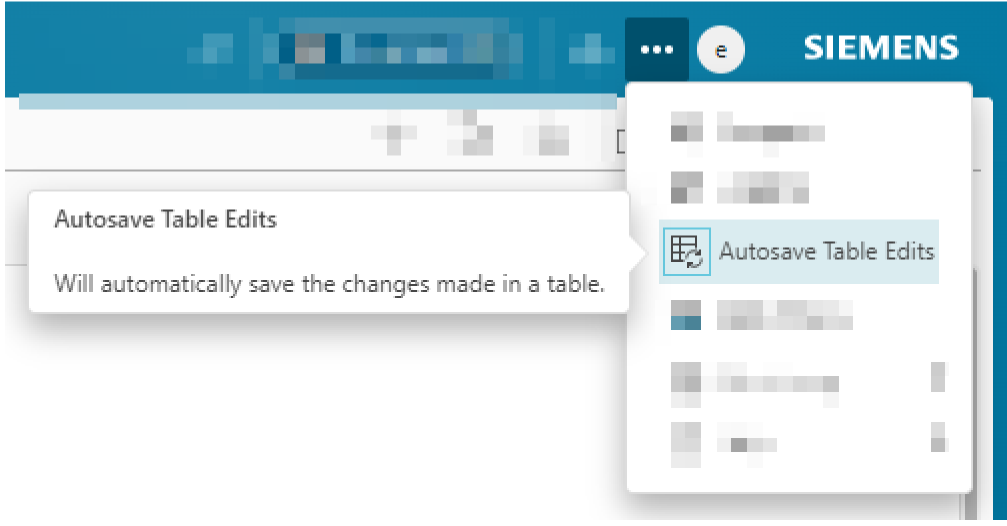
With an object set table, you can turn off filtering by setting **filterable="false"** on the table.

Table properties

Table properties are mentioned here because they look like a table full of many properties, but really they are a single property that contains a table. Creating a new property requires using the BMIDE to make schema modifications to your database.

Configuring the Autosave Tables default setting

Normally, changes to the value of an edited table cell are saved as soon as the user leaves the cell. This behavior is controlled by the **Autosave Tables** setting, which can be changed by the user at any time.



© 2025 Siemens

The **AWC_Autosave** preference controls the default value of this setting. Since this preference has a **User** scope, you may control this behavior at the group, role, or user level.

Disable automatic table cell editing mode

In the table's definition, add the **enablePropEdit** attribute, and set it to **false**.

This disables the user's ability to automatically start editing the values on this table by double-clicking. If the user wants to make changes to values on this table, they must start editing manually by using **Start Edit** .

- With style sheets, this works in **object set** tables, **name value pairs**, and with **table properties.**. For example:

```
<tableProperty name=... enablePropEdit="false">

<objectSet source=... enablePropEdit="false">

<nameValueProperty name=... enablePropEdit="false">
```

- In a declarative table, you can specify either a static boolean value, or a condition:

```
"enablePropEdit": false

"enablePropEdit": "conditions.isNoneContextNotActive"
```

Icons or text in tables

The data for some columns, like release status, can be displayed as either an icon or as text or both. Use the **AWC_TableIndicatorDisplay** user preference to control this display setting.

Siemens Web Framework

Declarative tables are part of the Siemens Web Framework upon which Active Workspace is built.
Refer to the following documentation Digital Engineering Services product on Support Center for more information.

- Read the *aw-splm-table* topic.

- Find the *aw-splm-table* component in the API reference.

© 2025 Siemens

Configuring declarative tables

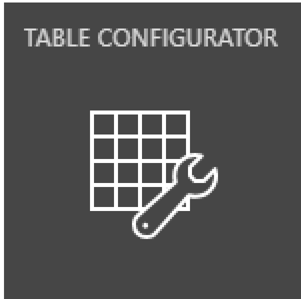
Using the Table Configurator

Use the **Table Configurator** in Active Workspace to configure the columns in your tables. You can create and modify the column configuration definitions for each eligible table in the user interface. To open the **Table Configurator**, you must:

- Be logged in as an administrator.
- Workspace must be either

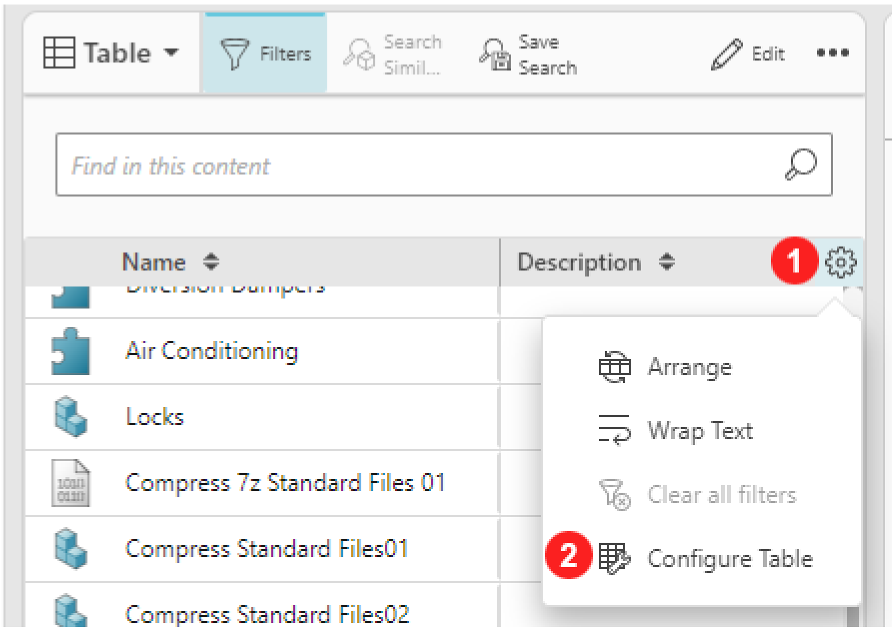
And then, either:

- Click the **Table Configurator** icon in global navigation.
- Open the **TABLE CONFIGURATOR** tile.



- Select **Table Settings** and then **Configure Table** from the column header of a table.

© 2025 Siemens



> Note:
>
> If you use the **Table Settings**, then the **Table Configurator** opens with that table's column configuration selected. If you use the tile, then no configuration is selected.

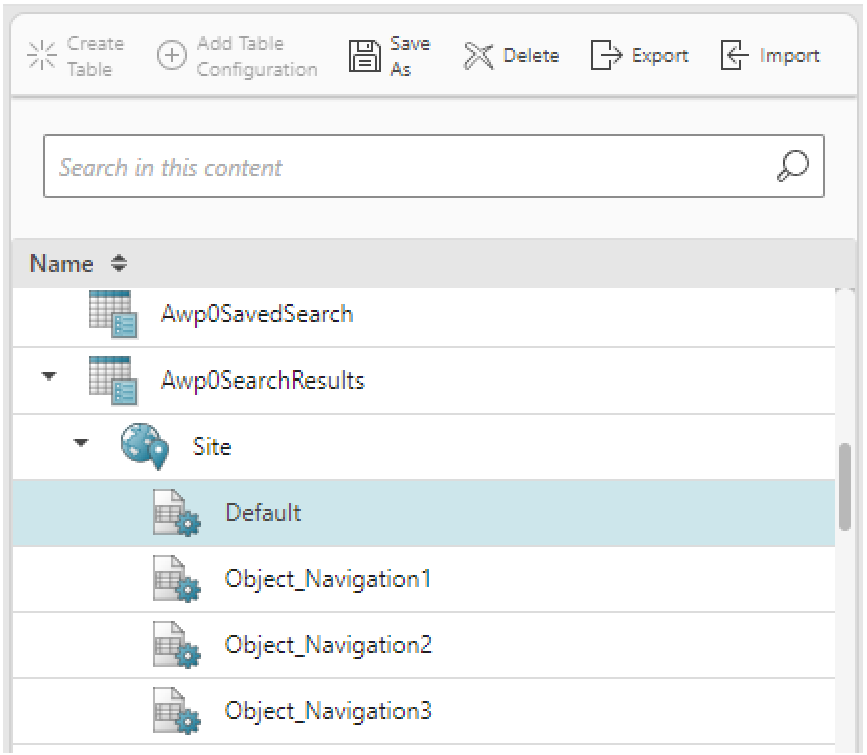
The table configurator interface

The **Table Configurator** page allows you to manage your tables and their column configurations from the user interface.

Primary workarea

This lists all of your tables along with their respective column configurations grouped by their scope.

© 2025 Siemens

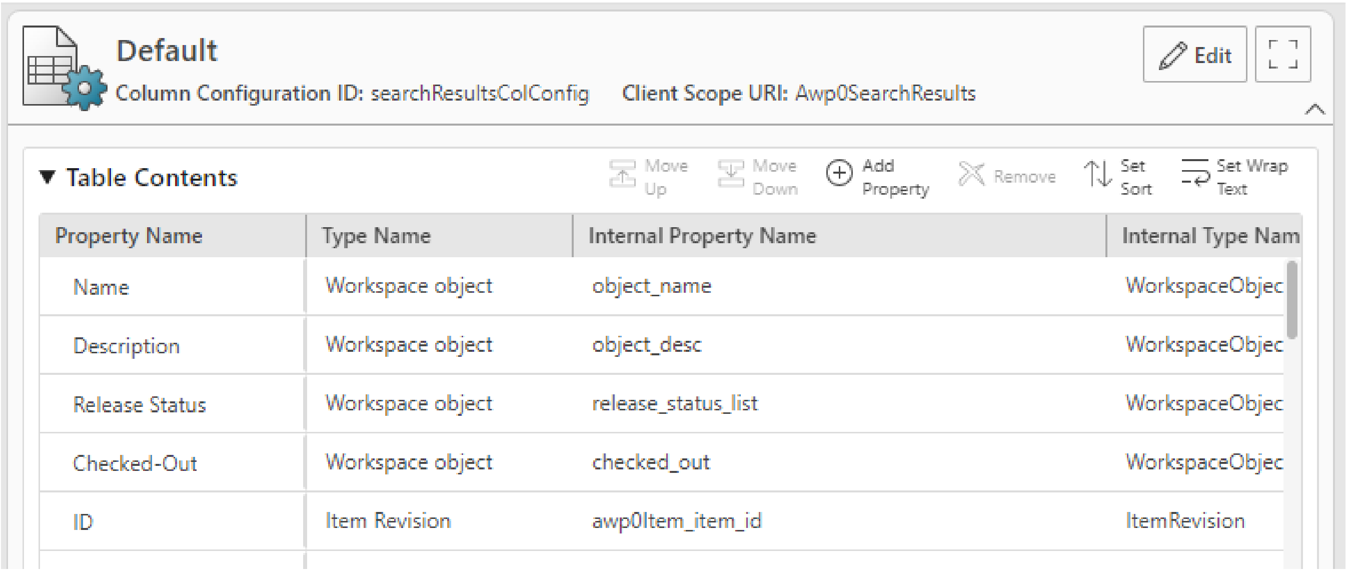


Use	to	You must
Create Table	Create a new table for use with a custom page.	Have nothing selected in the tree.
Add Table Configuration	Create a new column configuration for that table.	Select an existing table.
Save As	Make a copy of an existing column configuration.	Select an existing configuration.
Delete	Remove a copy of an existing column configuration.	Select an existing configuration.
Export	Export existing column configurations to an XML file.	Export all configurations by scope, or select a single configuration to export just that one.
Import	Import column configurations from an XML file.	Choose to **override, merge, or skip**.

Secondary workarea

This shows the currently selected column configuration and allows you to make changes to it.

© 2025 Siemens



Use	to	You must
Move Up	Modify the property order.	Select one or more existing properties.
Move Down	Modify the property order.	Select one or more existing properties.
Add Property	Locate and add a property to the table. Properties may be located directly on the target object type or be a dynamic compound property.	-
Remove	Remove the property from the table.	Select one or more existing properties.
Set Sort	Choose a property and direction by which the table will be sorted.[1]	-
Set Wrap Text	Choose if the table contents will wrap within the cells of the table. This is a toggle.	-
Edit	Change the table entries like **Pixel Width**, **Displayed Columns**, **Column Filtering**, and so on.	**Save** or **Cancel** your edits when you are done.

Create a table configuration

Create a new table to use in your custom pages.

1 You can turn off sorting for the table by clearing the **Sort By** field.

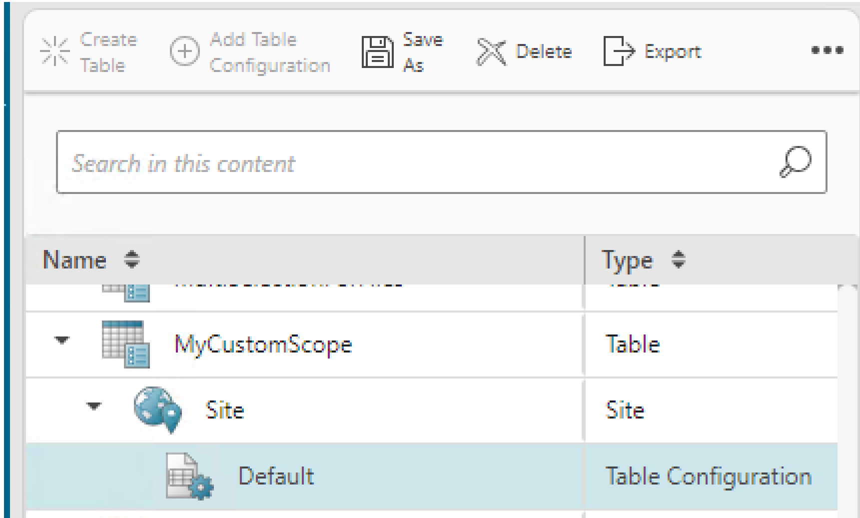
© 2025 Siemens

Procedure

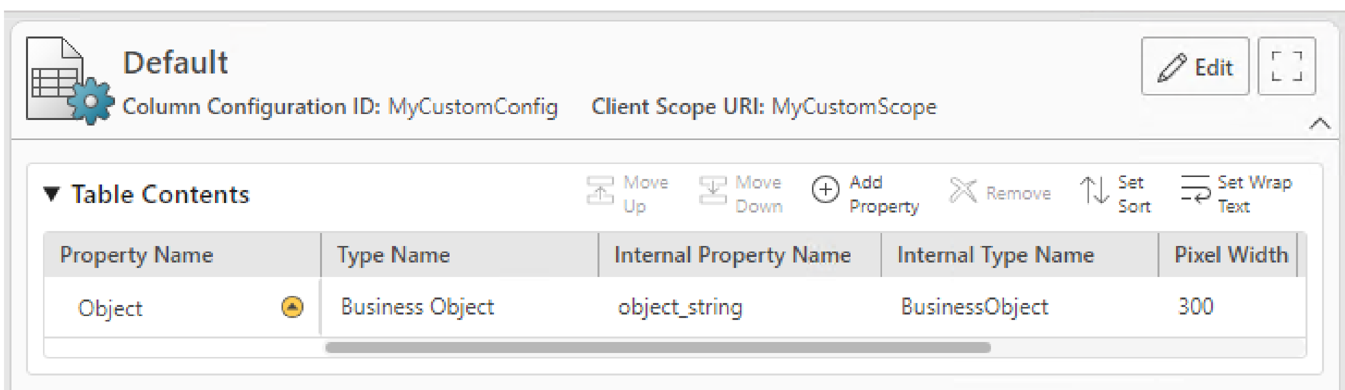
1. Open the **Table Configurator** and ensure nothing is selected.
2. Click **Create Table** ⁘.
3. Enter a unique **Client Scope URI** like **MyCustomScope**.
4. Enter a unique **Column Configuration ID** like **MyCustomConfig**.
5. Click **Create**.

Results

Your new table is displayed in the primary workarea.



Your new table configuration is selected and the default definition is displayed in the secondary workarea.



© 2025 Siemens

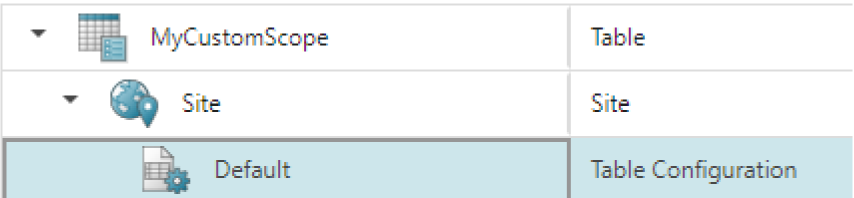
Save existing table configuration to a new scope

Instead of creating a new table configuration, save time by saving an existing table configuration to a new scope.

Procedure

1. Open the **Table Configurator** and select a **Table Configuration** you wish to copy.

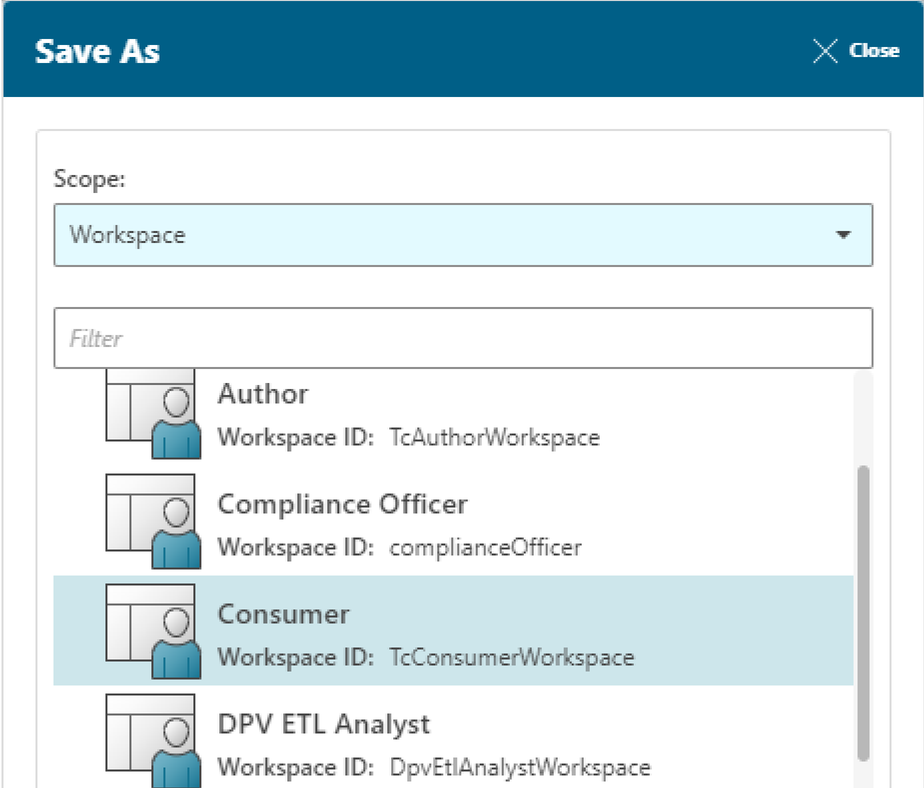
   In this example, the **Default** site configuration of the **MyCustomScope** table is selected.

2. Click **Save As** .

3. Choose a **Scope** and click **Save**.

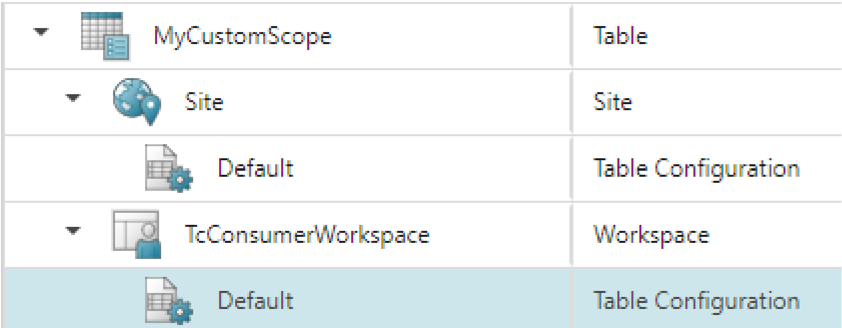
   In this example, the **Workspace** named **Consumer** is chosen.

© 2025 Siemens

Results

The new configuration is created. Modify the new table configuration to control which properties are displayed to users in the **Consumer** workspace.



Modify a table configuration

Modify an existing table's columns and settings.

Procedure

1. Open the **Table Configurator** from an existing table using **Configure Table**.

   Alternately, if you already have **Table Configurator** open, you can select a column configuration from the list.

2. Click **Edit**.

3. Click **Add Property**.

4. Choose **Basic**.

5. Pick **Item Revision** from the **Type** dropdown list.

6. Pick **Description** and **Date Modified** from the **Properties** list.

7. Click **Add**.

   Two new properties are added to your table.

8. Click **Add Property**.

9. Choose **Advanced**.

10. Pick **Item Revision** from the **Type** dropdown list.

© 2025 Siemens

11. Enter **REF(items_tag,Item).item_id** in the **Dynamic Compound Property** field.

12. Click **Add**.

    The new dynamic compound property is added to your table.

13. Click **Set Sort** to choose which column is sorted by default.

14. Toggle **Set Wrap Text** to choose if the property values wrap.

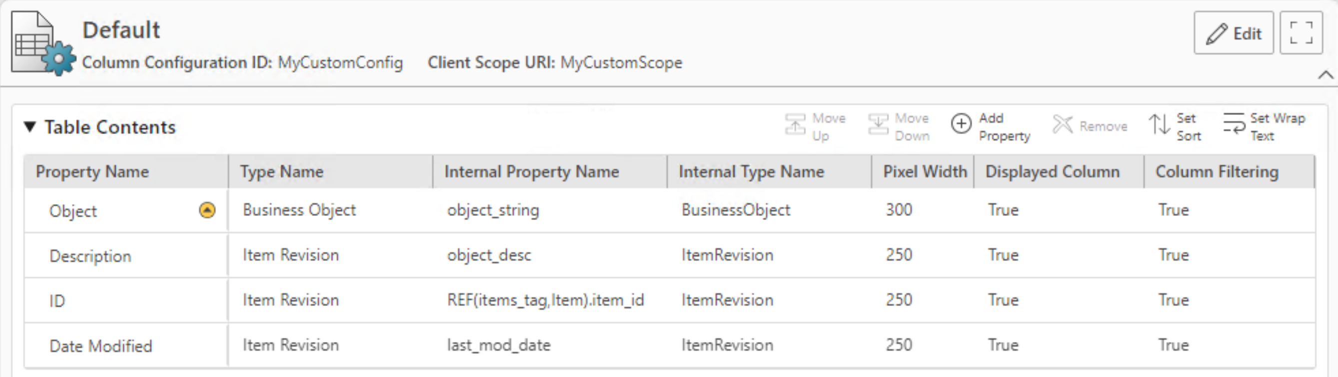
15. Click **Edit** to change various settings for each column.

    Modify the width, visibility, and filtering options.

16. Click **Save** to keep your changes or **Cancel** to undo your changes.

Results

Your custom table contains the properties you added.



Property Name	Type Name	Internal Property Name	Internal Type Name	Pixel Width	Displayed Column	Column Filtering
Object	Business Object	object_string	BusinessObject	300	True	True
Description	Item Revision	object_desc	ItemRevision	250	True	True
ID	Item Revision	REF(items_tag,Item).item_id	ItemRevision	250	True	True
Date Modified	Item Revision	last_mod_date	ItemRevision	250	True	True

Column configuration CLI reference

Using the command line to manage your column configurations

How do I manage column configuration using the command line?

There are two Teamcenter command-line utilities, **export_uiconfig** and **import_uiconfig**, that you can work with to define column configurations. You can specify normal properties and dynamic compound properties in your configuration.

How does Active Workspace determine which column configuration to use?

Column configurations have an order of precedence based on their scope. A scope is an instance of the column configuration that allows each user, group, or role to have their own personalized layout of a

© 2025 Siemens

table. Several tables may share a column configuration, and the changes made to one table are reflected in the others.

GroupMember

This scope is created when a user manually modifies their column configuration. It overrides all other scopes for that column configuration. This scope cannot be exported and can only be removed by an administrator using **delete_uiconfig -column_config_id** to remove *all* scopes for that configuration.

Workspace

This scope takes precedence over all others, except **GroupMember**.

Role

Takes effect for a specific role.

Group

Takes effect for a specific group.

Site

This is the default column configuration for the site. This scope only takes effect if there are no other valid scopes assigned to the user.

You can provide several column configuration arrangements for your users to choose from

By giving the column configuration a name (using **columnConfigName**), your users see that configuration as an option in their **Arrange** panel for the table. This allows you to create several column configuration arrangements for each **ID** and the user can choose between them. Users can also create their own custom column configuration arrangements for each table.

> Caution:
>
> While you can associate many column configuration names with a single column configuration ID, there must only be a single column configuration ID associated with a client scope or object set URI.

Create or modify a column configuration

To modify an existing column configuration or to create a new definition, you must define it in an XML file and then import it. Creating the XML definition is easiest if you export the existing column configurations for reference, copy one, and then modify it for import.

> Caution:
>
> Siemens Digital Industries Software recommends not moving or replacing the first column in a tree display. The object icon does not move from the first column.

© 2025 Siemens

It is recommended to export and save a copy of the default column configuration for reference before making changes.

1. Use the export_uiconfig utility to generate an XML file containing the column configuration used by Active Workspace.

2. Examine the column configuration definitions in the exported XML file. The table definitions shown in the file depend on the features you have installed. For example, if you have the Requirements Management feature installed, the generated file contains table definitions for this feature.

3. Use the exported XML file as reference to create the column configuration for your Active Workspace table.

   Save the original export for reference and backup.

4. Use the import_uiconfig to import your new XML file.

   You determine the scope to which your new definitions apply when you import the XML file.

> Tip:
>
> When importing or merging your column configuration, remember that any user that has customized their table arrangement has their own **GroupMember** column configurations and they will not see your new configurations until they use the **Arrange** panel ⚙ to choose a new arrangement. You may delete all configurations for your target scope before you import by using the delete_uiconfig utility.

Your column configuration definition gives you the following benefits. You may:

- **Determine which properties are available to the table.**

  Provide all possible properties you want the user to have access to. This is an exhaustive list. There is no way for the user to add properties that aren't defined in the column configuration.

- **Choose the default order of the columns**

  The user may make changes to your order to suit their viewing style. This creates a **GroupMember** column configuration definition for that user.

- **Allow certain columns to be filterable**

  A filterable column gives the user the ability to reduce the number of rows they see based on their criterion. They cannot filter on a column that isn't defined as being filterable. Dates and numbers are automatically detected and a range will be available for the user, but all other data types will be treated as text.

© 2025 Siemens

> Note:
>
> Column filtering is available in the **Home** folder primary work area tables, and in all style sheet tables.

- **Turn certain columns off by default**

  When you set a property to **hidden=true**, that property is available to the user if they want to add it, but it is not displayed by default. While a property remains hidden, it is not retrieved from the database, which improves table rendering time.

Merging a column configuration

Use the import_uiconfig utility with the **-action=merge** option to add new columns to an existing configuration without having to redefine the existing columns. Any columns you specify will be appended to the end of the table. If they already exist in the definition, they will be moved into the new position.

Export and save a copy of the default column configuration for reference before making changes.

If there were an existing column configuration in the database with four columns defined as follows:

```
...
<ColumnConfig sortDirection="Descending" sortBy="1" columnConfigId="sampleColConfig">
    <ColumnDef propertyName="object_name" ... />
    <ColumnDef propertyName="release_status_list" ... />
    <ColumnDef propertyName="fnd0InProcess" ... />
    <ColumnDef propertyName="ics_subclass_name" ... />
</ColumnConfig>
...
```

You could create a new XML definition containing two columns and then merge this into the existing configuration:

```
...
<ColumnConfig sortDirection="Descending" sortBy="1" columnConfigId="sampleColConfig">
    <ColumnDef propertyName="release_status_list" ... />
    <ColumnDef propertyName="object_desc" ... />
</ColumnConfig>
...
```

If you then export this configuration, your final column configuration would look as follows:

```
...
<ColumnConfig sortDirection="Descending" sortBy="1" columnConfigId="sampleColConfig">
    <ColumnDef propertyName="object_name" ... />
    <ColumnDef propertyName="fnd0InProcess" ... />
    <ColumnDef propertyName="ics_subclass_name" ... />
    <ColumnDef propertyName="release_status_list" ... />
    <ColumnDef propertyName="object_desc" ... />
```


© 2025 Siemens

```
</ColumnConfig>
...
```

The first configuration file contained four column definitions and the second contained two. The final configuration only contains five column definitions because the **release_status_list** column was present in both original XML files and is not duplicated, so it only exists once in the final column configuration. Notice how it moved from being the second column to being the fourth.

You can also use the merge option to change the **hidden**, **filterable**, or **width** options for a column, but because of the column re-ordering, you might consider importing without using the **-action=merge** option so you can replace the configuration completely.

> Tip:
>
> When importing or merging your column configuration, remember that any user that has customized their configuration has their own **GroupMember** column configuration and they will not see your new configuration until they delete them from the **Arrange** panel. You may delete *all* configurations for your target scope before you import by using the **delete_uiconfig** utility.

Syntax for column configuration

The XML file for column configuration consists of any number of *column configurations*. A column configuration specifies which columns are displayed and in which order, as well as sorting, and filtering options.

Following is the general structure of this file:

```
<Import>
   <Client abbreviation="" name="">
      <ClientScope hostingClientName="" name="" uri="">
         <ColumnConfig columnConfigId="" columnConfigName="" sortBy="" sortDirection="">
            <ColumnDef objectType="" propertyName="" width=""/>
            <ColumnDef objectType="" propertyName="" width=""/>
             ...
         </ColumnConfig>
      </ClientScope>
   </Client>
</Import>
```

Within the XML file, the following elements are used:

<Import>

The root element for this file. There are no attributes.

<Client>

This element contains the name and abbreviation of the client which will use this definition. At this time, the only valid client is Active Workspace, which is **AWClient**.

© 2025 Siemens

abbreviation	**AWClient**
name	**AWClient**

<ClientScope>

This element wraps the column configuration and identifies the URI to which it applies. Every client scope *must* have a site-level unnamed column configuration (one without a **columnConfigName** defined).

hostingClientName *Reserved for future use.*

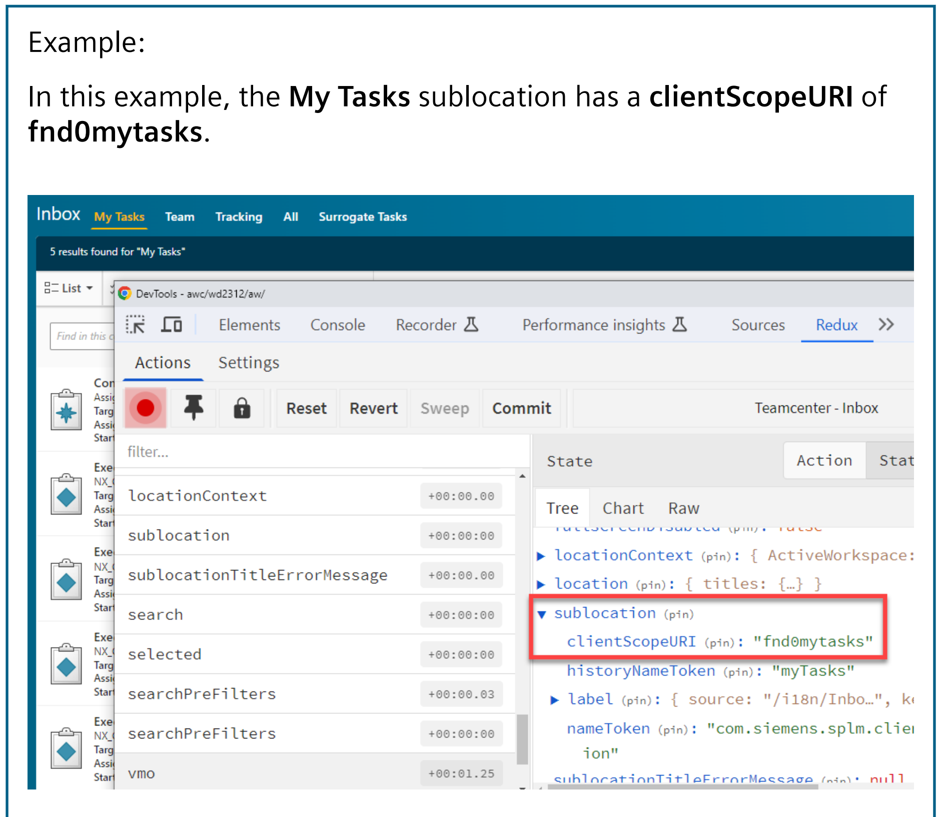
name The name of the table you are configuring. This field is populated on export; it is not processed during import.

uri Matches the **clientScopeUri** from a declarative state definition.

To discover a state's client scope URI, examine the page's CTX object.

Example:

In this example, the **My Tasks** sublocation has a **clientScopeURI** of **fnd0mytasks**.



© 2025 Siemens

<ColumnConfig>

When the client asks for a column configuration, this element defines the response. Internally, it either contains a list of column definitions, or a reference to another column configuration. Either way, the client does not know the difference.

columnConfigId

The unique identifier of the configuration being defined. Each client scope or object set **URI** can have only one column config ID associated with it.

columnConfigName (optional)

The user-facing name of the configuration. Setting this attribute will add this configuration to the users' **Arrange** panel, as a selectable choice. You can define multiple named configurations for each **columnConfigId**. You can also leave this undefined, making this an unnamed column configuration.

> Note:
>
> When you specify a named column configuration, it is *always* a site configuration. Any command-line arguments specifying any other scope, like **for_group** for example, will be ignored for this configuration.

sortBy

Specify the column by which the table is initially sorted. Valid values are:

0 | 1 | {null}	Choose the first column.
2 ... n	Choose another column by number. The third column is "3" for example.
-1	Let the data provider decide.

sortDirection

Ascending | Descending

wrapText

true | false

<ColumnDef>

This element specifies which property is to be displayed and must only be contained within a **<ColumnConfig>** element. Each row of this column is a property taken from an object in the table.

columnName

The TextServer key for the display name of the column. If this key string is not found in by the TextServer, it will be displayed as-is by the UI.

> Caution:
>
> Only use custom column names for dynamic compound properties. Do not override the names of real properties.

© 2025 Siemens

> Caution:
>
> Specifying more than one property with the same name in the same table is not supported.

filterable

Controls filtering for this column. The type of filter shown to the user will depend on the data type of the column. Dates and numbers are automatically detected and will allow a range filter, but everything else is treated as a string.

The default is **true**. Filtering is on by default.

```
<ColumnDef columnName="object_name"
filterable="false" ... />
```

hidden

If set to **true**, this column will not be displayed, but will be available for the user to display if they want.

The default is **false**. Columns are not hidden by default.

```
<ColumnDef columnName="object_name"
hidden="true" ... />
```

objectType

The type of object for which this column is valid. The column will only appear if every object in the table is a valid **objectType** or one of its children in the server-side **POM**.

> Example:
>
> **ItemRevision** will display **Document**s, **Design**s, **Part**s, and so on, but not files. **Dataset** would show all file types, but not item revisions. **WorkspaceObject** shows all normal objects, like item revisions, files, folders, and so on.

propertyName

The name of the property whose value will be displayed.

width

The initial width of the column in pixels.

> Caution:
>
> Avoid specifying the same column property twice if the types have a parent-child relationship. Table columns do not display duplicate names in column definitions.
>
> For example, if **Schedule** is a child type of **MyItem**, and the **item_id** is defined for both the **MyItem** parent type and the **Schedule** child type, only one of these columns can be displayed in the table.

© 2025 Siemens

> Tip:
>
> Active Workspace is designed primarily to display item revisions. If you decide to display an item, the **item_id** property of the item will not be available by default. You must add the **awp0Item_item_id** property to your column configuration XML file.

Example of column configuration

How do I know which column configuration is being used?

In this example, you search for **hard drive** and then examine the search results when viewed in the table format.

There are two pieces of information you need to determine which column configuration is used to display the table; your current page and the configuration identifier. Both can be retrieved by using your browser's developer tools to examine the context object.

- **clientScopeURI**

  The current page, or *client scope URI* in this example, it is the **Awp0SearchResults** URI.

```
▶ searchSearch: {criteria: "hard drive", searchFilterString:
  selected: null
▶ selectedModelTypeRelations: (7) ["Awb0DesignElement", "Iter
▶ serverCommandVisibility: {soaCallFinished: true}
▶ state: {params: {…}, processed: {…}}
▼ sublocation:
    clientScopeURI: "Awp0SearchResults"
    historyNameToken: "teamcenter_search_search"
  ▶ label: {source: "/i18n/SearchMessages", key: "resultsText
    nameToken: "com.siemens.splm.client.search:SearchResultsS
  ▶ sortCriteria: []
  ▶ __proto__: Object
```

- **columnConfigId**

  The column configuration identifier is part of the message for the network call that populates the table. In this example, the **performSearchViewModel3** call shows the **searchResultsColConfig** identifier.

© 2025 Siemens

Name	× Headers Preview Response Cookies Timing
performSearchViewModel3	▼{.QName: "http://awp0.com/Schemas/Internal/AWS2 .QName: "http://awp0.com/Schemas/Internal/AWS ▶ServiceData: {plain: ["ghaxXMlbJcQ1UA", "hyUx ▶additionalSearchInfoMap: {searchTermsToHighli ▼columnConfig: {columnConfigId: "searchResults columnConfigId: "searchResultsColConfig" ▶columns: [{displayName: "Name", typeName: " operationType: "Intersection"

You might see a different version of the **performSearchViewModel** call, but the content will be similar.

The column configuration

With the ID of the column configuration, you can locate the definition. Column configurations can be registered for the Site, Group, Role, and Workspace levels. In addition, when a user arranges a table in the UI, an automatic configuration is created at the Groupmember level. Following is an example of the search result column configuration shown in the exported XML file:

The column configuration associated with the search result table is defined under the **Awp0SearchResults** URI. This is the configuration that you see in the search results table. Following is an example of the object navigation column configuration:

```
<ClientScope hostingClientName="" name="Results" uri="Awp0SearchResults">
   <ColumnConfig columnConfigId="searchResultsColConfig" ...
      <ColumnDef columnName="object_name" objectType="WorkspaceObject"
propertyName="object_name" ...
      <ColumnDef columnName="object_desc" objectType="WorkspaceObject"
propertyName="object_desc" ...
      ...
   </ColumnConfig>
</ClientScope>
```

There are many columns defined in this configuration, from object name and description, to published object creation date and type as well as many others depending on which Teamcenter applications you have installed. They will appear in this order by default in the client.

Tip:

You must not use the same column configuration ID within more than a single client scope URI. You can define the same column configuration ID within the same client scope multiple times, but you must specify a different client configuration name each time.

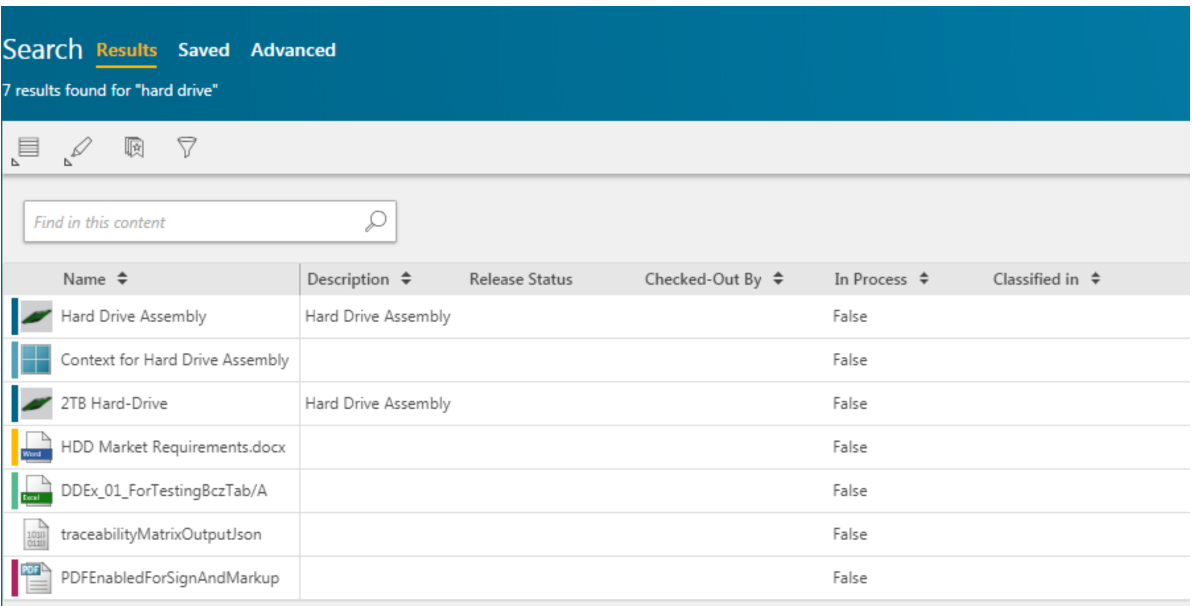
```
<ClientScope hostingClientName="" name="Results" uri="Awp0SearchResults">
  <ColumnConfig columnConfigId="searchResultsColConfig" ...
    <ColumnDef columnName="object_name" ...
```


© 2025 Siemens

```
            ...
      </ColumnConfig>
      <ColumnConfig columnConfigId="searchResultsColConfig" columnConfigName="Object_Navigation1"
...
         <ColumnDef columnName="object_name" ...
            ...
      </ColumnConfig>
      <ColumnConfig columnConfigId="searchResultsColConfig" columnConfigName="Object_Navigation2"
...
         <ColumnDef columnName="object_name" ...
            ...
      </ColumnConfig>
      <ColumnConfig columnConfigId="searchResultsColConfig" columnConfigName="Object_Navigation3"
...
         <ColumnDef columnName="object_name" ...
            ...
      </ColumnConfig>
   </ClientScope>
```

Search results with dissimilar types — Primary Work Area

The search produced multiple results, shown here in the table format. The objects returned are of differing types, some are item revisions, some are files, and so on.

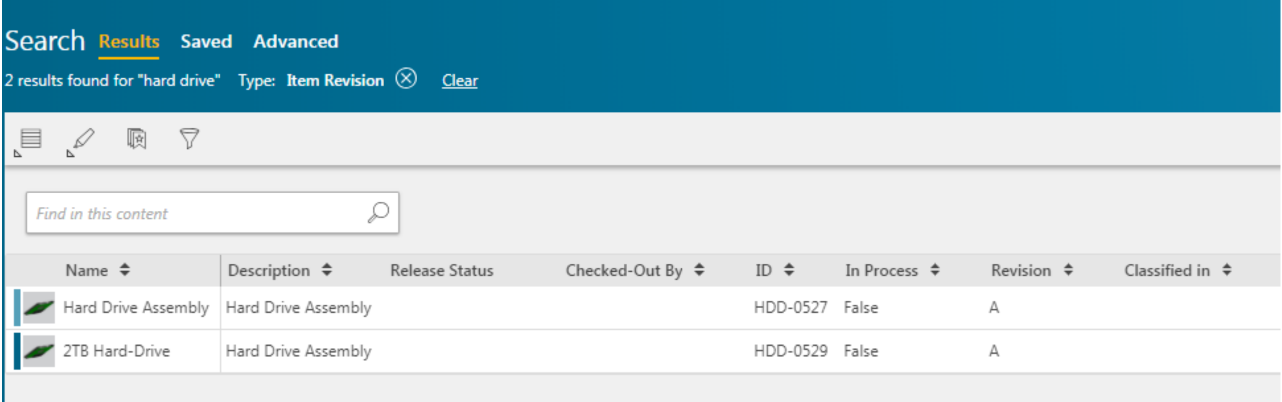


Even though there are 16 columns defined for this column configuration, you only see 6. That is because the other 10 properties do not exist on all the different object types. For example, files do not have an item ID or revision ID property, so that column is not displayed.

Search results with a common type

If you filter the results to show only **Item Revision**s, the **ID** and **Revision** columns appear, because all the listed objects have that property.

© 2025 Siemens



User modification of column configuration

After the column configurations have been created and stored in the system, the end user can modify their column layout to a certain extent using Active Workspace interface. They can:

- Change the column order.
- Change the sorting column and direction.
- Hide certain columns.
- Change where the column freeze begins. Anything to the right of the frozen columns will scroll horizontally if needed. The frozen columns will stay in place when the user scrolls to the right.

  If the user does change which column is frozen, this change does not persist. If they navigate away and come back, or even refresh the page, the frozen column returns to its original definition.

They can not:

- Add columns that are not present in the original definition.
- Reset more than one *GroupMember* configuration at a time.

> Tip:
>
> All of the user's changes are stored in the Teamcenter database at the *GroupMember* level (the specific combination of user and their group/role). The import_uiconfig and export_uiconfig utilities do not work with *GroupMember* configurations. To clear a *GroupMember* configuration for a page, the user must go to the page as the correct group and role, and then use the **Arrange** panel to **Reset** their configuration.
>
> You can delete *all configurations for all users* for a specific page or scope by using the delete_uiconfig utility.

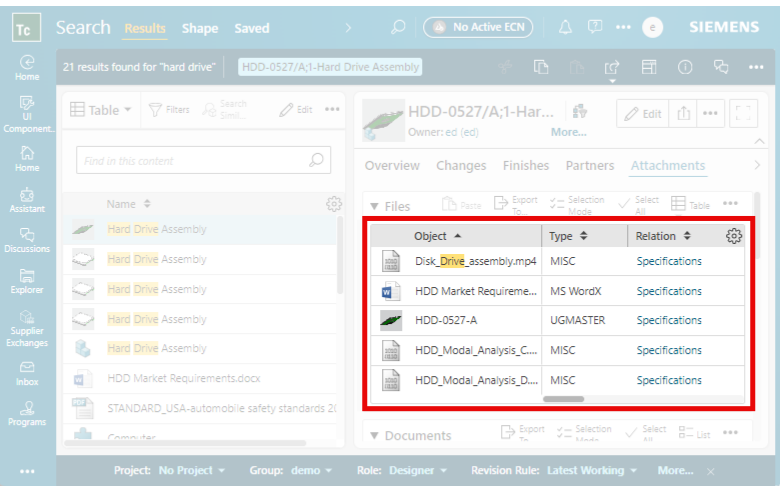
© 2025 Siemens

Configuring object set tables

Configuring columns using objectSet properties

Tables placed in the UI by style sheets (using the **objectSet** element) can only be located in the secondary work area. A few of the most common examples of object set tables are:

- Files attached to the selected object.
- Related **Simulation** results.



Using object sets

The source attribute of an object set not only filters all related objects down to the few you wish to highlight for a given purpose, showing the user only what they need to see at that time, but also restricts the creation of objects and their relations to the source combinations.

Example:

You want to create a table in which the user will only see word files that have the **Attaches** relation and PDF files that have the **manifestation** relation.

```
source="TC_Attaches.MSWordX,IMAN_manifestation.PDF"
```

This also has the benefit of showing the user only the **Word** and **PDF** types when they add new objects to the table. If the user chooses to add a word object to the table, then they can only choose the **Attaches** relation.

The initial list of properties and the column layout of object set tables is defined using the **property** element in the object set definition for the table's initial layout.

© 2025 Siemens

Property element

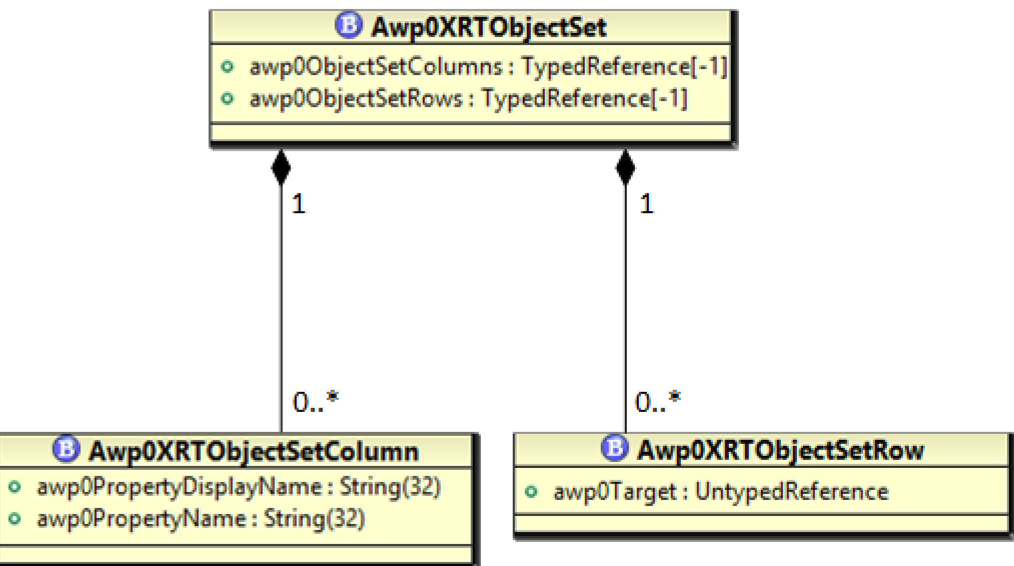
Define a style sheet table's initial list of properties and the column layout by defining it directly in the **objectSet** XRT element using **property** tags. A new column configuration will automatically be created and managed by Active Workspace.

How are objectSet tables rendered?

When Active Workspace uses XML rendering templates (XRT), also known as style sheets, to render a table of objects and their properties, it takes its cues from the **<objectSet> element**.

How does an objectSet work?

The **objectSet** XRT element models its table data using several runtime business objects, organized by the **Awp0XRTObjectSet** object.



objectSet rendering example

You want to create a table to display the following when a user selects a **DocumentRevision**:

- Any word files attached using the **Attaches** relation.
- Any PDF files attached using the **Manifestation** relation.

You create a style sheet entry for the **DocumentRevision** with an **objectSet** which uses **TC_Attaches.MSWordX** and **IMAN_manifestation.PDF** as the sources. Following is a simplified example:

© 2025 Siemens

```
<section titleKey="Files">
    <objectSet source="TC_Attaches.MSWordX,IMAN_manifestation.PDF" >
        <tableDisplay>
            <property name="object_string"/>
            <property name="object_type"/>
            <property name="owning_user"/>
        </tableDisplay>
    </objectSet>
</section>
```

When a user selects a document revision, Active Workspace renders the associated table as follows:

1. An **Awp0XRTObjectSet** object is created, along with an **Awp0XRTObjectSetColumn** object for each property to be displayed, in this case, three.

Awp0XRTObjectSet	→ Awp0XRTObjectSetColumn → object_string
	→ Awp0XRTObjectSetColumn → object_type
	→ Awp0XRTObjectSetColumn → owning_user

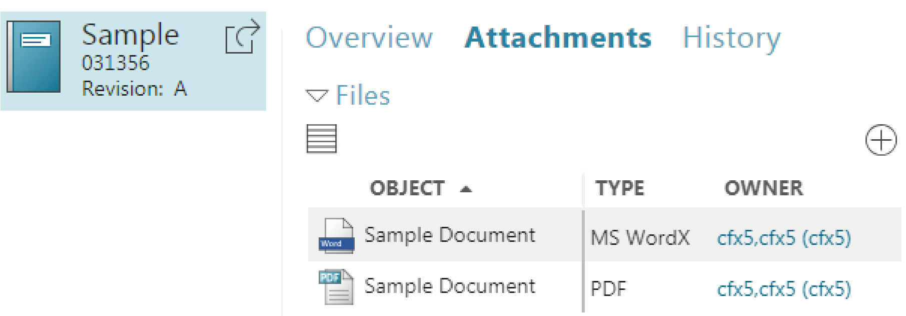
2. An **Awp0XRTObjectSetRow** object is created for each object found that matches the source, in this case, two. Each row references its target object with the **awp0Target** property.

Awp0XRTObjectSet	→ Awp0XRTObjectSetRow → Sample Document (word)
	→ Awp0XRTObjectSetRow → Sample Document (PDF)

3. Each row retrieves its target object's icon along with the properties specified by the column objects.

Awp0XRTObjectSetRow →	Word	Sample Document	MS WordX	cfx5, cfx5 (cfx5)
Awp0XRTObjectSetRow →	PDF	Sample Document	PDF	cfx5, cfx5 (cfx5)

4. The resultant table is displayed to the user.



© 2025 Siemens

Caveats

The **Awp0XRTObjectSetRow** displays the base type icon of the **awp0Target** object. It does not consider icons assigned declaratively. If you wish to display a different icon, you may dynamically assign an icon to the **Awp0XRTObjectSetRow** based on what it is pointing to.

The **Awp0XRTObjectSetRow** does not have access to properties not defined at the source level. For example, if you specify:

```
source="IMAN_reference.ItemRevision"
```

The table will display all **ItemRevision** objects attached as a reference, including any child types like **DocumentRevision**, **DesignRevision**, and so on. However, if you added the following to the table's properties:

```
<property name="DocumentTitle"/>
```

It would not be able to find it on related **DocumentRevision**s, because that property is not defined a the **ItemRevision** level. In order for the object set to know about the property, you would have to modify the source to include the object where the property is defined.

```
source="IMAN_reference.ItemRevision,IMAN_reference.DocumentRevision"
```

objectSet tables and revision rules

Normally the object set table retrieves the *source* objects and displays them, even if the source object is an item type. However, if you want the table to dynamically show a *configured* revision of the item instead of the item itself, you must set **showConfiguredRev** attribute to true in the **objectSet** definition.

If **showConfiguredRev="true"** and one of the sources specified is an item type, the table still keeps track of the item; it merely displays the configured revision. In this case:

- When a user cuts or copies a configured revision, it is the underlying source item that is acted upon.
- If a revision of a specified item type is pasted, the associated item is pasted instead.
- If the user pastes a specified item type, the item will be pasted, but the appropriate revision will be displayed.

Many of the tables and folders in Active Workspace are already configured for this behavior, but this is not the default behavior. If you create a custom table you must set **showConfiguredRev="true"** in your object set definition.

Example:

Following is an excerpt from the style sheet of the folder summary page:

© 2025 Siemens

```
<objectSet source="contents.WorkspaceObject" ... showConfiguredRev="true" >
```

objectSet tables and data providers

Data providers

Data providers are subclasses of the **Fnd0BaseProvider** business object. In Active Workspace, they can be specified as the **source** of an **objectSet**.

```
<objectSet source="< Data provider >.< Filtered BO >" defaultdisplay="..." sortby="..."
sortdirection="...">
```

Example

This example shows an **objectSet** table configured to show parent partitions.

```
<objectSet source="Fgf0ParentPartitionsProvider.Ptn0Partition"
           defaultdisplay="listDisplay"
           sortby="object_string"
           sortdirection="ascending">
```

objectSet tables and default relations

Default relations

In Active Workspace, when a user creates a new attachment using a table, the relation is chosen using a combination of Teamcenter's default paste relation preferences and the table's definition.

The **source="..."** attribute of the **<objectSet>** tag not only filters the related objects which are displayed on the table, but also defines the list of *allowed* relations for new objects. Object-relation pairs that are not defined in the **<objectSet>** are not allowed when adding to a table.

Following is the priority used when determining the relation of a newly attached object.

1. If the *type1*_*type2*_**default_relation** preference exists, and the relation specified in the value also matches a relation defined in the table, then use it.

2. If not, then check the *type1*_**default_relation** preference. If it exists and its value specifies a relation defined in the table, then use it.

3. If neither of the preceding were successful, then set the relation to the first value defined in the table.

Example

In the following scenarios, a table is defined on an **ItemRevision** business object as shown,

© 2025 Siemens

```
<objectSet source ="IMAN_reference.Dataset,IMAN_specification.Dataset,
                    IMAN_manifestation.Dataset,IMAN_Rendering.Dataset"
```

and the following preferences are set.

```
ItemRevision_default_relation = IMAN_specification
ItemRevision_DirectModel_default_relation = IMAN_Rendering
ItemRevision_MSWORD_default_relation = TC_Attaches
```

- Scenario 1 — A **UGMaster** is added.

    1. There is no **ItemRevision_UGMaster_default_relation** preference defined.

    2. The value of **ItemRevision_default_relation** is **IMAN_specification**. This relation combination is allowed by the table (`IMAN_specification.Dataset`), so this is chosen to be the relation of the new dataset.

- Scenario 2 — A **DirectModel** is added.

    1. The value of **ItemRevision_DirectModel_default_relation** is **IMAN_Rendering**. This relation combination is allowed by the table (`IMAN_Rendering.Dataset`), so this is chosen to be the relation of the new dataset.

- Scenario 3 — An **MSWORD** is added.

    1. The value of **ItemRevision_MSWORD_default_relation** is **TC_Attaches**. This relation combination is *not* allowed by the table.

    2. The value of **ItemRevision_default_relation** is **IMAN_specification**. This relation is allowed by the table (`IMAN_specification.Dataset`), so this is chosen to be the relation of the new dataset.

User modifiable relations

To allow the user to modify the relation of an attachment in an **objectSet**, you must add the **modifiable="true"** attribute to the relation property tag. For example:

```
<objectSet source="IMAN_specification.Dataset,IMAN_reference.Dataset,
                   IMAN_manifestation.Dataset, IMAN_Rendering.Dataset"
           defaultdisplay="listDisplay" sortby="object_string" sortdirection="ascending">
   <tableDisplay>
      <property name="object_string"/>
      <property name="object_type"/>
      <property name="relation" modifiable="true"/>
      <property name="release_status_list"/>
   </tableDisplay>
   <thumbnailDisplay/>
   <listDisplay/>
</objectSet>
```


© 2025 Siemens

Doing this allows the user to change the relation type using a drop-down list.

To disable this ability, remove the **modifiable** attribute from the **property** tag.

```
<objectSet source="IMAN_specification.Dataset,IMAN_reference.Dataset,
                   IMAN_manifestation.Dataset, IMAN_Rendering.Dataset"
           defaultdisplay="listDisplay" sortby="object_string" sortdirection="ascending">
   <tableDisplay>
      <property name="object_string"/>
      <property name="object_type"/>
      <property name="relation"/>
      <property name="release_status_list"/>
   </tableDisplay>
   <thumbnailDisplay/>
   <listDisplay/>
</objectSet>
```

objectSet tables and secondary-to-primary relations

Normally when retrieving related objects, you start with the primary object, and look for a secondary object attached with a certain relation. This is called a primary-to-secondary relation. When defining the **source** of an **objectSet**, this is the default behavior.

Secondary-to-primary relations

Active Workspace also allows you to retrieve a related primary object from a secondary object. This uses the same relation but in the opposite direction, and is called a secondary-to-primary relation (S2P), also commonly called where-referenced information.

Retrieving primary related objects in an **objectSet** table is accomplished by adding **S2P:** in front of the relation type.

```
<objectSet source="S2P:relation.object, ..."
```

Example

If there is a **Design Revision** which has a **Document Revision** related to it using the **References** relation, you can show the related **Design Revision** in an **objectSet** from the **Document Revision** XRT as follows:

© 2025 Siemens

```
<objectSet source="S2P:IMAN_reference.Design Revision"
```

Using dynamic compound properties as a source

If you want to display a table of properties from objects that are more than a single relation away, you must use dynamic compound properties (DCP) in your source attribute.

You can use any combination of DCP and regular source clauses in a comma-separated list. Pagination, sorting, filtering, and editing are supported on tables using DCP source attributes.

Warning:

In object set tables, authoring commands like **Add to**, **Paste**, **Delete**, **Cut**, and so on are not supported when you use DCP in your source attribute. Even if these commands are displayed elsewhere, they are still not supported in this case.

Because the source attribute is expecting objects, the DCP must point to an object, relation, or a reference to an object. It cannot point to a regular property.

Example:

SUPPORTED — A DCP source pointing to a typed or untyped reference or an untyped relation:

```
<objectSet source="REF(items_tag,Item).owning_user" >
```

This returns the **owning_user** typed reference which would point to a **User** object.

Example:

SUPPORTED — A DCP source resolving a reference, pointing to the target object, but without a property at the end:

```
<objectSet source="REF(items_tag,Item).REF(owning_user,User)" >
```

This returns the **User** object for the owner of the item, but doesn't specify a property.

Example:

NOT SUPPORTED — A DCP source pointing to a normal property.

```
<objectSet source="REF(items_tag,Item).REF(owning_user,User)
.user_name" >
```

This returns **user_name**, which is a string value from the **User** object, and will be ignored by the source attribute.

© 2025 Siemens

Supported syntax

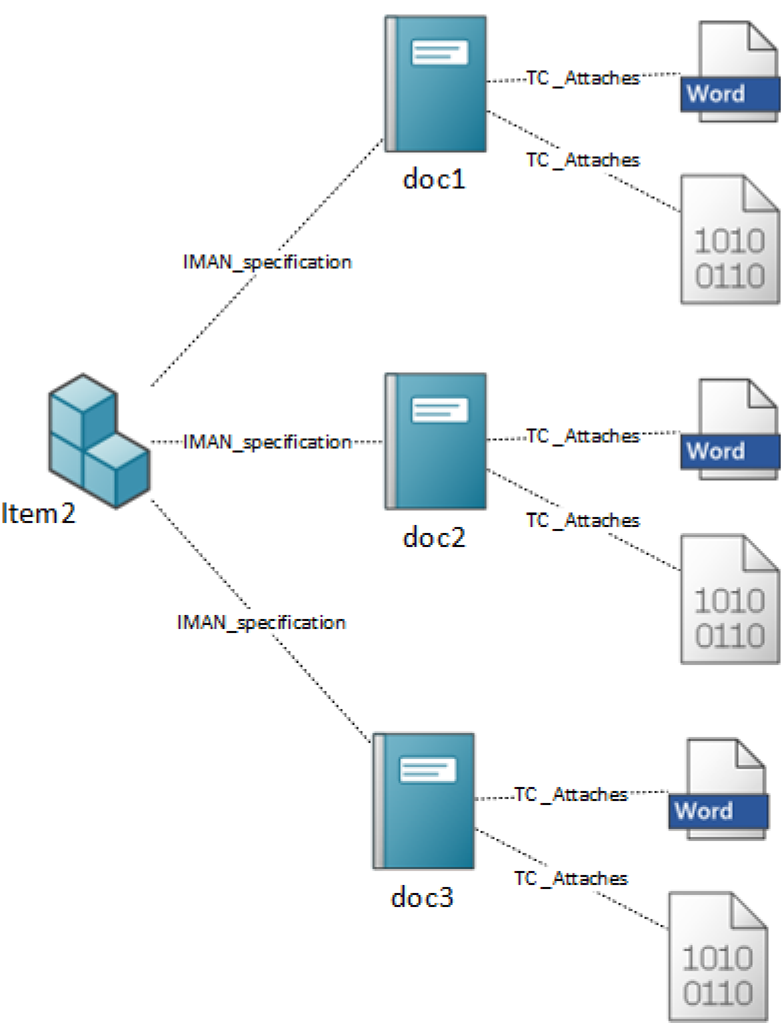
You can use the following **DCP syntax** types in your source attribute, including multiple-level traversal:

- Typed Reference property (REF)
- Back pointer reference property (REFBY)
- GRM Primary to Secondary (GRM)
- GRM Secondary to Primary (GRMS2P)
- Properties on Relation (GRMREL or GRMS2PREL)

Full example

In this example object set table, from the item revision you want to show the datasets of related documents without forcing the user to navigate through the document revisions.

© 2025 Siemens



Follow any specification relations to any document revisions, then follow any attaches relations to any datasets.

```
<objectSet
source="GRM(IMAN_specification,DocumentRevision).GRM(TC_Attaches,Dataset)"  ...>
    <tableDisplay>
        <property name="object_string"/>
        <property name="object_type"/>
        ...
    </tableDisplay>
    ...
</objectSet>
```

Your table shows the properties of all datasets attached to documents that are related using the specification relation.

© 2025 Siemens

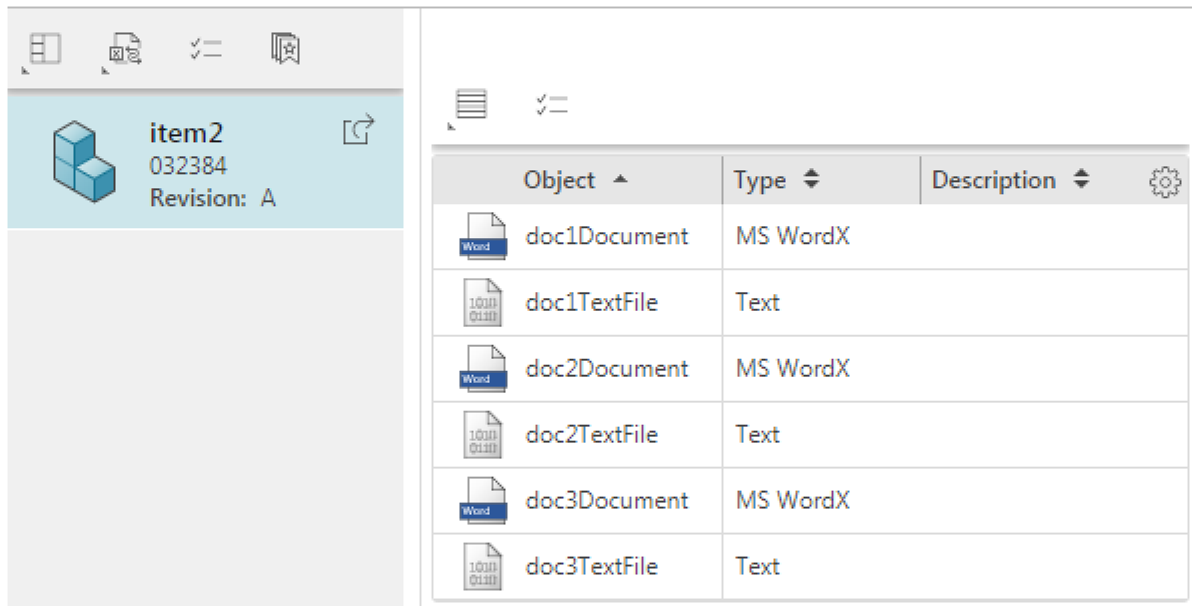
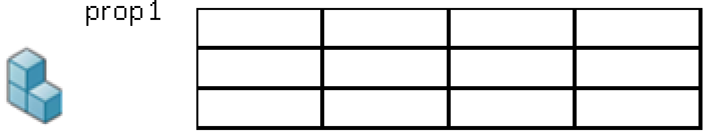


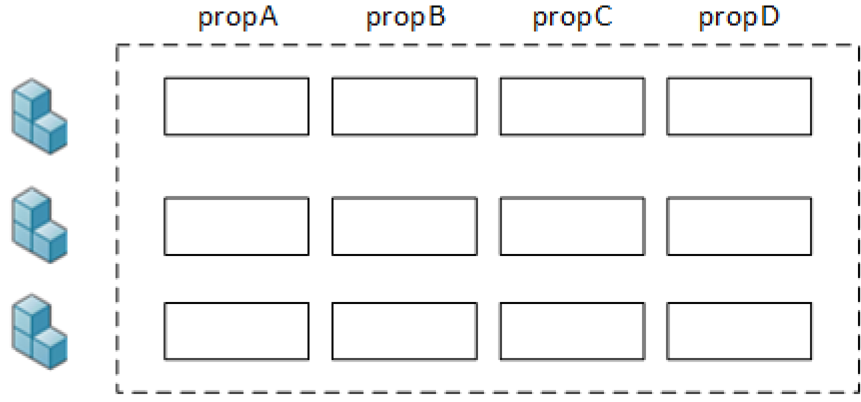
Table properties (are not objectSets)

A table property might seem like a table containing multiple properties, but it is actually a single property containing a table of values.

Table property A table property is a persistent table of property values stored on a single object. You can think of it like a spreadsheet as a single property.



Object set An object set is a runtime table of properties where each row is a set of properties from a separate object.

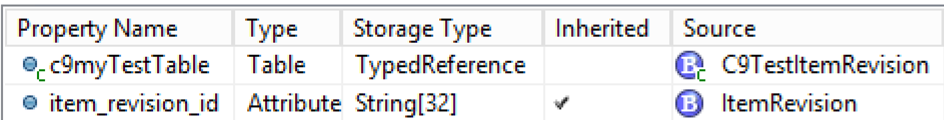


© 2025 Siemens

> Note:
>
> Information on creating table properties is found in BMIDE for Data Model Design in the Teamcenter Help.

Example

You want to display a table property **c9myTestTable** that is defined on a custom business object **C9TestItemRevision**.



Property Name	Type	Storage Type	Inherited	Source
c9myTestTable	Table	TypedReference		C9TestItemRevision
item_revision_id	Attribute	String[32]	✔	ItemRevision

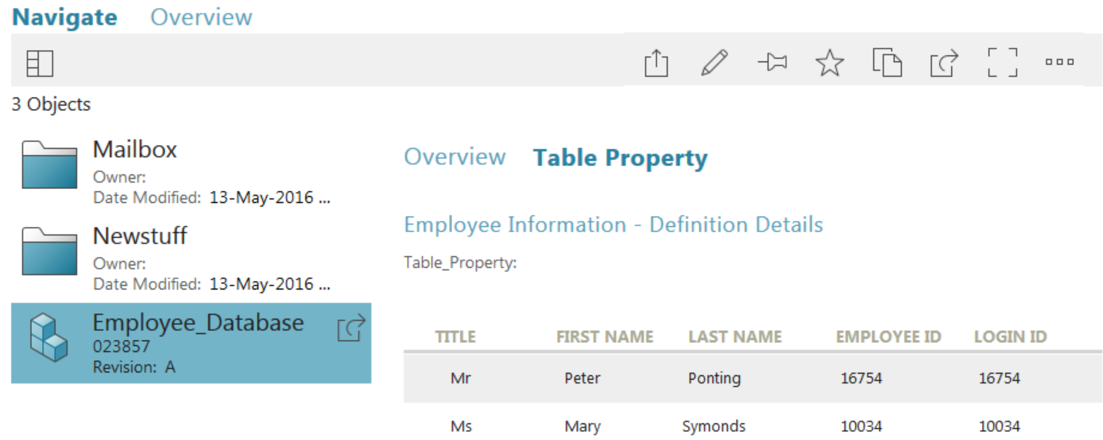
The individual properties are stored on a child of the **Fnd0TableRow** object, in this example **C9myTableRowObject**.

Property Name	Type	Storage Type	Inherited	Source
c9FirstName	Attribute	String[128]		C9myTableRowObject
c9LastName	Attribute	String[128]		C9myTableRowObject
c9EmployeeID	Attribute	String[128]		C9myTableRowObject
c9LoginID	Attribute	String[128]		C9myTableRowObject
fnd0RowIndex	Attribute	Integer	✔	Fnd0TableRow

To display the table property in Active Workspace, use the **tableProperty** element in your custom summary style sheet. For example:

```
<page title = "Table Property" visibleWhen="object_type==C9TestItemRevision">
  <section title ="Employee Information - Definition Details">
    <tableProperty name="c9myTestTable">
      <property name="C9Title"/>
      <property name="C9FirstName"/>
      <property name="C9LastName"/>
      <property name="C9EmployeeID"/>
      <property name="C9LoginID"/>
    </tableProperty>
  </section>
</page>
```


© 2025 Siemens



Configuring page layout

XRT information specific to Active Workspace

Using XML rendering templates (XRT) with Active Workspace

XRT files use the XML format. They are used to configure layout in Teamcenter clients, including Active Workspace, based on object type, group, and role. XRT files are also commonly referred to as style sheets; however, they neither follow CSS or XSL standards, nor do they perform transformations.

In Active Workspace, XRT files control areas such as the *secondary work area* and the *tools and information panel*.

Style sheet preference names follow this format:

AWC_<*type-name*>**.*RENDERING**

The value of the preference points to a style sheet dataset name, which is used by Active Workspace's XRT renderer to produce the UI. Create a copy of an OOTB style sheet for your custom layout, and then modify or override the preference value to use your new dataset instead. Use the **XRT Editor** to assist in editing style sheets.

Considerations for using XRTs in Active Workspace

Although Active Workspace uses XRTs like the other clients do, there are some differences:

- Active Workspace's XRT rendering preferences include **AWC_** as a prefix to the preference name, allowing for the assignment of style sheets that are unique to Active Workspace.

  For example, normally the **ItemRevision.SUMMARYRENDERING** preference registers the XRT used for the summary display for the ItemRevision in all clients, but there is also an

© 2025 Siemens

AWC_ItemRevision.SUMMARYRENDERING preference that overrides it specifically for Active Workspace, allowing for separate style sheets.

- Layout of Active Workspace XRTs in landscape mode is wide compared to the rich client and therefore, requires multiple columns.

- When creating copies of the out-of-the-box XRT files for Active Workspace, create the XRT files with unique names and assign them using the **AWC_**<*type-name*>**.*RENDERING** preferences.

- Active Workspace can share XRT files with the rich client for the **Summary** tab. The default summary XRT preference for Active Workspace is **AWC_ItemRevision.SUMMARYRENDERING**. If you remove this preference, Active Workspace uses the default summary XRT preference used by the rich client: **ItemRevision.SUMMARYRENDERING**.

- If you do not want to use the header on the overview page, you can remove it from the XRT files used by the **AWC_**<*type-name*>**.SUMMARYRENDERING** preferences.

- Active Workspace supports:

  - Multiple pages using **visibleWhen**.
  - A single level of columns, sections, and separators.
  - You can use a percentage in the **width** attribute for the **<column>** XRT element. The default is 100% divided by the number of columns.

    ```
    <column width="30%" ... />
    ```

    This is always a percentage, even if the percent sign is not used. This is a percentage of the overall screen width. When the column percentages add to less than 100%, Active Workspace will not fill. When the column percentages add to more than 100%, overflow columns are displayed on a new row.
  - Labels
  - Breaks

- The following are not currently supported in Active Workspace:

  - Custom rendering hints

  - Multiple levels of columns, sections, and separators.

  - Using the **visibleWhen** attribute with preferences.

  - Using the **visibleWhen** attribute with multiple-level property traversal.

© 2025 Siemens

Note:

Embedded task actions replace the **Perform Task** button in the **Task Overview**. For customers with custom style sheets that wish to retain the perform command in place of the embedded actions set the **WRKFLW_Hide_Perform_Task_Command_ToolAndInfo** preference to **false**. The default value is **true**.

Configure the information panel using XRTs

The information panel in Active Workspace displays details about the opened object and is accessed by clicking the ⓘ button.

Information panel

You use XRT datasets to configure the layout of the information panel. By default, there is an XRT dataset for **WorkspaceObject** and **ItemRevision** object types. To modify the information displayed in the information panel for other types, you must create an **XMLRenderingStylesheet** dataset, attach an XML file to it, and then create a preference to point to the dataset. The XRT is registered using the **AWC_**<*type-name*>**.INFORENDERING** preference.

1. Create a dataset of type **XMLRenderingStylesheet**.

   Tip:

   You can copy an existing XRT dataset and rename it rather than create a new one. Find existing XRT datasets in the rich client by searching for **XMLRenderingStylesheet** dataset types. Then copy an existing XRT dataset by selecting it and choosing **File→Save As**. Make sure you change the named reference file attached to the dataset to point to a unique file name.

2. Attach the XML file to the new dataset as a named reference.

   Siemens Digital Industries Software recommends that your style sheet be set up to display content in the information panel as follows:

   - Limit to one or two pages
   - Limit to one column per page
   - Use list displays for object sets

   Keep in mind the following:

   - Keep it simple. Do not make the layout the same as the summary or overview pages.

© 2025 Siemens

- Active Workspace supports multiple pages with the **visibleWhen** tag, sections, and **objectSet** tables (use the tile/list mode to fit the narrow display).
- The XRT used in the user interface is based on the selected object's hierarchy. For example, if you select an **Item** object type, but it does not have an XRT associated with it, the XRT for **AWC_WorkspaceObject.INFORENDERING** is used because an **Item** is also a **WorkspaceObject**.

3. Create a preference using the following parameters:
    - **Name**: **AWC_**<*type-name*>**.INFORENDERING**, for example, **AWC_WorkspaceObject.INFORENDERING**.
    - **Value**: Name of the dataset created in step 1.
    - **Scope**: Site preference.

Create style sheets for Active Workspace

If you want to have different style sheets in Active Workspace than you have in other Teamcenter clients, you must create preferences which begin with **AWC_**. This has no effect on the other clients or on any customer-created style sheets.

1. Create a dataset with a type of **XMLRenderingStylesheet**.
2. Attach the XRT style sheet to the new dataset as a named reference.
3. Use a text editor to edit the style sheet as necessary.
4. Create a preference using the following parameters:
    - **Name**:

      AWC_<*type-name*>**.SUMMARYRENDERING**

      AWC_<*type-name*>**.CREATERENDERING**

      AWC_<*type-name*>**.INFORENDERING**

      AWC_<*type-name*>**.REVISERENDERING**

      AWC_<*type-name*>**.SAVEASRENDERING**

      For example:

      ```
      AWC_WorkspaceObject.INFORENDERING
      ```


© 2025 Siemens

- **Value**: Name of the dataset created in step 1.
- **Scope**: Site preference.

When rendering style sheets, Active Workspace first searches for **AWC_**<*type_name*>**. [SUMMARYRENDERING, CREATERENDERING, ...]**. If no match is found, it searches for <*type-name*>**. [SUMMARYRENDERING, CREATERENDERING, ...]**. If it still does not find a match, it continues with the standard lookup mechanism for style sheets.

It is possible to register style sheets to a specific location, sublocation, or object type.

Use the following format to create the registration preferences:

type . location . sublocation . `SUMMARYRENDERING`

- **type** specifies the type of object. PartRevision or DesignRevision, for example.
- **location** specifies the location in the UI. For example, if the context is `com.siemens.splm.clientfx.tcui.xrt.showObjectLocation`, then the location is **showObjectLocation**.
- **sublocation** specifies the sublocation in the UI. For example, if the context is `com.siemens.splm.client.occmgmt:OccurrenceManagementSubLocation` then the sublocation is **OccurrenceManagementSubLocation**.

As with normal registration preferences, the value of this preference is the name of the dataset. When rendering a page, the system will search for the most specific case to the most general.

- **type . location . sublocation .** `SUMMARYRENDERING`
- **type . location .** `SUMMARYRENDERING`
- **type .** `SUMMARYRENDERING`

If none of the above preferences are found for the object, the immediate parent type will be searched in the same manner. This process continues until a match is found.

Modular style sheets

Use the `<inject>` tag to refer to another **XMLRenderingStylesheet** dataset that contains a block of XML rendering tags. These injected XML rendering style sheets would be incomplete on their own, normally containing only a single page or section each, but they allow a modular approach to style sheet design and maintenance.

In the example below, a second **XMLRenderingStylesheet** dataset exists with the name **myXRTblock**.

© 2025 Siemens

There are two methods of specifying which dataset is to be used.

- Directly, by referring to the name of the **XMLRenderingStylesheet** dataset.

```
<inject type="dataset" src="myXRTblock"/>
```

- Indirectly, by referring to a preference which contains the name of the dataset.

```
<inject type="preference" src="additional_page_contributions"/>
```

Then create the **additional_page_contributions**preference, containing the value **myXRTblock**.

If the preference contains multiple values, then each dataset will be located and injected in order.

As well formed XML files must have a root node in order to be well-formed, the XRT you inject must be wrapped in a **<subRendering>** element.

```
<subRendering>
      <label text="This text will get injected."/>
</subrendering>
```

Someone leveraging injection must think about the resulting XML file, so that the resulting XRT will be correct. The injection mechanism does not make any assumptions about where it is injecting data.

Tip:

- Using a large amount of **<inject>** elements in your XML rendering templates can negatively impact the performance of the client.
- Avoid using **visibleWhen** to check object types in an XRT. The object type is determined when the XRT is registered. Do not attempt to create a single, over-arching style sheet for all object types.

Conditional content

This element determines the visibility of a container based on a condition.

Supported style sheets

This tag is supported on the following types of style sheets:

Summary
Information
Revise
Save As

© 2025 Siemens

visibleWhen

Active Workspace supports the **visibleWhen** attribute for the following tags:

- <content>
- <page>

The **visibleWhen** attribute accepts the following comparison results:

Property type	Exact match	NULL
string, char, note, int, short, double, float	Yes	Yes
logical	Yes (see below)	Yes
date, typed reference, untyped reference	Not available	Yes

Note:

To test logical (Boolean) values, compare to true first, then if not true and not NULL, it is false. Use **Y**, **YES**, **T**, **TRUE**, **1**, or **ON** to match true.

Verify the property type you are comparing. Some object properties, like **Owning User** for example, are reference properties, not strings.

Hidden required properties

In the Business Modeler IDE **Operation Descriptor** tab, it is possible to set a property to be both hidden (**Visible=false**) and required (**Required=true**). This allows for a rare scenario where a custom client programatically populates the property during the operation without prompting the user for input, and yet won't allow the operation to continue if the client is unable to populate the required property.

For operations that use an **Operation Descriptor**, the Active Workspace XML rendering template (XRT) processor does not automatically fill in all missing required properties, nor does it display any properties which are hidden by the **descriptor**, even if they are requested by the operation's XRT.

In this special case, Active Workspace will not allow the operation to continue because the required property cannot be populated.

To avoid this behavior, either change the property to be visible in the Business Modeler IDE **Operation Descriptor** tab, or remove the property from the operation's XRT.

nameValueProperty

Name-value properties are a specialized form of the table property designed for name-value pairs.

© 2025 Siemens

To display a name-value property in Active Workspace, use the **nameValueProperty** tag instead of the **objectSet** tag in your custom summary style sheet.

Attributes

name (required)

The name of the name-value property you want to display.

source (optional)

Specifies the related object from which to retrieve the name-value property. You must use dynamic compound properties to specify a source.

sortBy (optional)

Specifies the object property to sort the set of objects by prior to rendering. If this is not specified, the rows will be displayed in the order they were created.

sortDirection (optional)

Specifies the direction in which the set of objects on the **nameValueProperty** should be sorted.
Valid values are **ascending** or **descending**.
The default value is **ascending**.

enablePropEdit (optional)

Set this to **false** to disable the user's ability to automatically start editing on this table. While this is disabled, If the user wants to make changes to values on this table, they must start editing manually.

maxRowCount (optional)

The maximum number of rows displayed in the **nameValueProperty** regardless of available vertical space.

The <property> element

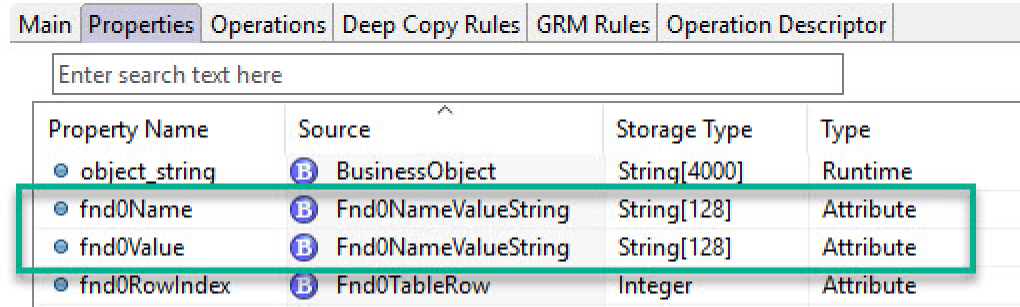
Use the **<property>** element inside the **<nameValueProperty>** element. You *must* use the **fnd0Name** and **fnd0Value** properties.

```
<section title=...>
   <nameValueProperty
name="myNVProp">
      <property name="fnd0Name"/>
      <property name="fnd0Value"/
>

   </nameValueProperty>
</section>
```

OOTB name-value pair properties use **fnd0Name** and **fnd0Value**.

© 2025 Siemens



Supported style sheets

This tag is supported on the following types of style sheets:

Summary

Examples

Example:

This example shows a name-value property on the selected object.

```
<nameValueProperty name="aw2_NameValue_Pair">
  <property name="fnd0Name"/>
  <property name="fnd0Value"/>
...
</nameValueProperty>
```

Example:

This example uses DCP to show a name-value property on a related object.

```
<nameValueProperty source="REF(IMAN_specification,AW2_NameValueRevision)"
               name="aw2_NameValue_Pair">
  <property name="fnd0Name"/>
  <property name="fnd0Value"/>
...
</nameValueProperty>
```

SaveAs new and naming rules

The **ItemID** and **ItemRevID** properties are not displayed by default on the **SAVEASRENDERING**.

If you are using multiple naming rules for either of these properties, they must be added to the style sheet.

© 2025 Siemens

```
<property name=”items_tag:item_id” />
<property name=”item_revision_id” />
```

Add the ability to send a newly created business object to a workflow

As an administrator, you can add the ability to send a business object to a workflow while creating the business object. To do so:

1. In Business Modeler IDE, locate the required business object, for example, **ItemRevision**.

2. In the **Main** tab, go to the **Business Object Constants** tab.

3. Locate **Awp0EnableSubmitForCreate** and set the value to **true**.

4. In My Teamcenter, locate the XML rendering style sheet for the business object. For example, locate the **Awp0ItemCreate** style sheet for the **ItemRevision** business object.

5. In the **Viewer** tab, add the following property to the style sheet:

   ```
   <property name="revision:awp0ProcessTemplates"/>
   ```

6. Click **Apply**.

7. In Business Modeler IDE, locate the business object, click **Operation Descriptor→CreateInput**, and verify that the **awp0ProcessTemplates** property is added.

   This ensures that users can submit business objects to a workflow while creating the objects.

Displaying form properties using a style sheet

In Active Workspace, you can create a new form XML rendering style sheet to render only those form properties you want to display.

There are three possible property display configurations for form style sheets:

- To display only the properties belonging to that form type, use the following tagging:

  ```
  <all type="form"/>
  ```

- To display the properties of the form type and include properties from objects in the form type's hierarchy, use the following tagging:

  ```
  <all type="property"/>
  ```

  By default, if there is no style sheet defined for the form type, the default style sheet is loaded, which includes the configuration for including all properties, for example:

© 2025 Siemens

```
<rendering>
  <page titleKey="tc_xrt_Summary">
     <all type="property"/>
  </page>
</rendering>
```

- To display only the properties defined in the **page** tag, use tagging like the following:

```
<page>
    <property name="property-1"/>
    <property name="property-1"/>
    <property name="property-1"/>
</page>
```

> Tip:
>
> For examples of form style sheets for Active Workspace, see the **Awp0FormSummary** and **Awp0FormInfoSummary** XML rendering style sheet datasets. (They use the **AWC_Form.SUMMARYRENDERING** and **AWC_Form.INFORENDERING** preferences.)

Working with HTML panels in XRT

HTML panel in style sheets for Active Workspace

The **<htmlPanel>** tag supports URL and HTML content to be included in an XML rendering.

- **<htmlPanel>** can be specified as a child tag in **<page>**, **<column>**, and **<section>** tags.

  > Note:
  >
  > The **<htmlPanel>** tag is supported only by Active Workspace XML renderer.

- The URL and HTML content can have the value of properties of the currently selected object introduced into them. This technique is known as *data binding*.

Instead of embedding HTML directly into an XRT, it is possible to use the `<inject>` tag to refer to an HTML dataset instead. There are two methods of specifying which HTML dataset is to be used. The HTML dataset must contain only valid HTML code, with no XML style sheet tags.

In the example below, an HTML dataset exists with the name **myHTMLblock**.

There are two methods of specifying which dataset is to be used.

- Directly, by referring to the name of the HTML dataset.

```
<inject type="dataset" src="myHTMLblock"/>
```


© 2025 Siemens

- Indirectly, by referring to a preference which contains the name of the dataset.

  ```
  <inject type="preference" src="additional_page_contributions"/>
  ```

  Then create the **additional_page_contributions**preference, containing the value **myHTMLblock**.

If the preference contains multiple values, then each dataset will be located and injected in order.

> Caution:
>
> Uncontrolled JavaScript code included in the HTML panels can be used to exploit a security issue or other network policy violation. System administrators must exercise care to ensure the XML rendering preferences, datasets, and any WAR build changes are monitored and require DBA level access.

Specifying a URL

The **src** attribute is used to specify the fully qualified URL as the source of the content to display in a new **iframe**.

- The **src** attribute can be complete or can contain references to various properties in the currently selected object.
- You can specify multiple properties.

Example: simple static URL

To display the contents of the given URL within the **iframe**:

```
<htmlPanel src=”https://www.mycorp.com/info”/>
```

Example: include a property value in a URL

To display the contents of the given URL with the current value of the **item_id** property used as the ID in the URL query:

```
<htmlPanel src=”https://www.mycorp.com/info?
id={{selected.properties[‘item_id’].dbValue}}”/>
```

If the **item_id** property for the selected object is **Part 1234**, the final **<iframe>** tag is encoded as:

```
<iframe src=”https://www.mycorp.com/info?id=Part%201234”/>
```

> Note:
>
> The resulting URL is made *safe*—all characters are encoded to assure they are valid for a URL.

© 2025 Siemens

For example, any space characters in the property value for the object are encoded as **%20** in the final URL.

Specifying HTML content

HTML content in an **<htmlPanel>** entity can be rendered by using the **declarativeKey** property. In the following example code snippets, the **itemIdObjectStringView.html** declarative view will be inserted. Just like any other declarative view, there must be a corresponding declarative view model file along with any other supporting files that may be needed.

Note:

These snippets are provided for reference only, and are not designed to be production-ready code.

htmlPanel

```
<htmlPanel declarativeKey="itemIdObjectString"/>
```

itemIdObjectStringView.html

```
<aw-panel>
    <aw-label prop="data.objectString"></aw-label>
    <aw-label prop="data.itemId"></aw-label>
</aw-panel>
```

itemIdObjectStringViewModel.json

```
{
    "schemaVersion": "1.0.0",
    "imports": [
        "js/aw-panel.directive",
        "js/aw-label.directive"
    ],
    "data": {
        "itemId": {
            "displayName": "Item Id",
            "type": "INTEGER",
            "isRequired": "false",
            "dbValue": 1075,
            "dispValue": 1075
        },
        "objectString": {
            "displayName": "Object String",
            "dispValue": "Displaying object string value"
        }
    }
}
```


© 2025 Siemens

Passing properties to the panel

If your **htmlPanel** uses Teamcenter properties, you must specify those properties in the style sheet. For example, if you want to use the **item_id** and **object_name** properties, specify the following.

```
<htmlPanel declarativeKey="itemIdObjectString">
    <property name="item_id"/>
    <property name="object_name"/>
</htmlPanel>
```

Calling a Teamcenter service from an htmlPanel

To call a Teamcenter service from an **htmlPanel**, create a simple view and a view model to perform the service call as an action.

In the following code snipped example, the view defines two buttons; **checkItOut** and **checkItIn.** Only one button is displayed at a time because of the **visible-when** conditions. When these buttons are used, actions associated with the action attribute of the **aw-button** element is called. This action is defined in the view model to make the requested Teamcenter service call and will also display the messages associated with the service call's success or failure.

> Note:
>
> These snippets are provided for reference only, and are not designed to be production-ready code.

htmlPanel

```
<page titleKey="HtmlPanel Check In/out">
    <htmlPanel declarativeKey="HtmlPanelCallTcService"/>
</page>
```

htmlPanelCallTcServiceView.html

```
<aw-panel context="subPanelContext">
    <aw-button action="checkItOut" visible-
when="subPanelContext.selected.properties.checked_out.dbValue !== 'Y'">Check Out</aw-button>
    <aw-button action="checkItIn" visible-
when="subPanelContext.selected.properties.checked_out.dbValue === 'Y'">Check In</aw-button>
</aw-panel>
```

htmlPanelCallTcServiceViewModel.json

```
{
    "schemaVersion" : "1.0.0",
    "imports": [
        "js/aw-panel.directive",
        "js/aw-button.directive",
        "js/visible-when.directive"
```


© 2025 Siemens

```
    ],
    "data": {},
    "actions": {
        "checkItOut": {
            "actionType": "TcSoaService",
            "serviceName": "Core-2006-03-Reservation",
            "method": "checkout",
            "inputData": {
                "objects": [
                    {
                        "uid": "{{ctx.selected.uid}}",
                        "type": "{{ctx.selected.type}}"
                    }
                ]
            },
            "outputData": {
                "checkOutPartialErrors": "partialErrors"
            },
            "actionMessages": {
                "success": [
                    {
                        "message": "checkoutSuccess"
                    }
                ],
                "failure": [
                    {
                        "condition": "(ctx.selected &&
ctx.selected.length === 1)",
                        "message": "checkoutFailure"
                    }
                ]
            }
        },
        "checkItIn": {
            "actionType": "TcSoaService",
            "serviceName": "Core-2006-03-Reservation",
            "method": "checkin",
            "inputData": {
                "objects": [
                    {
                        "uid": "{{ctx.selected.uid}}",
                        "type": "{{ctx.selected.type}}"
                    }
                ]
            }
        }
    },
    "messages": {
        "checkoutSuccess": {
```


© 2025 Siemens

```
                "messageType": "INFO",
                "messageText": "Checkout Successful",
                "messageTextParams": [
                    "{{ctx.selected.length}}",
                    "{{ctx.selected.length}}"
                ]
            },
            "checkoutFailure": {
                "messageType": "ERROR",
                "messageText": "Failed to checkout",
                "messageTextParams": [
                    "{{ctx.selected[0].props.object_string.uiValues[0]}}",

    "{{data.checkOutPartialErrors[0].errorValues[0].message}}"
                ]
            }
        },
        "functions":
        {
        },
        "conditions":
        {
        },
        "i18n":
        {
        }
    }
```

Data binding

Data binding is a way to connect the current property values of an object model to attributes and text in an HTML content view.

A section of HTML to be replaced with some other value is enclosed in double braces, **{{***xxxx***}}**, with *xxxx* indicating a reference to a property in the current *scope* object.

Specialized HTML tags

In addition to standard HTML tags such as **<b>**, for bold text, and **
** (to force a line break), XML rendering can use new specialized tags to simplify the work to display and edit Teamcenter data. This new tag reduces the amount of HTML required to accomplish common tasks.

<aw-property>

The **<aw-property>** custom tag is used to simplify label and value display for Teamcenter properties. It also handles the editing of these properties when appropriate.

© 2025 Siemens

You can use the **<aw-property>** tag to display all supported Teamcenter property types including single and multiple values, images, object sets, and object references.

The **<aw-property>** tag supports these attributes:

- **prop**

  For labels and values, this required string attribute specifies the property to display. This attribute supports **data binding** value substitution.

- **hint**

  This optional string attribute specifies variations in the way a property is displayed.

  The valid values are:

  - **label**
  - **objectlink**

- **modifiable**

  This optional Boolean attribute specifies whether the property can be modified during edit operations. It applies only when the property is otherwise editable.

<aw-frame>

The **<aw-frame>** custom tag is used to simplify displaying URL contents in an **iframe**. It supports a single **src** attribute.

The **<aw-frame>** attribute inserts HTML structure and CSS styling to correctly display the **iframe** using the full width and height available in the page, column or section in which it is placed.

There are limitations on what can be shown in an **iframe**:

- Not all external URLs can be used within an **<iframe>** tag.

  Some sites detect this tag and prevent their content from being displayed within an **<iframe>** tag. This is a way sites control content display.

- Some browser, site, and network settings prevent some scripts from running if they come from a location other than the root location of a page.

  This capability, also called *cross-site scripting*, is a potential source of network attack.

  This tag supports the following attribute:

© 2025 Siemens

src

This required attribute specifies the URL to be displayed in the **iframe**.

The **src** attribute supports data binding value substitution.

Specifying CSS styling

Creators of HTML content are free to specify their styling in their application. However, all of the CSS styling selectors provided with Active Workspace are available for use in HTML content contributed by the **<htmlPanel>** tag. Use these existing styling selectors to save time and ensure UI consistency.

© 2025 Siemens

© 2025 Siemens

3. Changing Active Workspace behavior

The Active Workspace customization process

The complete customization process of Active Workspace requires knowledge of the following:

The gateway architecture

Understand where the Active Workspace site files are stored, where your modifications and customizations are stored, and how the various parts of Active Workspace work together.

The Active Architect user interface

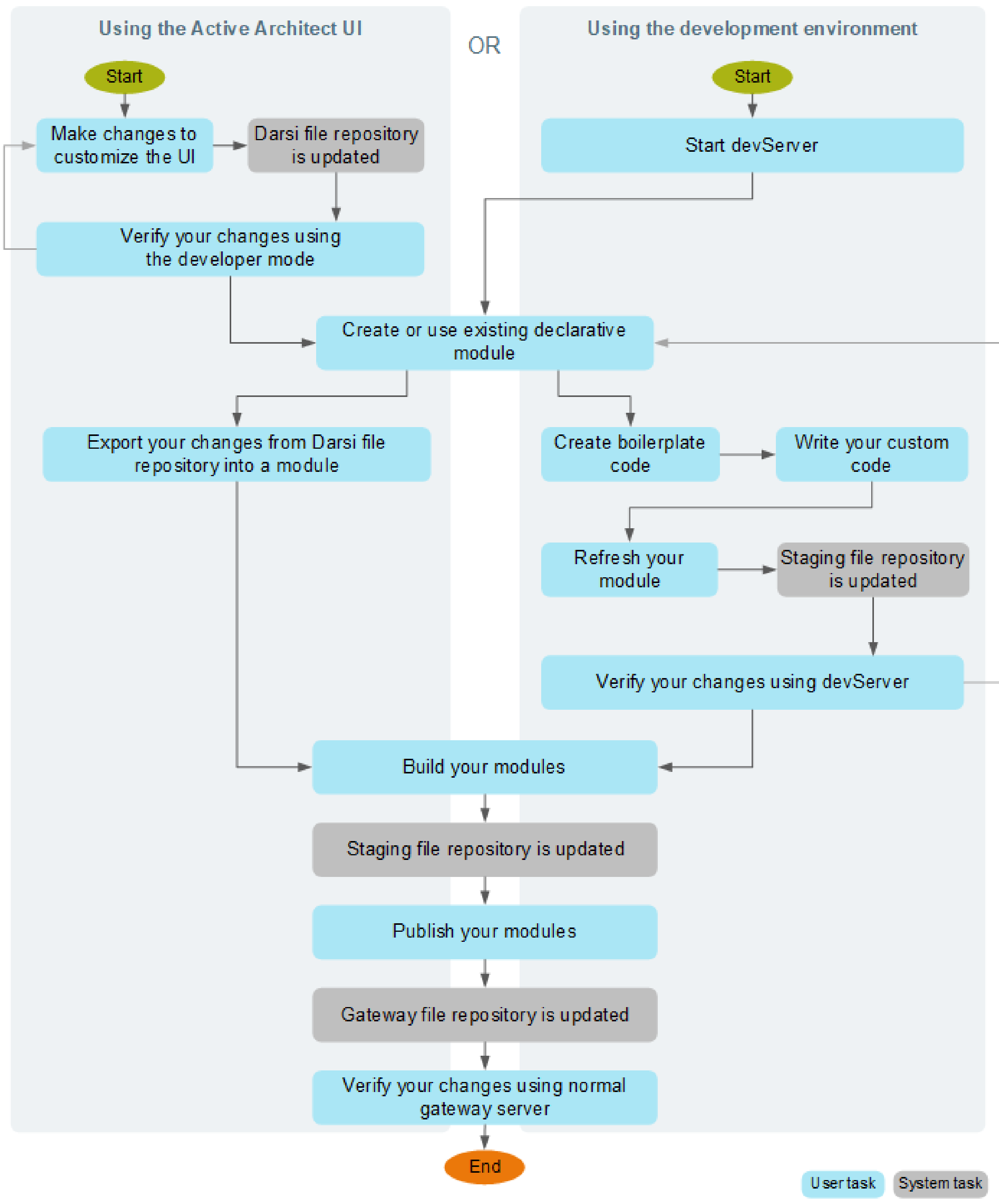
You can easily manage the provided commands and your custom commands using the command, action, and panel builders available in the Active Architect workspace.

The Active Workspace development environment

All your other customization needs are available from the command-line development environment.

You *must* use the command-line development environment to build and publish *all* your Active Workspace customization changes.

© 2025 Siemens

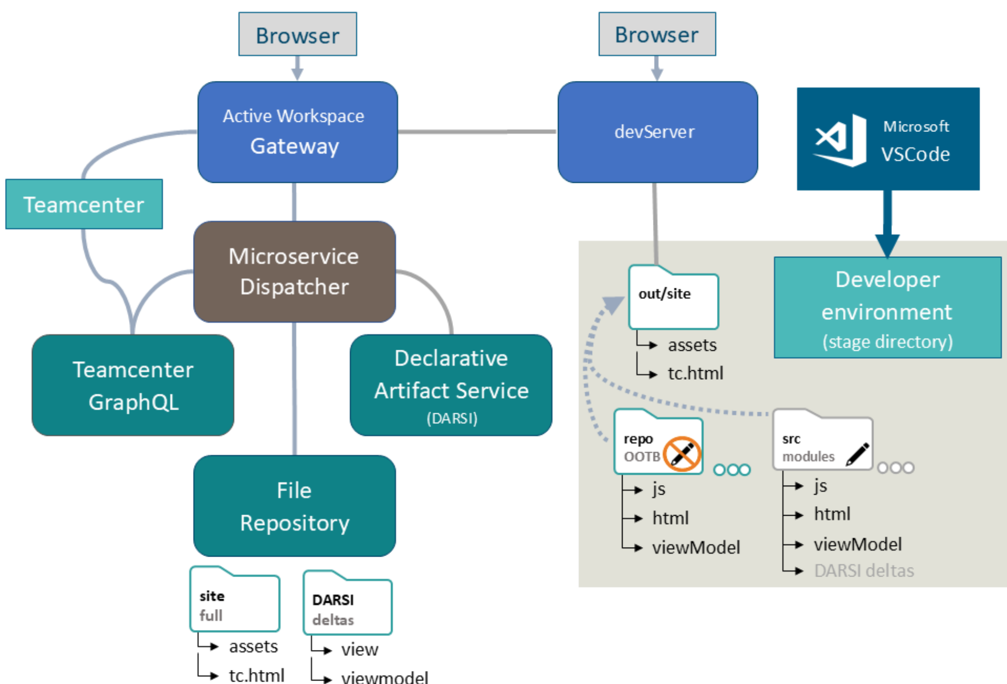


© 2025 Siemens

The Active Workspace gateway architecture

The Active Workspace gateway architecture

Using the Active Workspace development environment with the microservices framework requires knowledge of how the gateway architecture is designed and how its components work together. The following topics break down this architecture based on what is used during certain operations.



Active Workspace gateway architecture

The left portion of the diagram shows components of the microservices dispatcher and the gateway.

Active Workspace Gateway

Provides the reception area into Active Workspace. From here, users' requests are directed to various other microservices or locations depending on your site's configuration.

> Note:
>
> Not all communication goes through the microservices yet. Some functionality still communicates directly to Teamcenter.

File Repository

© 2025 Siemens

A web-based file storage system. It stores both the baseline Active Workspace application web site and the declarative artifact service overlays in two separate repositories.

Declarative Artifact Service

Manages any changes made by Active Architect's UI builder tools, like **Command builder**, **Panel Builder**, and so on. The changes are stored in the file repository separately from the production site. When you use the development mode of Active Workspace, those changes are overlayed onto the production site.

Teamcenter GraphQL

An assistant for Active Workspace which helps locate and consolidate data.

Teamcenter

The entirety of the 4-tier Teamcenter architecture. The Active Workspace application communicates with Teamcenter as a 4-tier client.

Active Workspace developer environment

The right portion of the diagram shows components of the developer environment.

The stage directory

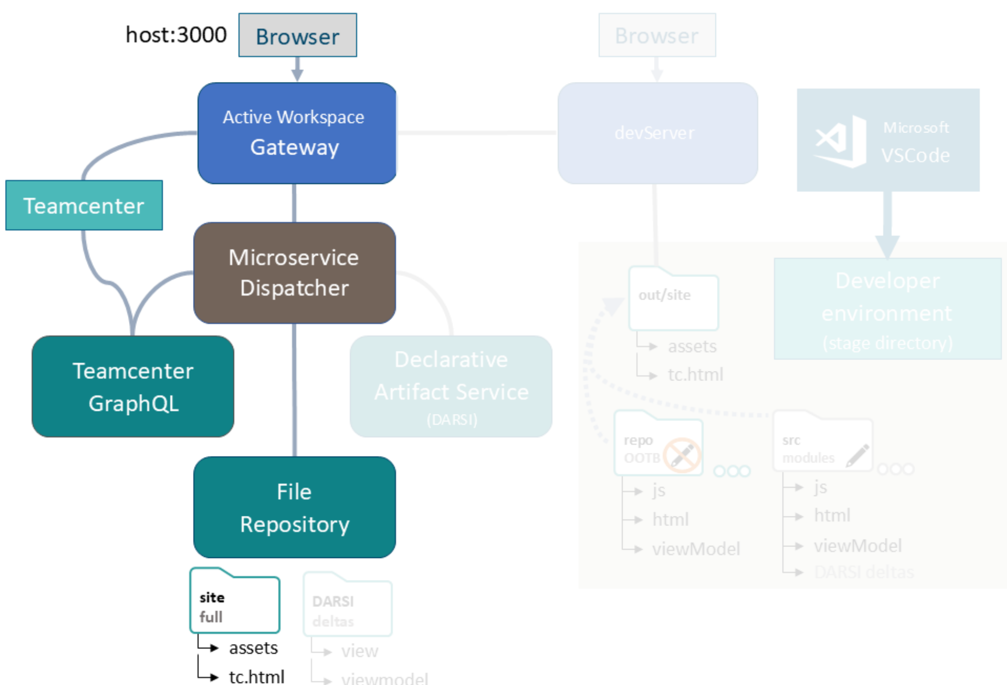
The **STAGE** directory, which is used for local development and builds.

The development server

This web application server will incorporate the changes you make in the development environment.

© 2025 Siemens

Using the production logon



What is happening?

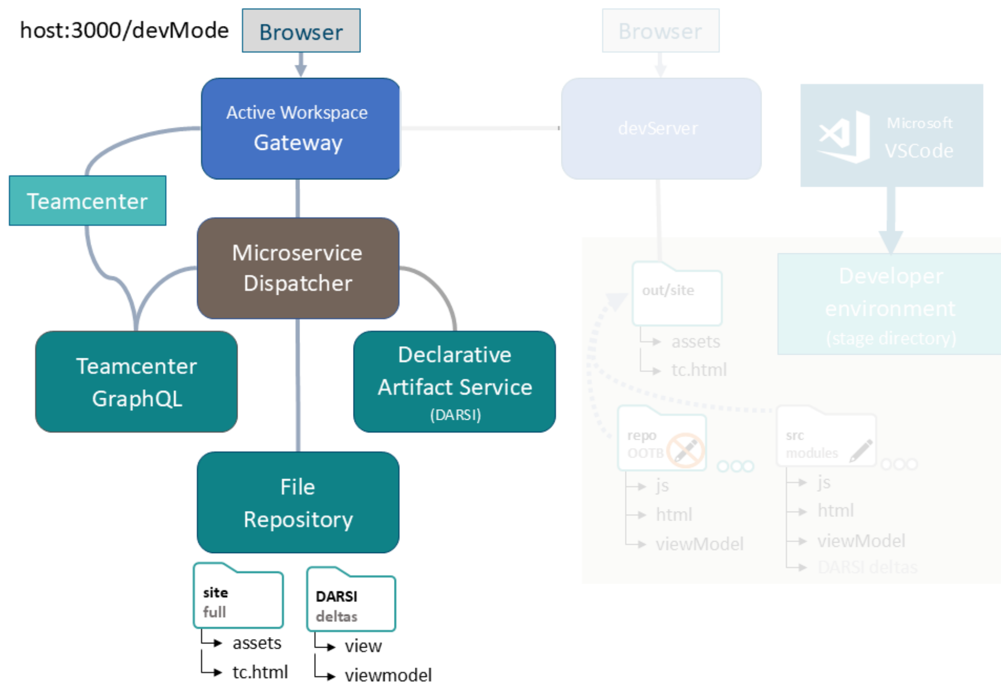
This is how your regular users will connect to perform their everyday work.

The user connects to the gateway service using port 3000. They experience the Active Workspace site as it exists in the file repository without seeing any changes from Active Architect or the local development site.

- The gateway retrieves the Active Workspace site from the file repository.
- Any changes managed by the declarative artifact service or created in your local developer environment are not displayed.

© 2025 Siemens

Using developer mode



What is happening?

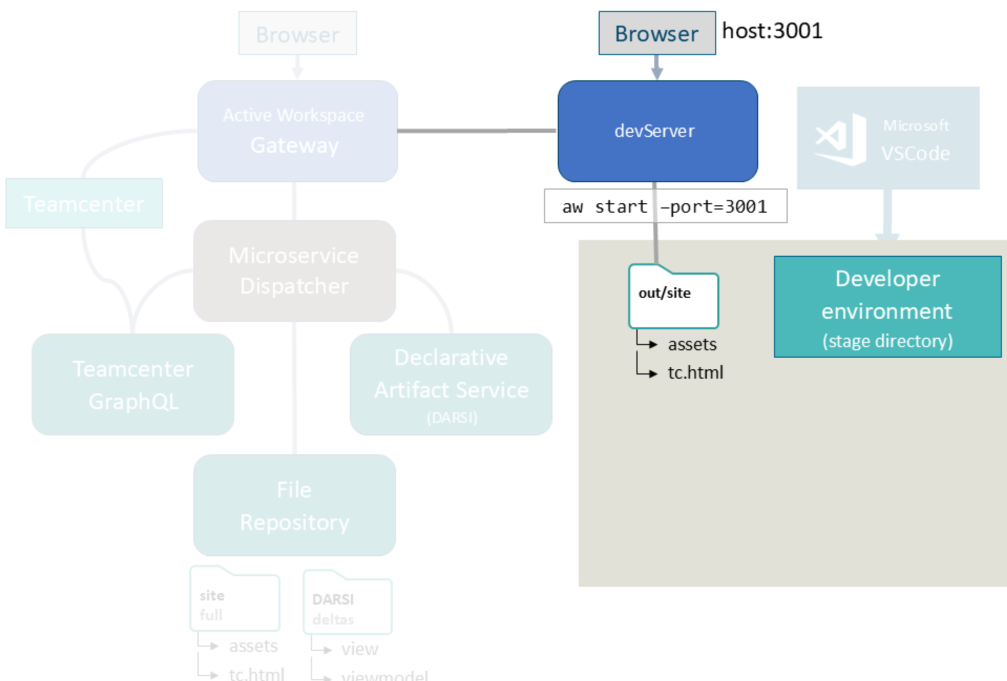
A user would connect using developer mode if they want to make or test changes made with the Active Architect workspace tools.

The user connects to the gateway service using port 3000, but adds **devMode** to the URL to activate *developer mode*. They experience the Active Workspace site as it exists on the file repository, plus any changes that have been made using the UI builder tools, like **Command Builder**, **Panel Builder**, and so on.

- The gateway requests the Active Workspace configuration from the declarative artifact service.
- The declarative artifact service retrieves and overlays its stored configuration deltas onto the full site.
- If the user has administrative privileges and has access to the **Active Architect** workspace they can use the UI builder tools to make changes to the UI.
- Any changes made to the UI using the UI builder tools are stored by the declarative artifact service in a separate section of the file repository as adds or deltas. No changes are made to the full site file repository.

© 2025 Siemens

Using the development server



What is happening?

Use this mode when you want to verify your local declarative modules. You will not see any current Active Architect changes stored in the Darsi repository.

The user connects to the development server service using the configured port (3001 shown). They experience the Active Workspace site as it exists on the local *stage\out\site* directory. This allows developers to make changes without impacting the full site served from the gateway.

Setup

- Run the **initEnv** script from the **STAGE** directory to initialize the environment.
- Set the listening port for the development server.

```
set PORT=3001
```

- Point to the gateway service.

```
set ENDPOINT_GATEWAY=http://gatewayserver:3000
```

- Use **aw start** to start the development server.

© 2025 Siemens

This process first compiles the source code (which may take a few minutes) and then runs the development server in a command-line window until you stop it with **CTRL-C**. This allows you to validate your changes without having to build and publish each time.

- Wait for the following to be displayed in the command-line window. Use the URL provided when you started the development server to connect to the Active Workspace application. In the command output, it looks like this:

  ```
  setupProxy: devServer running on http://localhost:3001
  ```

  If you change your configuration, the host and port may be different.

Stop any running developer server before performing a full build.

Use

- The gateway requests the Active Workspace site and configuration from the **siteDir** instead of the file repository.
- Use this setup when working with declarative modules. For example, creating a new theme, location, type icon registry, and so on.

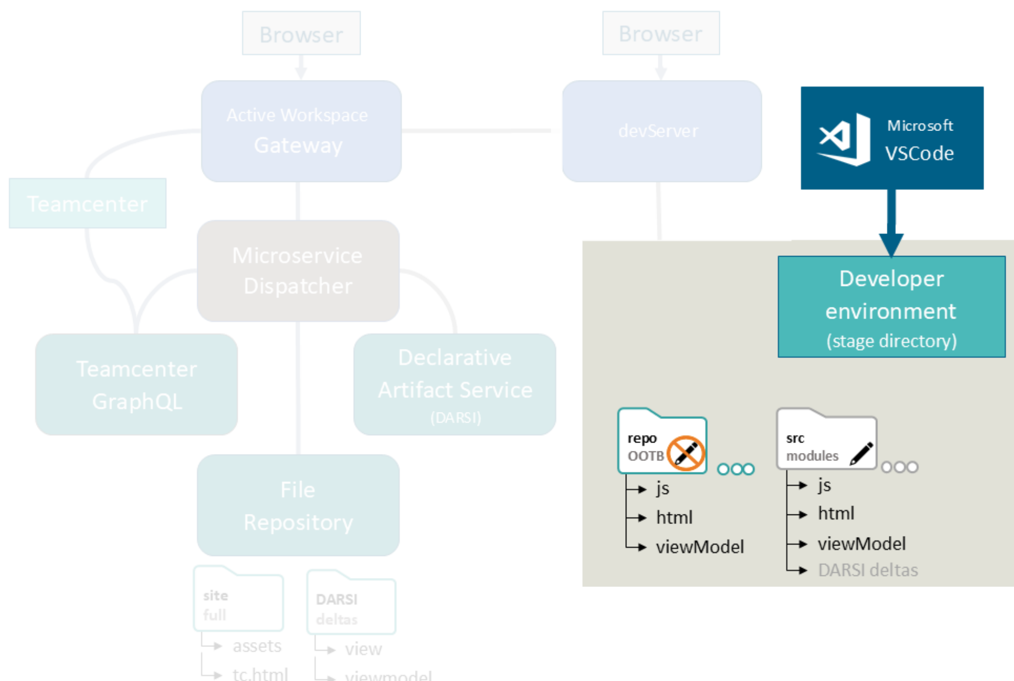
> Note:
>
> Do not use the **devMode** in combination with the *development server*.

This is hosted locally and it monitors the declarative code in the stage directory. Changes you make will be reflected immediately without needing to manually rebuild the application. It does *not* detect changes to build-related files, like *kit.json*, *module.json*, and so on, so new modules will not be detected.

You will still need to build and publish to the gateway when you are ready to implement your changes into the production environment.

© 2025 Siemens

Work with a declarative module



What is happening?

You can create and edit your declarative modules in this command-line environment. You have read access to the *repo* directory, which contains the OOTB source for Active Workspace for reference and builds. Even if you have write permissions to the *repo* directory, do not modify those files. Make all your changes in your custom modules.

Setup

- If you haven't done so already, prepare your gateway to **view your local site**.
- Optionally, run the **aw build** script to start with a clean build. You can run it directly, it runs the **initenv** script to configure the environment.
- Use a JavaScript project editor to work in your environment. Our examples use Microsoft's Visual Studio Code (not the same as Visual Studio).

Use

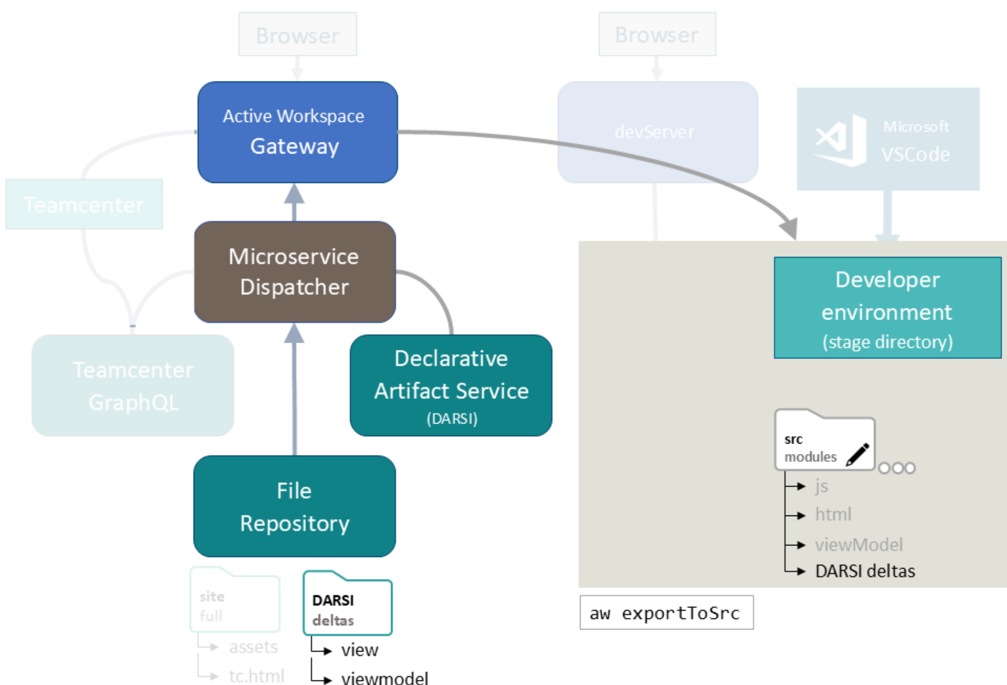
- Use your JavaScript project editor to load the entire **STAGE** directory.
- From the command-line, **generate a module** to hold your customizations.

© 2025 Siemens

Note:

Any changes you make to the declarative files will be automatically detected by the development server and you will see your changes when your browser refreshes.

Commit your UI Builder changes to your module



What is happening?

When you are done making your UI builder changes, you must export them from the declarative artifact service into your declarative module to be able to publish them This is like a commit for your UI builder changes, however they still only exist in the local module until they are published.

Setup

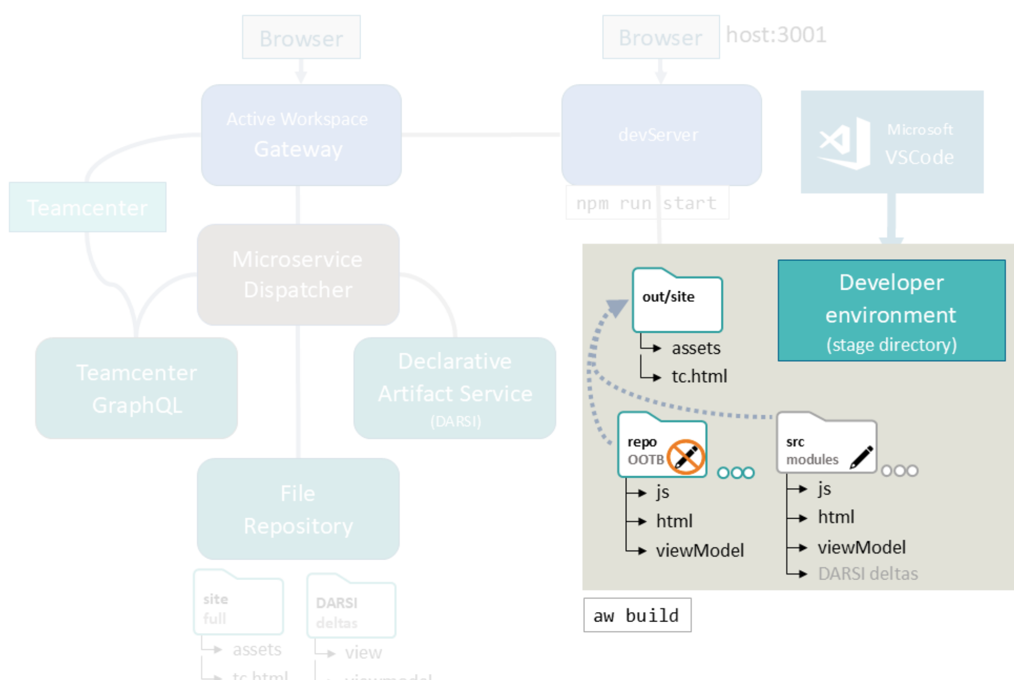
- If you haven't done so already, prepare a declarative module.
- Open a command prompt to the **STAGE** directory.
- If you are using Windows, run the initEnv script from the **STAGE** directory to initialize the environment.

Use

© 2025 Siemens

- From the command-line, use the **aw exportToSrc** script to contact the gateway service and copy the UI builder changes into your custom module.

Build your module



What is happening?

You can manually refresh or build your declarative modules along with the local source repository into the local site.

Setup

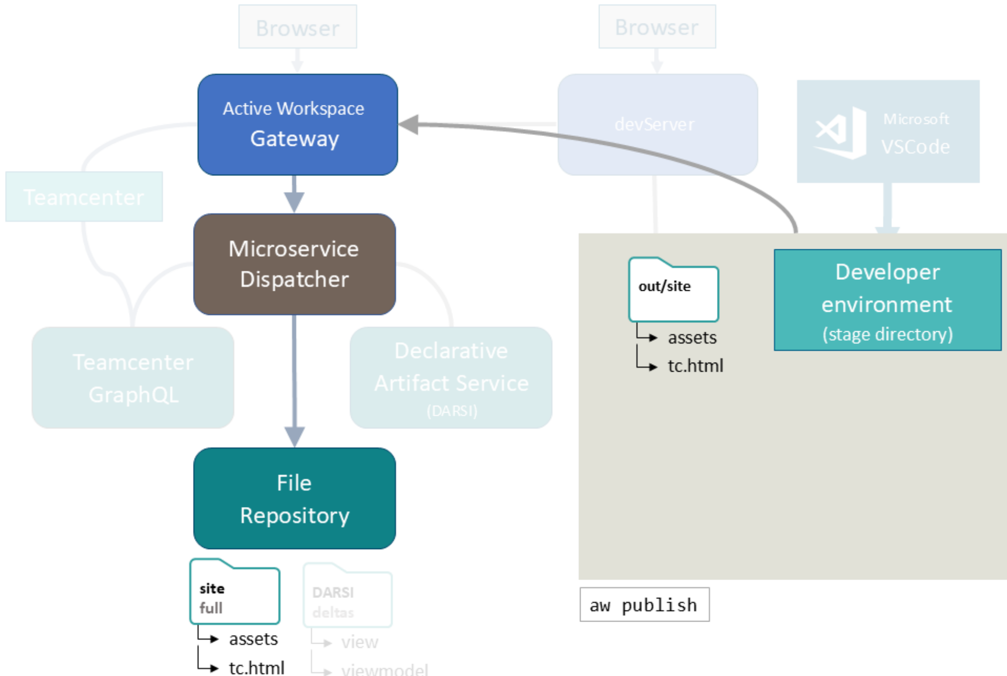
- If you haven't done so already, **prepare a declarative module** and possibly **export your UI builder changes**.
- Open a command prompt to the **STAGE** directory.
- If you are using Windows, run the **initEnv** script from the **STAGE** directory to initialize the environment.

Use

- From the command-line, **build** the Active Workspace application from the local source files into the local site.

© 2025 Siemens

Publish your local site



What is happening?

You can publish your local site to the production file repository. This is similar to a full commit for all your customizations.

Setup

- If you haven't done so already, **export your UI builder changes** and **perform a build** of your local files. This ensures that your all of your Active Architect changes and your local modules are incorporated into your local site.
- Open a command prompt to the **STAGE** directory.
- If you are using Windows, run the **initEnv** script from the **STAGE** directory to initialize the environment.

Use

- From the command-line, use the **publish** script to contact the gateway service and push the Active Workspace application from the local site to the gateway's file repository service.

© 2025 Siemens

Tip:

As a convenience you can use the **awbuild** script, which performs the following actions for you:

1. initenv
2. aw build
3. aw publish

The Active Architect workspace

What is Active Architect?

Active Architect is a collection of pages that allow you to easily create and manage commands, panels, and other configuration points using the Active Workspace interface. Your changes are stored in the Declarative File Repository as deltas and are displayed in real time while using developer mode. This helps improve your iteration times for UI design changes.

Caution:

Use dynamic configuration in your development and testing sites only. Siemens Digital Industries Software does not recommend using Active Architect or any other dynamic configuration capabilities in your production environment. Perform user acceptance testing before implementing your changes into a production environment.

Enable Active Architect

You can install Active Architect using either Deployment Center or Teamcenter Environment Manager (TEM).

Using Deployment Center:	Using TEM:
In the **Software** task, add the Teamcenter software to your environment. This automatically adds the required applications and components to your environment. Complete configuration of your environment and deploy it as described in *Installing Teamcenter Using Deployment Center*.	Create a Teamcenter environment that includes the following components as described in the TEM-based Teamcenter installation guides for Windows or Linux: • **Active Workspace client** • **Teamcenter Microservices** • **Active Workspace Gateway** Make sure your **Corporate Server** includes these features:

© 2025 Siemens

Using Deployment Center:	Using TEM:
	• **Active Architect Core** • **Active Architect→UI Builder**

Once installed, log into Teamcenter using the Active Workspace interface and switch to the Active Architect workspace.

Command Builder

Command Builder overview

You can use **Command Builder** to modify existing commands or create new ones. You can configure nearly every aspect of the commands for the Active Workspace interface.

Command Builder communicates with the **Declarative Artifact Service** to store your changes. New content is stored as an add, while changes you make to content for which you do not have write access are stored as a delta.

The **Command Builder** is displayed only when you are:

- Logged on as an administrative user, such as a member of the **dba** group.
- Using the **Active Architect** workspace.

The workarea toolbar

When using the **Command Builder** pages, you will have access to the work area toolbar.

Three commands on this toolbar allow you to easily perform necessary tasks instead of doing them manually.

Build Changes	Perform a build of the Active Workspace development environment site plus the Active Architect changes. This *does not* export changes to the development environment, like **exportToSrc** would. It only compiles using both sources.
Development Mode	Toggles the client to either use or not use **developer mode**.
Reset Deltas	Resets (deletes) *all* of the customizations that anyone has created using the Active Architect builders.

© 2025 Siemens

	Caution: You will lose *all* Active Architect customizations created by *all* users since the last time an **exportToSrc** was performed.

Commands

Command Builder Commands Toolbars Actions

Use this page to define a command's:

- Localized title
- Extended tool tip view name (and open the view)
- Command type
- Icon
- Placements
- Handlers

Toolbars

Use this page to view existing toolbars and to:

- Add commands to a toolbar.
- Delete commands from a toolbar.
- Change a command's priority on a toolbar.
- Decide if a command is relative to another command.

© 2025 Siemens

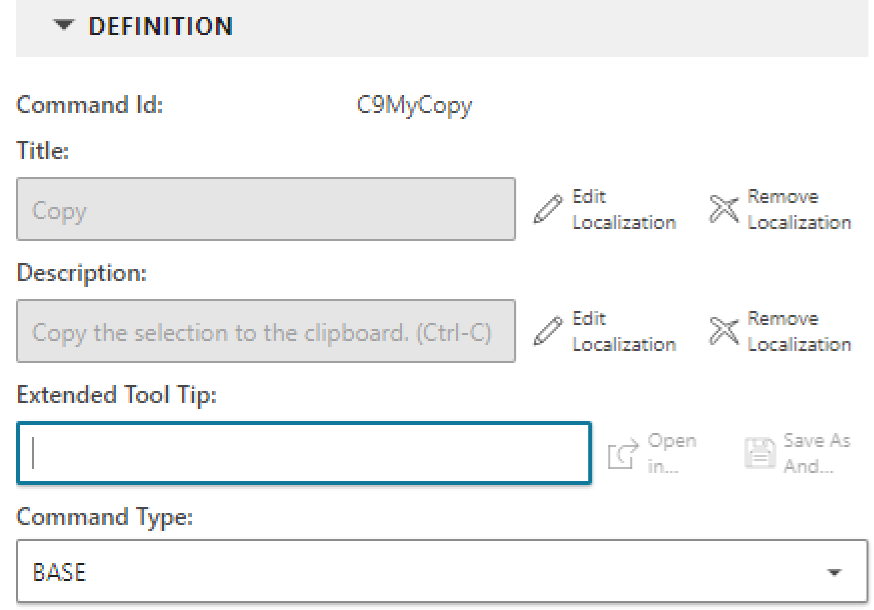
Actions

Use this page to:

- View and modify command actions already defined.
- Create new actions for use with commands.
- Edit action properties.

Using Command Builder

Definition



Use this section to define or edit the following components of a command:

Title

You can enter the title of the command in the field. This field displays the value for the current locale. Select **Edit Localization** to see the table of locales and their values.

Description

© 2025 Siemens

You can enter the description of the command in the field. This is the default extended tool tip text. This field displays the value for the current locale. Select **Edit Localization** to see the table of locales and their values.

Extended tool tip

The tool tips for commands in Active Workspace are full declarative pages, complete with a view and view model. The default view for extended tool tips displays the localized command name and description. If you want to create a custom view and viewmodel for your extended tool tip, enter the view name here. If it does not exist, the view and viewmodel will be created. There is no requirement on the name of the view other than it must be unique. We recommend using the command ID and add a custom ending.

In this example, you want to create a custom tool tip layout for the **Awp0Copy** command. You enter `Awp0CopyMyCustomTooltip` for the ID. The new view file name is `Awp0CopyMyCustomTooltipView.html`, and its view model file name is `Awp0CopyMyCustomTooltipViewModel.json`.

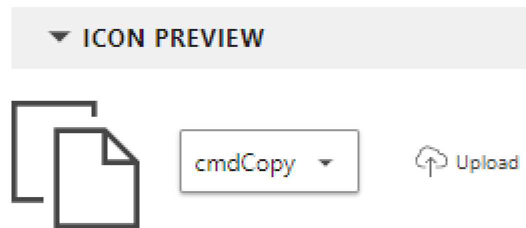
Choose **Save As And Replace** to copy an extended tool tip view and view model.

Use **Open in Panel Builder** to edit your custom view and viewmodel.

Command Type

Choose the type of command. The command types and descriptions are shown on the list.

Icon Preview



Either choose the command's icon from the list of existing icons, or select **Upload** to provide your own. Any icon you upload must conform to the requirements for Active Workspace command icons found in the **UI Pattern Library** on Support Center.

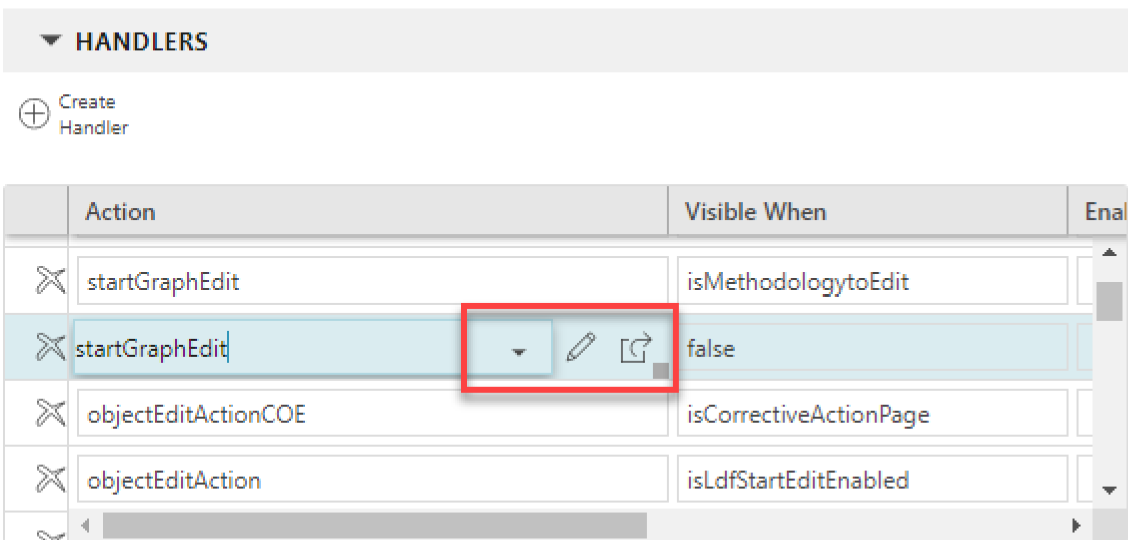
```
https://www.plm.automation.siemens.com/locale/docs/
```


© 2025 Siemens

Placements

Change, remove, or create the command **placements** here. Add the command to a command group, give it a priority, or place it relative to another command.

Handlers



Use this page to define the command **handlers**.

You can **Edit** the handler's JSON definition directly, or **Open in Action Builder** to **work in the UI**.

Select a row to see editing options for existing handlers.

© 2025 Siemens

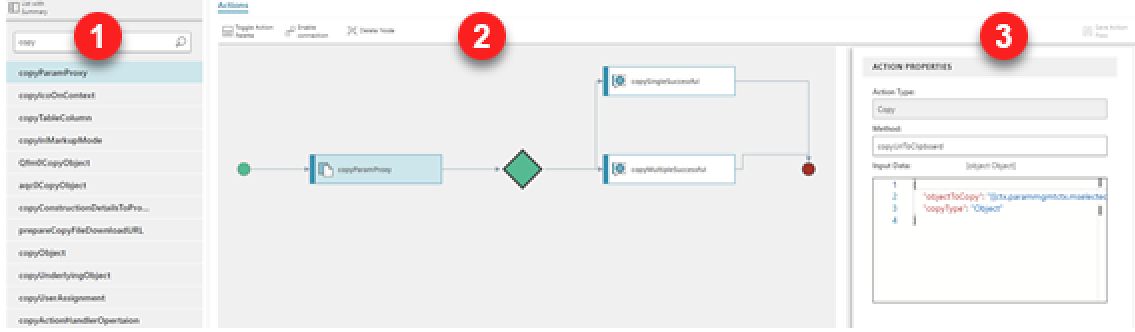
Creating a copy of a command

Use the **Save As** command to create a copy of an existing command. You are prompted to give your new command a unique command ID. All command metadata will be copied to the new command, like the icon, handlers, placements, and so on.

Using Action Builder

Use the **Actions** page of **Command Builder** or **Panel Builder** to define their behavior. If you make any changes, select **Save Action Flow** to save your changes.

The Actions panel

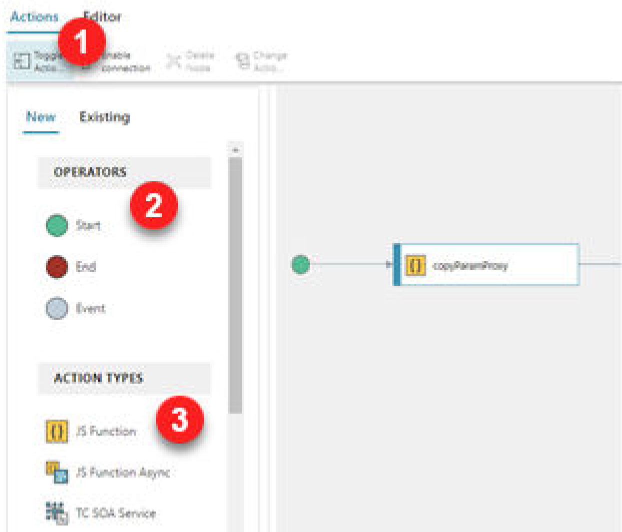


1. Find and select the action you wish to view.

2. Use the **Actions** viewer to scroll and zoom around the action flow. Select an action or operator.

3. Use the **Action Properties** panel to view the various properties of the selected action or operator.

The Action Palette

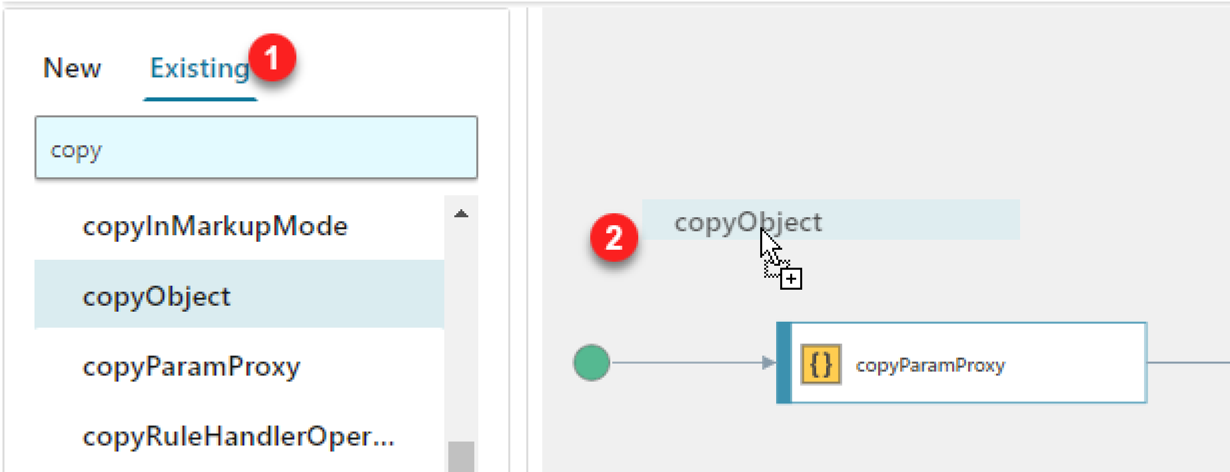


© 2025 Siemens

1. Select **Toggle Action Palette** to display or hide this panel.
2. Use **Operators** for splits and events as well as the start and end of the flow.
3. Use **Object Activities** to define behavior during the action flow. Drag new activities onto the action flow to create new nodes.

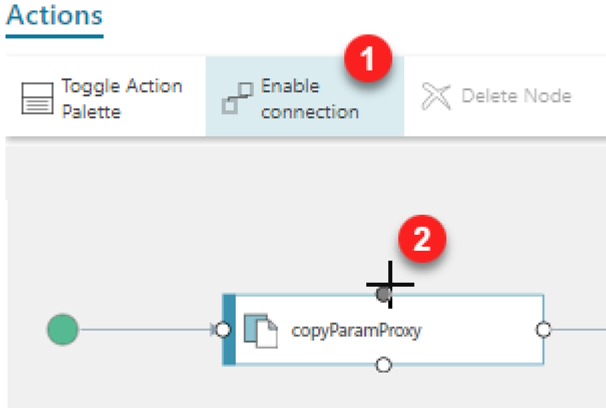
Adding existing actions

You can search for and add existing actions to your action flow.



1. Switch to the **Existing** tab to search for actions.
2. Drag an action into the action viewer.

Creating connections

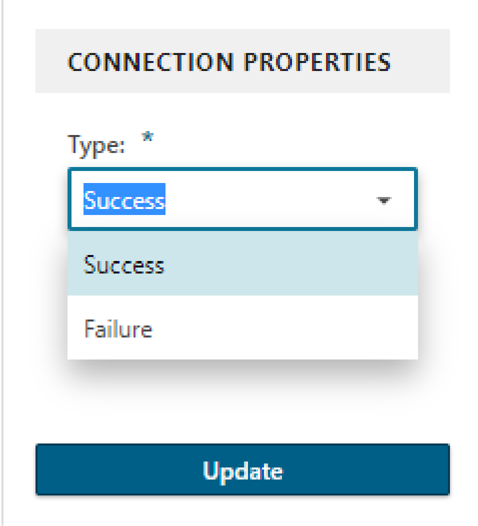


1. Toggle the **Enable connection** setting to create flow lines between actions.

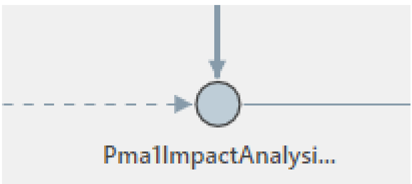
© 2025 Siemens

2. Draw connections between a circle of one action to a circle on another action.

By default, a success connection is drawn. You can change it to a failure connection by selecting the connection, and choosing **Failure** in the **CONNECTION PROPERTIES** panel.

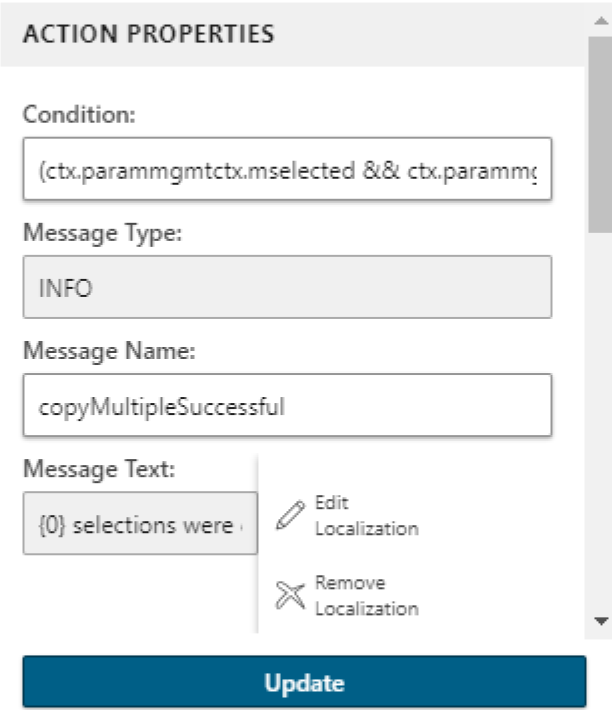


A **Success** connection is represented by a solid line, while a **Failure** connection is dashed.



© 2025 Siemens

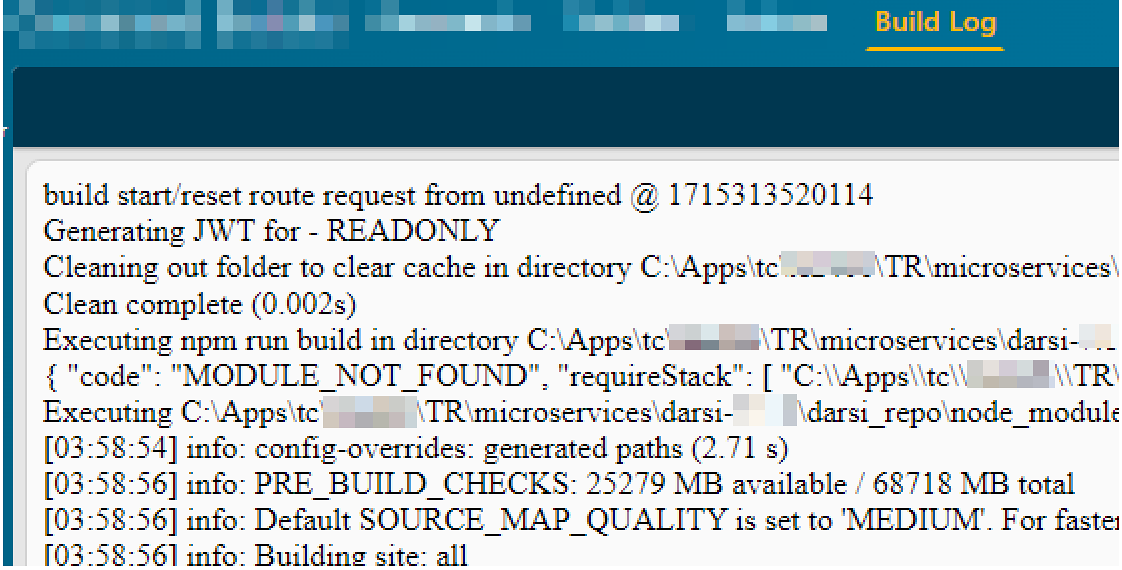
Editing properties



Use this panel to view or edit properties for the action selected in the action flow. After making changes, select **Update** to save the definition.

Review the build log

Use this tab to check on the status of the most recent build.



© 2025 Siemens

Panel Builder

Panel Builder overview

You can use **Panel Builder** to modify the layout of declarative panels.

Panel Builder communicates with the **Declarative Artifact Service** to store your changes. New content is stored as an add, while changes you make to content for which you do not have write access are stored as a delta.

The **Panel builder** is displayed only when you are:

- Logged on as an administrative user, such as a member of the **dba** group.

- Using the **Active Architect** workspace.

On all pages of this location, you can use the **Tools** panel to change to subpanel **Views** (and use the breadcrumb trail to return) and to work with the drag-and-drop **Elements**.

Opening Panel Builder

Open the **Panel Builder** in one of several ways:

- Open it without a target panel and use it to follow your main UI choices.

- Create or modify command panels directly by command ID.

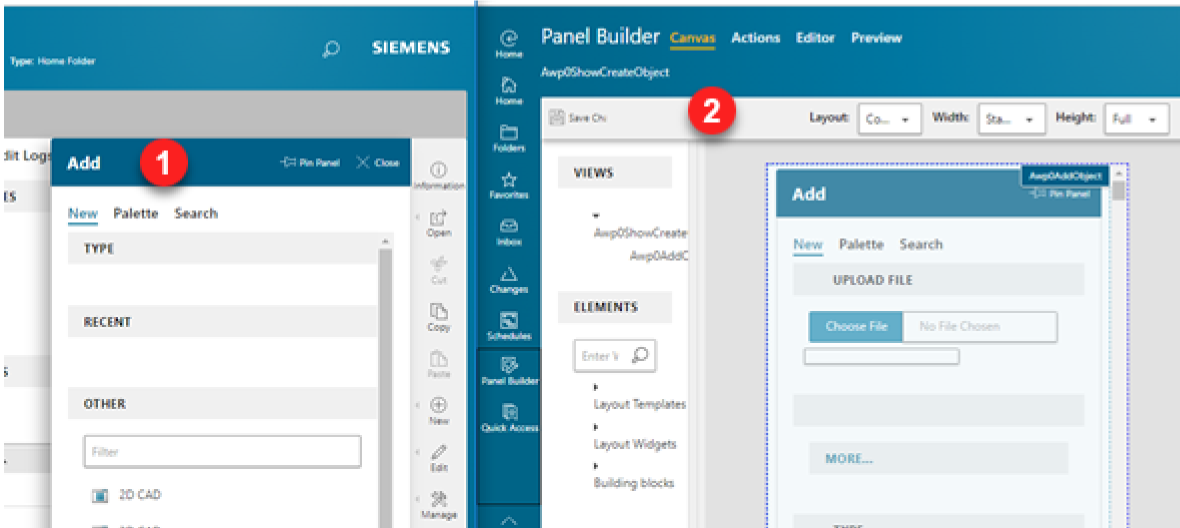
- Open it from the **Command Builder** while editing a command.

After making any changes, save your work by selecting **Save Changes** .

Follow the UI

In a duplicate browser tab, open the **Panel Builder** directly from the global navigation toolbar.

© 2025 Siemens



As you open the Active Workspace command panels in your original browser tab (1), the panel builder detects them and opens the panel definition for that command in the editor (2).

Open a command panel directly

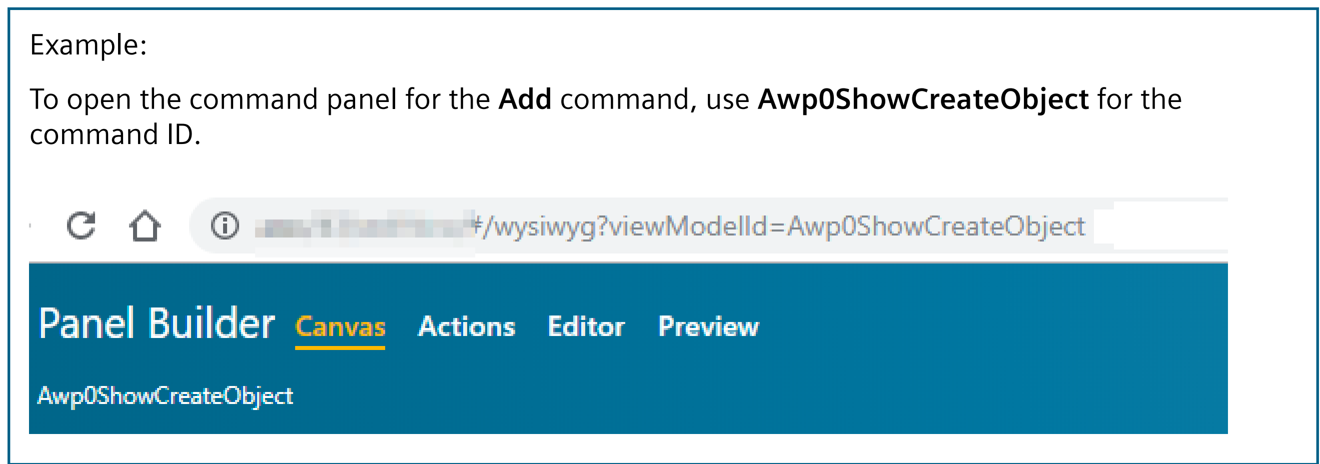
You can open a command panel directly if you know the command ID. After opening **Panel Builder**, add the following to the URL:

`?viewModelId=`**{commandID}**

If the command ID you enter does not have a command panel defined for it, **Panel Builder** creates one. If the command does not implement a panel, this new panel is not used.

Example:

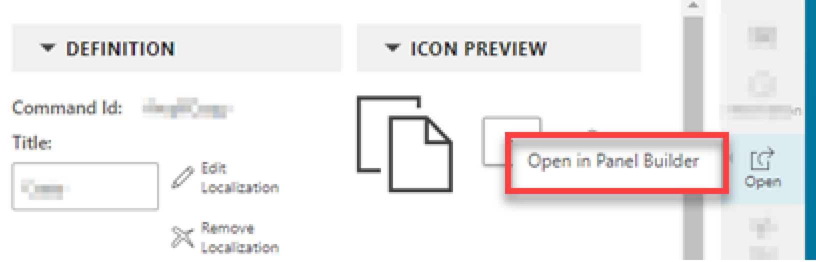
To open the command panel for the **Add** command, use **Awp0ShowCreateObject** for the command ID.



Open from the Command Builder

When using **Command Builder**, you can select **Open in Panel Builder** from the **Open** command group.

© 2025 Siemens



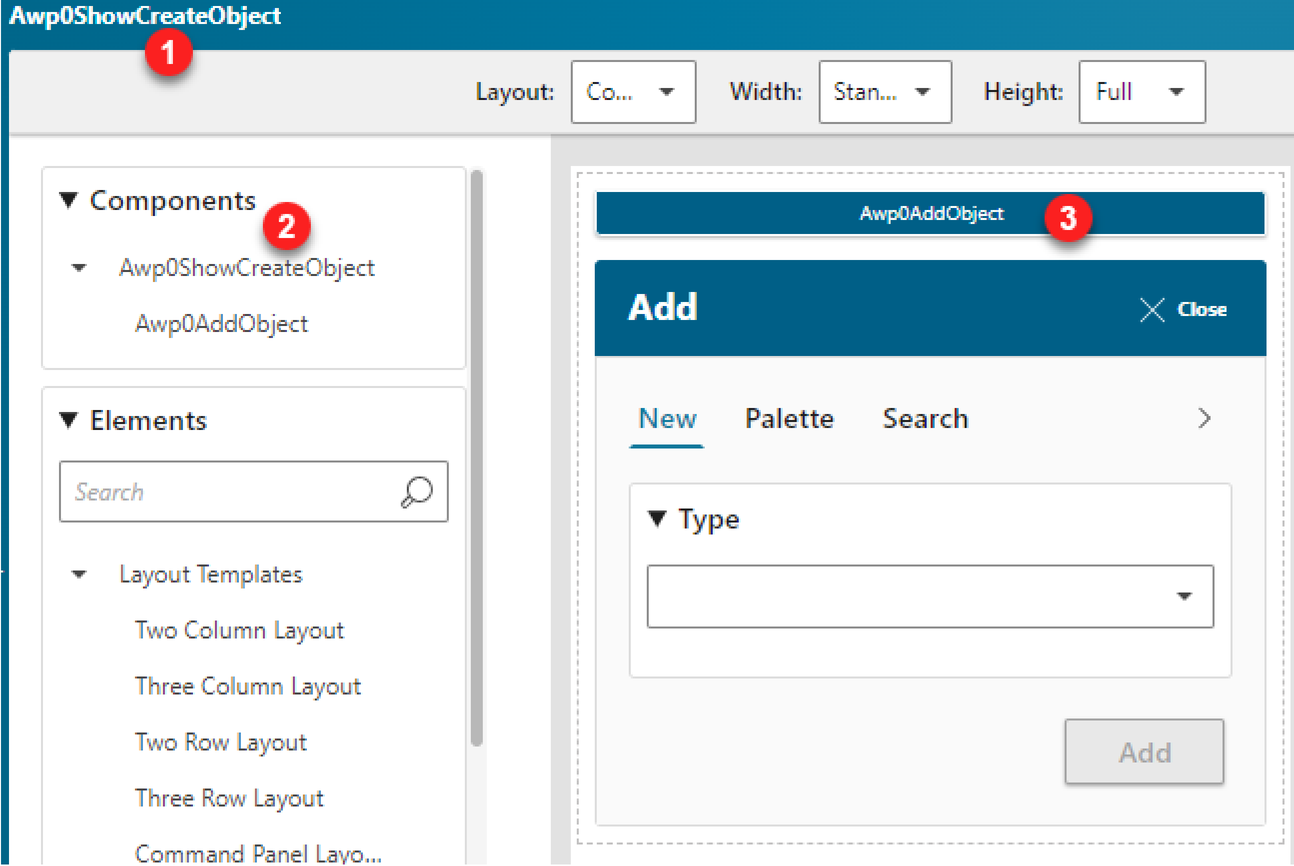
This sends the command panel view ID to **Panel Viewer** and opens it for you.

Panel builder canvas

Use the **Panel Builder** canvas to create or modify a panel using drag-and-drop.

Navigating nested panels

The **Panel Builder** supports nested panels. To navigate nested panels:

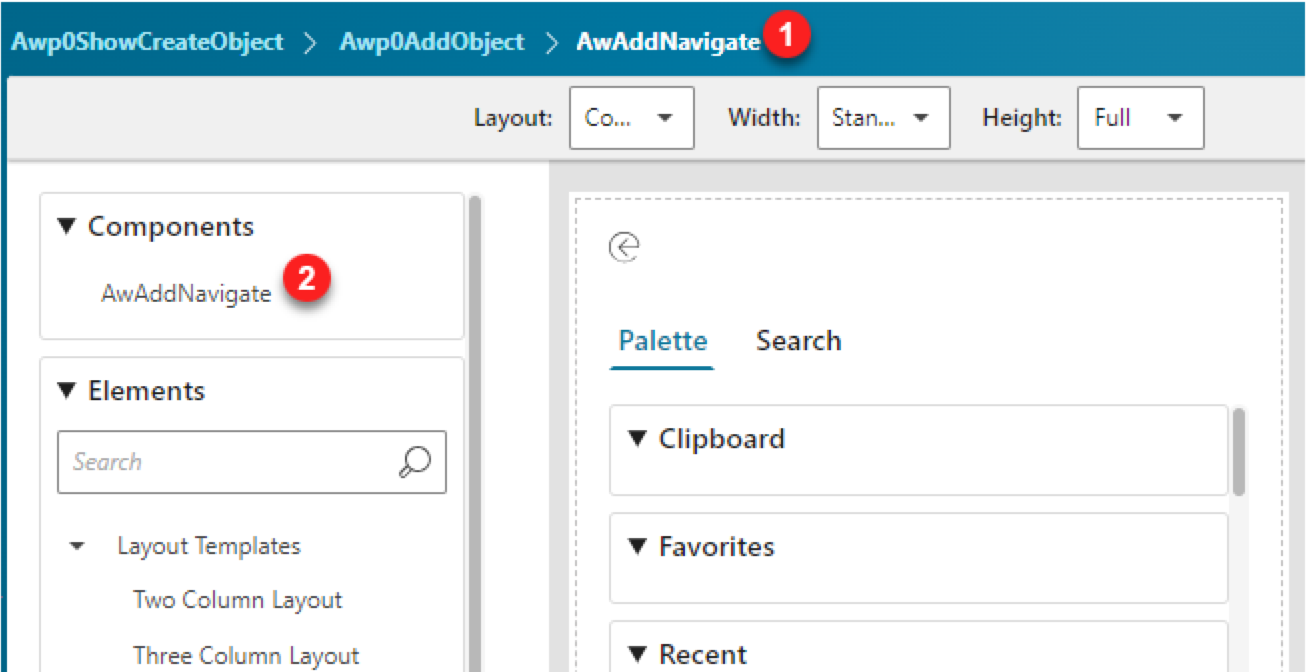


© 2025 Siemens

1. The breadcrumb shows you where you are in the nested panel hierarchy. In this example, you are at the top: **Awp0ShowCreateObject**.

2. The **Components** area shows you any subpanels that exists. Selecting one will change to that subpanel.

3. Subpanels are also shown in the main canvas area, and can be selected to change to that subpanel.



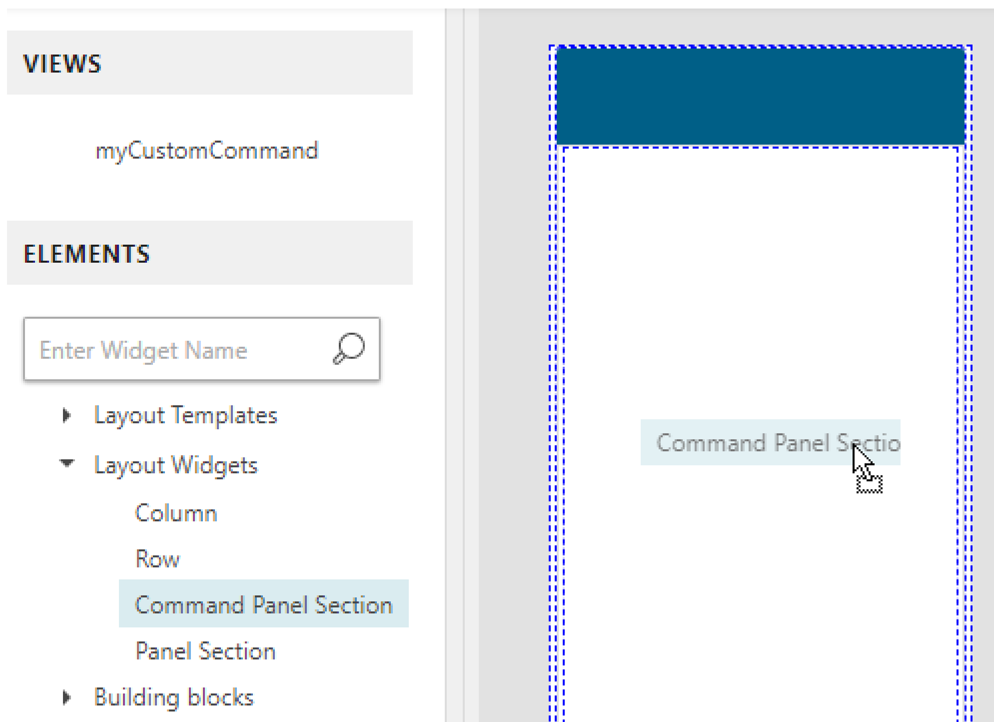
1. In this example, the breadcrumb shows that you are viewing the **AwAddNavigate** subpanel.

2. The **Components** area shows you that there are no subpanels.

Drag and drop elements

Drag **elements** onto the canvas to create new panel widgets, layouts, or properties.

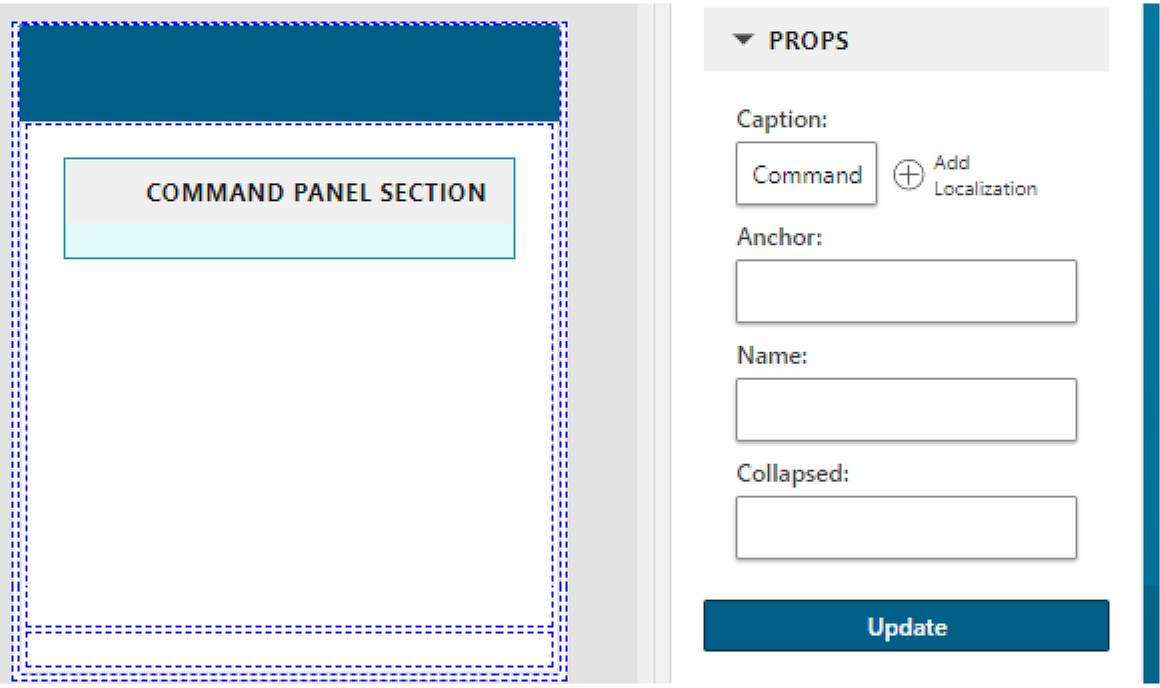
© 2025 Siemens



Save your changes using the **Save Changes** button.

Edit properties

Select a UI element on the canvas to edit its properties in the **props** panel.



© 2025 Siemens

Panel builder editor

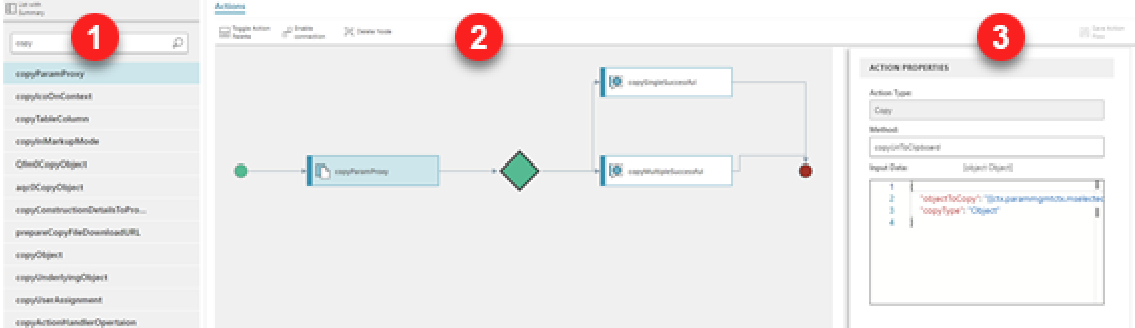
Use this page to work directly with the view and view model files.

- Breadcrumb
- Component tree navigation
- View
- View model

Using Action Builder

Use the **Actions** page of **Command Builder** or **Panel Builder** to define their behavior. If you make any changes, select **Save Action Flow** to save your changes.

The Actions panel



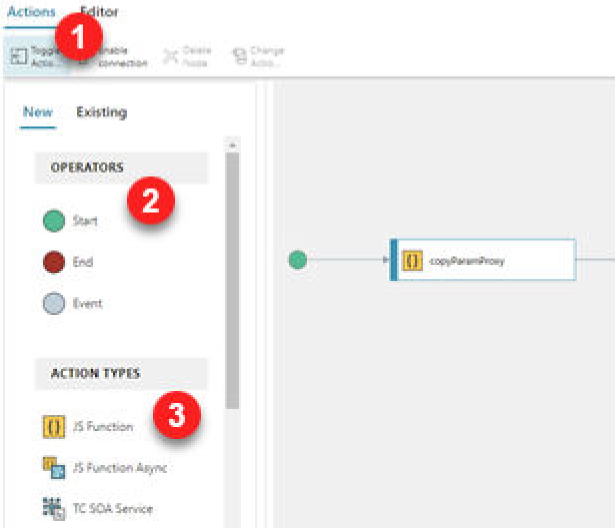
© 2025 Siemens

1. Find and select the action you wish to view.

2. Use the **Actions** viewer to scroll and zoom around the action flow. Select an action or operator.

3. Use the **Action Properties** panel to view the various properties of the selected action or operator.

The Action Palette



1. Select **Toggle Action Palette** to display or hide this panel.

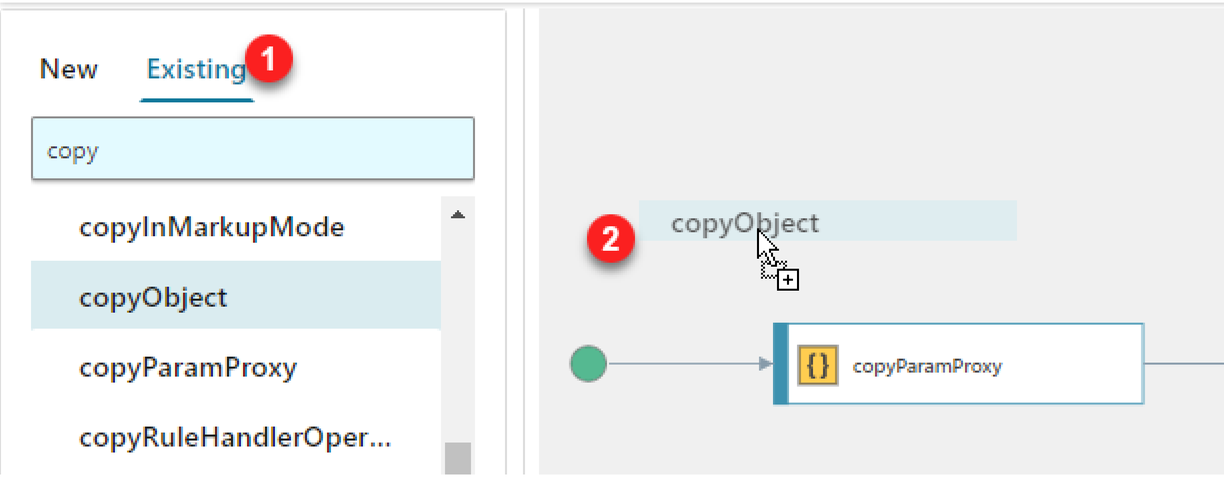
2. Use **Operators** for splits and events as well as the start and end of the flow.

3. Use **Object Activities** to define behavior during the action flow. Drag new activities onto the action flow to create new nodes.

Adding existing actions

You can search for and add existing actions to your action flow.

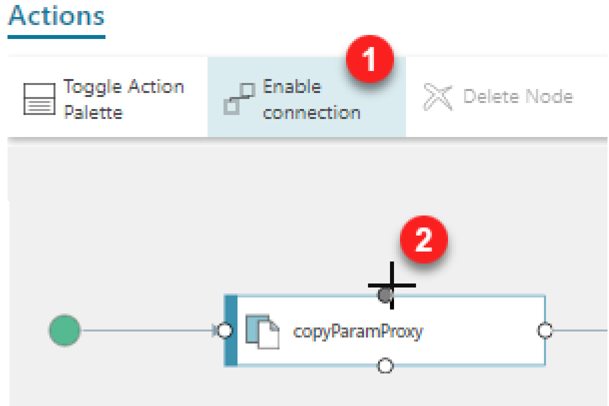
© 2025 Siemens



1. Switch to the **Existing** tab to search for actions.

2. Drag an action into the action viewer.

Creating connections

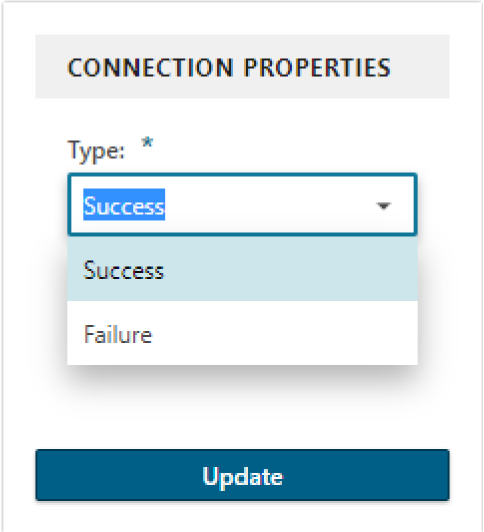


1. Toggle the **Enable connection** setting to create flow lines between actions.

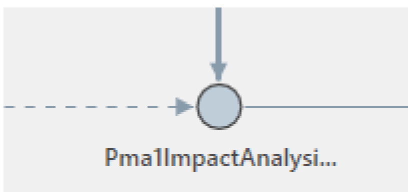
2. Draw connections between a circle of one action to a circle on another action.

By default, a success connection is drawn. You can change it to a failure connection by selecting the connection, and choosing **Failure** in the **CONNECTION PROPERTIES** panel.

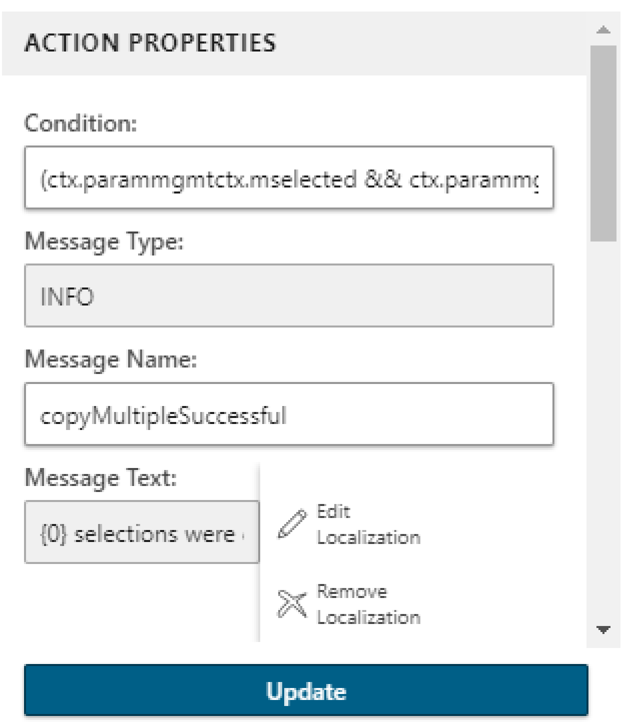
© 2025 Siemens



A **Success** connection is represented by a solid line, while a **Failure** connection is dashed.



Editing properties

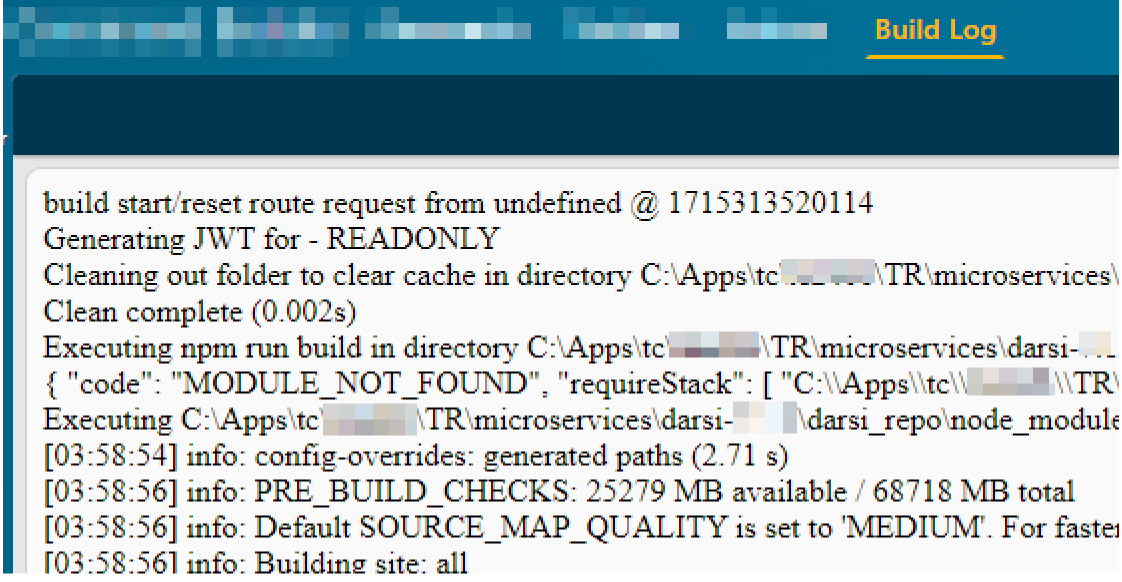


© 2025 Siemens

Use this panel to view or edit properties for the action selected in the action flow. After making changes, select **Update** to save the definition.

Review the build log

Use this tab to check on the status of the most recent build.



Configuration Builder

Configuration Builder overview

Use the **Configuration Builder** tabs to maintain the following Active Workspace configuration files.

- Routes
- Workspaces
- Indicators
- Type Icons

While in the **Active Architect** workspace, click **Config Builder** to open these pages.

Like the other active architect pages, your changes are stored as deltas in the Declarative Artifact Service (Darsi) and can be viewed dynamically using Development Mode.

© 2025 Siemens

Using configuration builder to manage routes

Create Page

Create a new page (**Location**), and a new tab (**Sub Location**) within Active Workspace. Since pages are essentially a container for tabs, when you create a new page, you must also create at least one tab with it.

You must provide a **Page ID** and a **Tab ID**. These IDs must be unique and cannot contain spaces or any symbols other than underscore and dash.

Create Tab

Create a new tab (**Sub Location**) for the currently selected page. The list of tabs is only displayed in the UI when there are more than one tab available for the page. For example, the **Configuration Builder** page shows 4 tabs.

You must provide a **Tab ID**. This ID must be unique and cannot contain spaces or any symbols other than underscore and dash.

Create Component

Create a new component (**Stand Alone Component**) for integration with hosted mode.

You must provide a **Component ID**. This ID must be unique and cannot contain spaces or any symbols other than underscore and dash.

Create a custom page and tabs

Create a custom page and two tabs using the default view and view model.

Prerequisites

You must:

- Be logged on as a user with administrative privileges.
- Switch to the **Active Architect** workspace.

Procedure

1. Open the **Config Builder** .

© 2025 Siemens

2. Switch to the **Routes** tab if it is not already active.

3. Switch to **Development Mode** if it is not already active.

4. Select **Create Page**.

5. In the **Create Page** dialog, fill in the following information:

   Page ID: MyCustomPage

   Tab ID: MyCustomTab1

   And then select **Create Page**.

   Your new page and tab is created and your new page is selected. Notice the created code for your page.

```
{
    "data": {
        "browserSubTitle": "MyCustomPage_Page",
        "headerTitle": "MyCustomPage_Page"
    },
    "abstract": true,
    "view": "AwSearchLocation",
    "parent": "root"
}
```

6. With your custom page still highlighted, select **Create Tab**.

7. In the **Create Tab** dialog, fill in the following information:

   Tab ID: MyCustomTab2

   And then select **Create Tab**.

   Your new tab is created and your new tab will be selected. Notice the created code for your tab.

```
{
    "data": {
        "label": "MyCustomTab2",
        "priority": 0,
        "nameToken": "/MyCustomTab2NameToken"
    },
    "params": {
        "secondaryCriteria": "*"
    },
    "parent": "MyCustomPage",
```


© 2025 Siemens

```
        "view": "MyCustomTab2Page",
        "url": "/MyCustomTab2"
    }
```

8. With your second tab highlighted, select **Open in PanelBuilder**.

9. Modify the search criteria to look for **spindle** instead of **hdd**.

```
"criteria": {
"searchString": "spindle"
},
```

10. Select **Save Changes** and confirm without changing the view/viewModel name.

11. Switch to the **default** workspace.

12. Change your URL to see your custom page and tabs.

```
http://host:3000/devMode/#/MyCustomTab1
```

Results

Your new page and two new tabs are created. However, you must add them to the workspaces where you want them to be displayed. The **default** workspace is an *inclusive* workspace and is only for testing.

Making search filters immutable

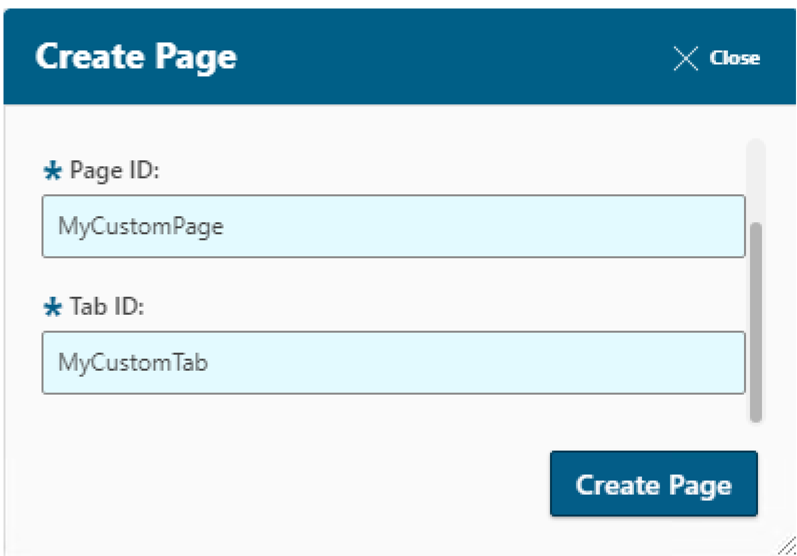
Use immutable search filters to prevent your custom page search criterion from being removed.

Modify the **searchState** section of your custom tab's view model definition. After creating your custom page and tab, open your custom tab in the panel builder.

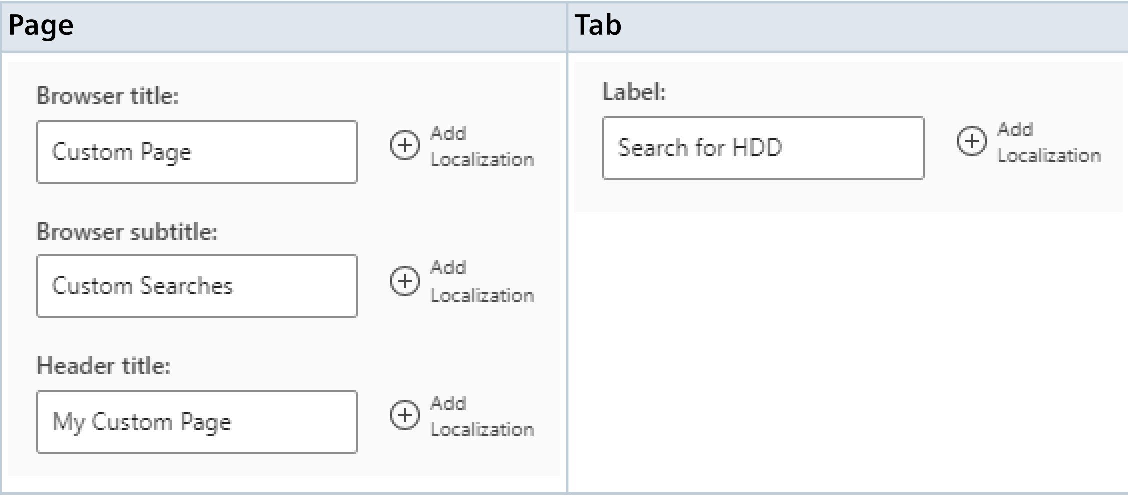
Process overview

1. Use the **Configuration Builder** to create a custom page and tab.

© 2025 Siemens



2. Optionally, click **Localization** to change the displayed titles for your page and tab.



3. Select your tab, and then click **Open in Panel Builder**.

4. Modify the **searchState** section of your custom tab's view model to include immutable search filters using the following code snippets.

   This is the **initialValues** content extracted from the **searchState** section. This example specifies that the **Item Reivision** type be pre-selected and can not be changed (immutable).

```
"initialValues": {
  "immutableSearchInfo": {
    "immutableSearchString": "hdd",
    "immutableSearchFilterMap": {
      "WorkspaceObject.object_type": [
        {
```


© 2025 Siemens

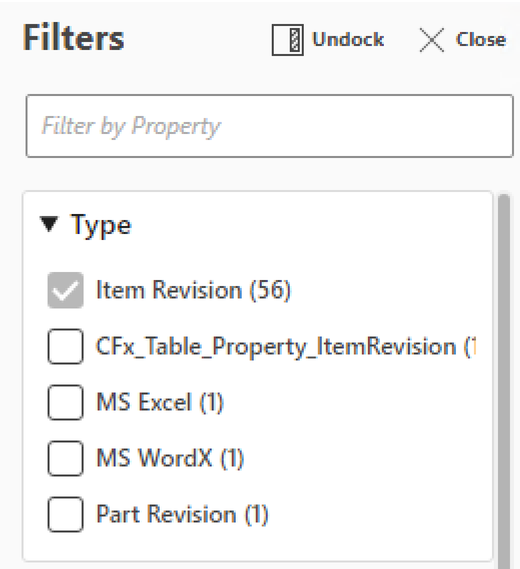
```
                "searchFilterType": "StringFilter",
                "stringDisplayValue": "Item Revision",
                "stringValue": "ItemRevision"
              }
            ]
          }
        }
      }
```

This is a sample **meta** block that can be used in the **searchState** section.

```
"meta": {
  "categories": [
    {
      "filterValues": [
        {
          "selected": {
            "field": "true",
            "type": "ViewModelProperty"
          }
        }
      ]
    }
  ]
}
```

5. Save your changes and ensure the developer mode is enabled.

Navigate to your new custom tab and open the filter panel to see your changes.



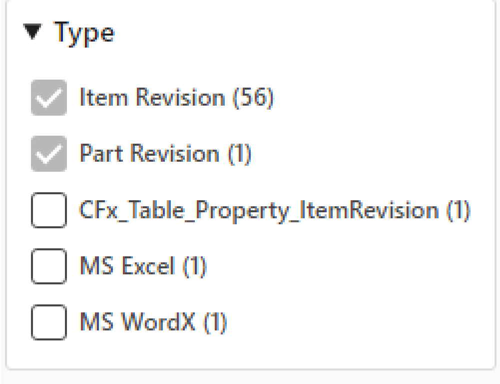
© 2025 Siemens

Your users can add additional filters, but they may not remove the ones that you have marked as immutable.

Multiple filters

You may specify multiple filters to be immutable. In this example, **ItemRevision** and **Part Revision** have both been specified.

```
"WorkspaceObject.object_type": [
  {
    "searchFilterType": "StringFilter",
    "stringValue": "ItemRevision"
  },
  {
     "searchFilterType": "StringFilter",
    "stringValue": "Part Revision"
  }
```




```
]
```

Using configuration builder to modify workspaces

Use the **Workspaces** tab to modify the definitions of workspaces. Learn about workspaces, including information about workspace definition files, how to create new workspaces, and map them to your organization.

JSON editor

Use the editor to manage the definition of a workspace, including its available pages and commands.

© 2025 Siemens

Localization

Create, modify, or remove the localized name of the workspace.

Modify an existing workspace

Change the **Author** workspace to display your custom page and tabs.

Prerequisites

You must:

- Be logged on as a user with administrative privileges.
- Switch to the **Active Architect** workspace.
- Complete the create a custom page and tabs task.

Procedure

1. Open the **Config Builder** .
2. Switch to the **Workspace** tab if it is not already active.
3. Select the **Author** workspace.
4. Add your custom tabs to the list of available pages.

```
"defaultPage": "showHome",
"availablePages": [
"MyCustomTab1",
"MyCustomTab2",
"aborted_tasks",
```

5. Select **Save**.
6. Switch to the **Author** workspace.
7. Change your URL to see your custom page and tabs.

```
http://host:3000/devMode/#/MyCustomTab1
```

Results

Your custom page and tabs are available to any user who uses the **Author** workspace.

© 2025 Siemens

Using configuration builder to register indicators

Configuration Builder Routes Workspaces Indicators Type Icons

Create Indicator

Create new visual indicator definition.

You must provide an **ID** for your indicator definition. This ID must be unique and cannot contain spaces or any symbols other than underscore and dash.

A boilerplate JSON definition structure is created for you. Fill in the details of your definition to **register your icons using conditions**.

Upload Icon

Upload your custom SVG indicator icon.

Delete

Remove the indicator definition.

Create an indicator

Create a visual indicator on the item revision type which is displayed based on the description.

Prerequisites

You must:

- Be logged on as a user with administrative privileges.
- Switch to the **Active Architect** workspace.

Procedure

1. Open the **Config Builder** .
2. Switch to the **Indicators** tab if it is not already active.
3. Select **Create Indicator**.
4. In the **Create Indicator** dialog, enter the following information:

© 2025 Siemens

ID: ItemRevision_indicator_demo

Definition

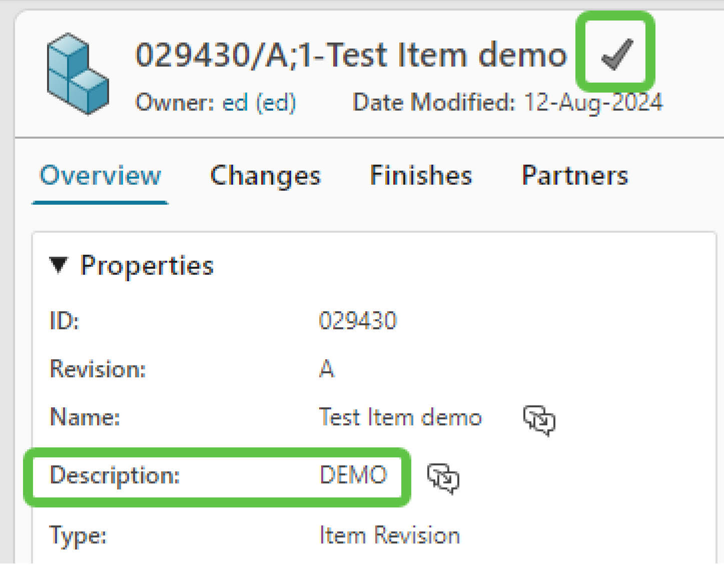
```
{
    "tooltip": {
        "showPropDisplayName": true,
        "propNames": [
            "object_desc"
        ]
    },
    "type": {
        "names": [
            "ItemRevision"
        ],
        "prop": {
            "names": [
                "object_desc"
            ],
            "condition":
"( localContext.vmo.props.object_desc.dbValue !== undefined &&
localContext.vmo.props.object_desc.dbValue === 'DEMO' ||
localContext.vmo.props.object_desc.dbValues !== undefined &&
localContext.vmo.props.object_desc.dbValues[0] === 'DEMO')"
        }
    },
    "iconId": "indicatorApprovedPassGray"
}
```

5. Select **Create Indicator**.
6. Switch to the **Author** workspace.
7. Open the **Explorer** tile.
8. Search for or create an **Item Revision** to which you have write access.
9. Modify the value of the **Description** field to **DEMO** and save your changes.

Results

When the **object_desc** field of an **ItemRevision** object contains a value of **DEMO**, the **indicatorApprovedPassGray** indicator will be displayed.

© 2025 Siemens



Using configuration builder to register type icons

This is only necessary if the name of the type icon does not match the name of the type. If you are only creating new icons for new business object types see adding custom type icons.

Create Type Icon

Create new type icon mapping definition.

You must provide an **ID** for your type icon mapping definition. This ID must be unique and cannot contain spaces or any symbols other than underscore and dash. A boilerplate JSON definition structure is created for you. Fill in the details of your definition to register your icons using conditions.

Upload Icon

Upload your custom SVG type icon.

Delete

Remove the type icon mapping definition.

Assign an icon to a type

If the name of your icon does not match the name of the business object type, you must assign it manually using the **Type Icons** page.

© 2025 Siemens

Prerequisites

You must:

- Be logged on as a user with administrative privileges.
- Switch to the **Active Architect** workspace.
- Have a custom business object deployed to your site.
- Have an **SVG** icon for your custom business object type.

Procedure

This activity has the following assumptions:

- Custom business object: C9TestItemRevision
- Custom SVG icon: typeCC024.svg (CCO)

1. Open the **Config Builder**.
2. Switch to the **Type Icons** tab if it is not already active.
3. Click the **Upload Icon** command.
4. Either drag and drop your icon into the **Drag and Drop file here** area, or click **Choose File** and pick your icon.
5. Click **Upload** to add your icon to the Active Workspace repository.

   The following system message is displayed. **Icon uploaded successfully**.
6. Click **Create Type** to create a type icon registration.
7. In the **Create Type Icon** panel, enter the **ID** and modify the **Definition**.

   ID: **C9TestItemRevision**

   Definitions

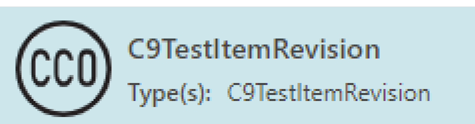
   ```
   {
       "type": {
           "names": [
               "C9TestItemRevision"
   ```


© 2025 Siemens

```
            ]
        },
        "iconId": "typeCC024"
    }
```

8. Click **Create Type Icon** to register your icon to the type.

Your type icon definition is registered.



Create a new business object of your custom type to see your custom icon instead of the one it would have inherited.

Registering icons using conditions

Type icons configure the icon of a **ViewModelObject**.

Indicators are used to render an icon on a **ViewModelObject** based on some condition, like whether it is checked out or checked in, has a certain release status attached, and so on.

There can be multiple indicators assigned to an object, but it can only be represented by a single type icon. They have a few similarities, though.

- Given a list of types and given a condition that matches, then pick the specified indicator or type icon.
- Given a mechanism to traverse from one reference object to another, and given a condition has been met, pick the given indicator or type icon.

Declarative type icon assignment

For real examples of type icon definitions, examine the existing files shown in the **Configuration Builder**, or use your JSON editor, like Visual Studio Code for example, to search for **"indicators"** in the *STAGE\src\repo* directory. Following in an outline of what an indicator definition looks like. Within the **indicators** object, there may be an array of objects defined.

```
{
    "$schema": "...",
    "typeIconsRegistry": {
    "...": {
      "type": {
        "names": [
          "..."
        ],
        "prop": {
          "names": [
```


© 2025 Siemens

```
              "..."
            ],
            "type": {
              "names": [
                "..."
              ],
              "prop": {
                "names": [
                  "..."
                ],
                "condition": "..."
              }
            }
          }
        },
        "iconId": "..."
      }
      }
    }
```

Declarative Indicator icon assignment

For real examples of indicator definitions, examine the existing files shown in the **Configuration Builder**, or use your JSON editor, like Visual Studio Code for example, to search for **"indicators"** in the *STAGE\src\repo* directory. Following in an outline of what an indicator definition looks like. Within the **indicators** object, there may be an array of objects defined.

```
{
    "$schema": "...",
    "indicators": {
      "...": {
        "tooltip": {
          "showPropDisplayName": true,
          "propNames": [
            "...",
            "..."
          ]
        },
        "type": {
          "names": [
            "...",
            "...",
            "..."
          ],
          "prop": {
            "names": [
              "..."
            ],
            "type": {
              "names": [
                "..."
              ],
              "prop": {
                "names": [
                  "..."
                ],
                "condition": "..."
              }
```

```
            }
          }
        },
        "iconId": "..."
      }
    }
  }
```

Functionality

Following are rules for creating the registration JSON objects.

- **"type"**

  Is the starting point for indicator and type icon evaluation. Intermediate level "**type**" nodes are considered to be evaluated using a new reference model object.

  Must have an array of type names.

- **"prop"**

  Is an optional property of "**type**".

  Must have a name property along with either condition or another type of property

```
    "prop": {
            "name": ["aaaa"],
            "condition": "ctx.test"
        }
    OR
        "prop": {
            "name": ["aaaa"],
            "type": {
                "names": [
                    "ref_obj_type"
                ]
            }
        }
    }
```

  If the prop has a condition, then it must be the leaf level node.

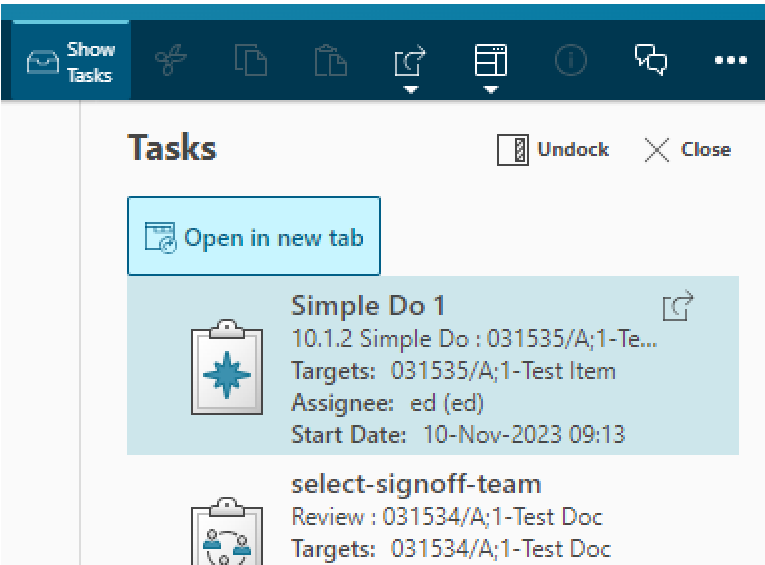
Active Architect workshop

Create a command to quickly display your tasks

In this workshop, you create a command on the work area toolbar which opens a panel, displaying your current workflow tasks.

© 2025 Siemens

This functionality is also found in the **Inbox** location, but creating this new command allows you to quickly see your task list in an information panel without leaving your current page. You will add the ability to open any of the tasks on the list. Finally, you will control where and when this command is displayed.



Create a new command with a panel

Use the **Command Builder** and **Action Bulider** to create a new command.

Prerequisites

You must:

- Be logged on as a user with administrative privileges.
- Switch to the **Active Architect** workspace.

Procedure

1. Open the **Command Builder** .
2. Select **New > Create Command** .
3. Specify **custCmdShowTasks** in the **Command ID** field and then select **Create**.

   Select **Refresh** if it appears.
4. Enter the following property values for your new command:

© 2025 Siemens

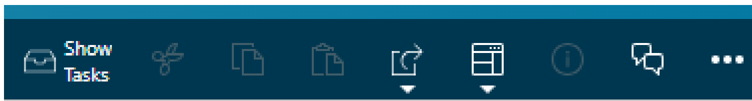
Property Name	Value
Title	Show Tasks
Description	Display the list of your current tasks
Icon Preview	**cmdOpenInbox**

5. If it is not already enabled, select **Development Mode** to see your new command.

   If you see **devMode** in the URL, then **Development Mode** is enabled.

Results

You will see your new command on the **aw_workarea_toolbar** toolbar. This is the default toolbar for new commands.



When you select the **Show Tasks** command, a blank command panel is displayed.

Add a list to the panel

Add a list of your current tasks to the command panel.

Prerequisites

You must:

- Have completed *Create a new command with a panel*.
- Have the **custCmdShowTasks** command open in **Command Builder**.
- Be in developer mode.
- Have at least one task in your inbox.

Procedure

1. Select the **Open in Panel Builder** button to see the empty panel.

   Your command will not be visible while you are in the **Panel Builder** because that toolbar is not displayed.

2. Switch to **Canvas** for a WYSIWYG layout.

© 2025 Siemens

3. With the panel selected, change the **Caption** to **Tasks** and then select **Update**.

4. Drag a **List** component onto the canvas and drop it inside the inner-most rectangle.

5. With the new list selected, choose **Allows the user to select a single item** and then select **Update**.

6. Select **Save Changes**.

   Do not change the view and view model name when prompted. Just select **Save**.

7. Switch to the **Editor** tab and make the following changes:

   a. Replace the **"loadData": {}** action with the following.

```
"loadData": {
    "actionType": "TcSoaService",
    "serviceName": "Internal-AWS2-2023-06-Finder",
    "method": "performSearchViewModel5",
    "inputData": {
        "searchInput": {
            "maxToLoad": 50,
            "maxToReturn": 50,
            "providerName": "Awp0InboxProvider",
            "searchCriteria": {
                "searchInboxContentType": "myTasks",
                "searchFromLocation": "global",
                "searchString": ""
            },
            "searchFilterFieldSortType": "Priority",
            "startIndex": 0
        }
    },
    "policy": {
        "types": [
            {
                "name": "BusinessObject",
                "properties": [
                    {
                        "name": "awp0CellProperties"
                    }
                ]
            }
        ]
    },
    "outputData": {
        "totalFound": "totalFound",
        "filterResults": "{{json:searchResultsJSON}}"
    }
}
```

   b. Add the following **initDataProvider** action:

```
"initDataProvider": {
        "actionType": "dataProvider",
```


© 2025 Siemens

```
        "method": "wysListDataProvider1"
    },
```

c. Add a **onMount** lifecycle hook for the data provider.

```
"lifecycleHooks": {
  "onMount": "initDataProvider"
}
```

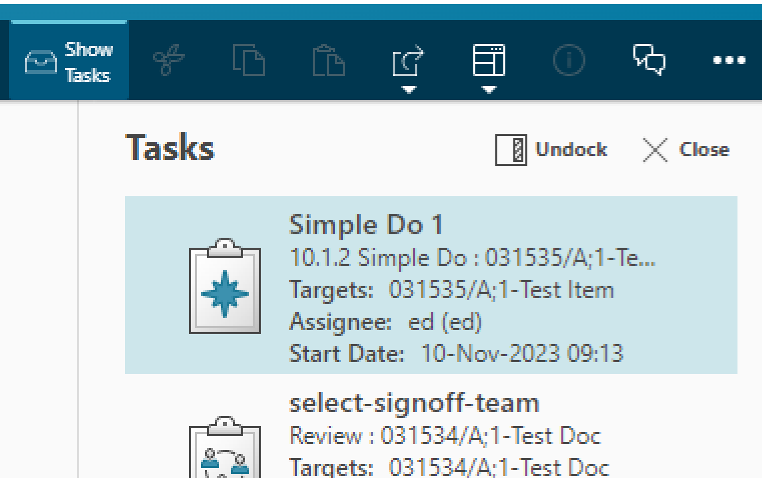
8. Select **Save Changes** .

   Do not change the view and view model name when prompted. Just select **Save**.

9. Switch back to the **Command Builder** and test your updated command. The panel should now display your list of tasks.

Results

You can see a list of your tasks in the command panel and select one of them.



Add controls to the list

Add the cell list anchor commands, which includes the **Open** command, to your task list.

Prerequisites

You must:

- Have completed *Add a list to the panel*.
- Have the **custCmdShowTasks** command open in **Command Builder**.

© 2025 Siemens

- Have the **custCmdShowTasks** command open in the **Panel Builder**.
- Be in developer mode.
- Have at least one task in your inbox.

Procedure

1. From the **Panel Builder**, switch to the **Editor** tab.
2. In the **wysListDataProvider1** object, add the following property.

```
"commandsAnchor":
"com.siemens.splm.clientfx.tcui.commands.modelObjectCellListActionCom
mands"
```

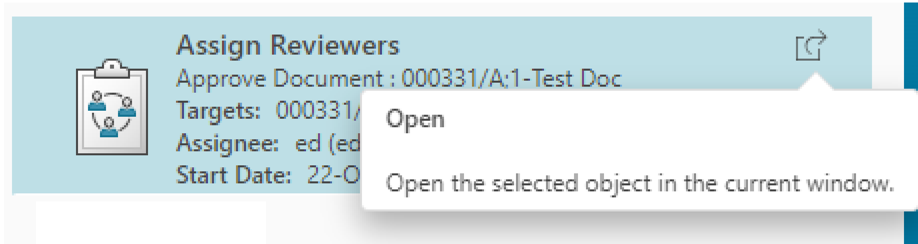
3. Select **Save Changes** .

   Do not change the view and view model name when prompted. Just select **Save**.

4. Deselect the action and then select the **Save Action Flow** command.

Results

This adds the cell list anchor commands to your task list entries. Commands that attach to that anchor are displayed on your tasks. In this example, the **Open** command is displayed.



Add a command bar and command

Add an **Open in New Tab** command to the panel you have created previously.

Prerequisites

You must:

- Have completed *Add controls to the list*.
- Have the **custCmdShowTasks** command open in **Command Builder**.

© 2025 Siemens

- Have the **custCmdShowTasks** command open in the **Panel Builder**.
- Be in developer mode.
- Have at least one task in your inbox.

Procedure

1. From the **Panel Builder**, switch to the **Canvas** tab.
2. Drag a **Command Bar** component onto the canvas and drop it above the list component.
3. In the **Anchor** field, add **custShowTasksAnchor** and select **Update**.
4. Switch to the **Editor** tab.
5. In the view area, update the **context** attribute in the **aw-command-bar** element.

   ```
   context="{selection: data.selectionData}"
   ```

6. In the view model area, add the following to the empty **data** object.

   ```
   "selectionData": {
   "initialValues": {},
   "meta": {}
   }
   ```

7. Add the following to the **wysListDataProvider1** data provider section.

   ```
   "inputData": {
     "selectionData": "{{fields.selectionData}}"
   }
   ```

8. Select **Save Changes**.

   Do not change the view and view model name when prompted.

   > Note:
   >
   > There will be no visible difference in your panel. You have created a command bar, but there are no commands anchored to it.

9. Return to the **Command Builder** to create your new command to open a task in a new tab.
10. Select **Create Command**.

© 2025 Siemens

> Note:
>
> Remember, this command will not be displayed if you have a command selected.

11. Specify **custCmdOpenTask** in the **Command Id** field and choose **Navigate to a URL** in the **Initial Action** field, and then select **Create**.

    Select **Refresh** if it appears.

12. Enter the following property values for your new command:

Property Name	Value
Title	Open in new tab
Description	Open this task in a new tab
Icon Preview	**cmdOpenInNewTab**

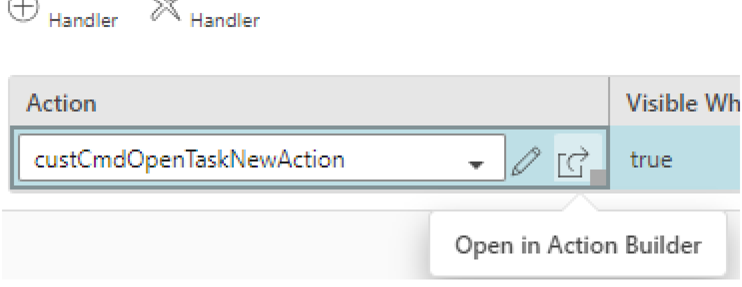
13. In the **Placements** section, change the **Toolbar** to **custShowTasksAnchor** and set the **Style** to **Primary**.

14. In the **Handlers** section, change the **Enable When** to **custTaskSelected**.

15. Edit the new **custTaskSelected** condition, add the following code, and select **Save**.

```
commandContext.selection.selected[0]
```

16. Open the **custCmdOpenTaskNewAction** action in the **Action Builder**.



17. Select the **custCmdOpenTaskNewAction** navigation action and edit the following parameters:

© 2025 Siemens

Property Name	Value
Navigate Params	{ "uid": "{{commandContext.selection.selected[0].uid}}" }
Navigate In	newTab
Navigate To	#/com.siemens.splm.clientfx.tcui.xrt.showObject

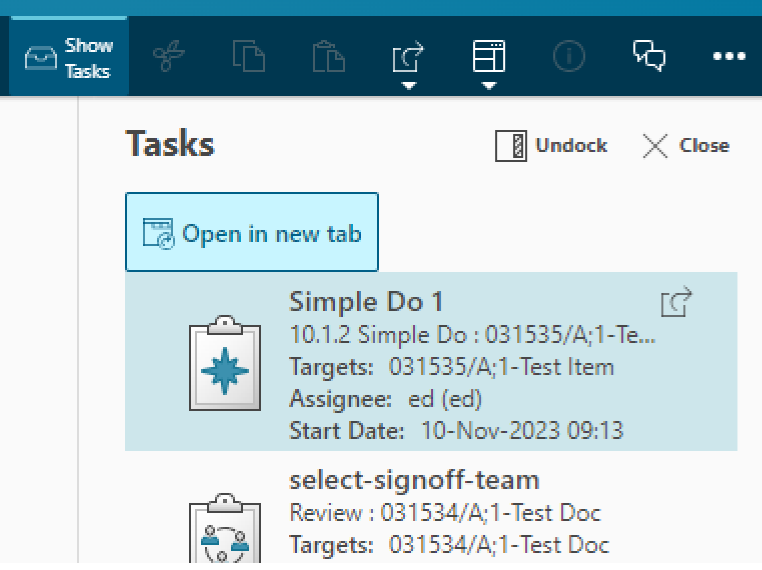
18. Select the **Update** button.

19. Deselect the action and then select the **Save Action Flow** command.

20. Switch back to the **Command Builder** and refresh the page.

Results

Your new command and panel are fully functional. Try it!



Control command visibility and location

Move the location of your command and limit its visibility to the **Author** workspace.

Prerequisites

You must:

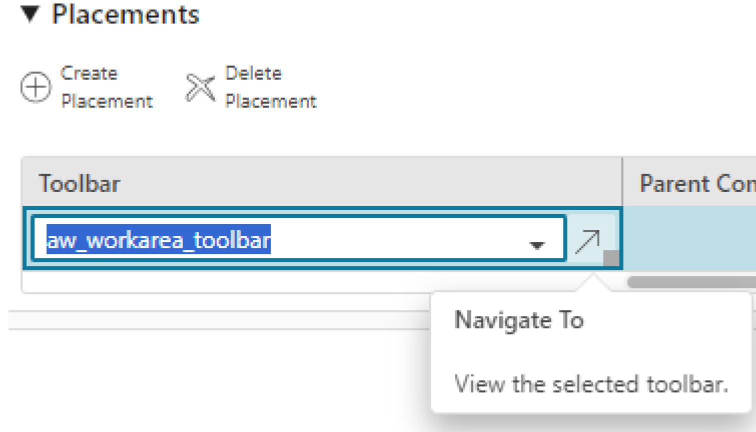
- Have completed *Add a command bar and command*.
- Have the **custCmdShowTasks** command open in **Command Builder**.

© 2025 Siemens

- Be in developer mode.

Procedure

1. In the **Placement** section, **Navigate To** the **aw_workarea_toolbar** to review the list of commands anchored to this toolbar and their priority.



2. Click the browser's back button to return to the **custCmdShowTasks** command.

3. In the **Placement** section, change the **Priority** value from **6001** to **150**.

4. In the **Handlers** section, change the **Visible When** value from **true** to **isAuthorWorkspace**.

5. **Edit** the **isAuthorWorkspace** condition with the following, and then select **Save**.

```
ctx.workspace.workspaceId === 'TcAuthorWorkspace'
```

   Notice the command disappears from the toolbar. The command will no longer be displayed when you are in the Active Architect workspace, only in the Author workspace.

6. Click the **Refresh** button.

7. Switch to several different workspaces to see when and where the command appears.

Results

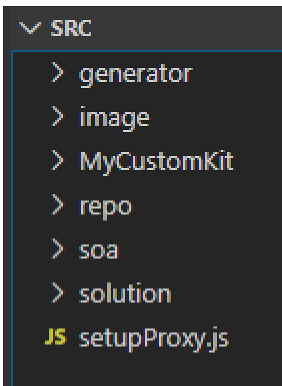
Your new command is next to the **More Commands** menu and only appears when you are in the **Author** workspace.

© 2025 Siemens

The Active Workspace development environment

The stage directory

Following is a description of several of the important directories for Active Workspace customization. Not all directories present in the environment are shown.



stage

This is the base directory for Active Workspace customization. It is located in **TC_ROOT***aws2*.

stage/out

This is where the local build can be found.

site	This directory is a live web application site which can be **served by the development server**.

stage/src

The files and directories located here are for your customizations. They are pre-populated with actual source code and resources that are built along with the contents of the *repo* directory. The **src** directory contains the following directories:

image	Contains the icons used by the Active Workspace UI are located here. If you want to add custom icons for your modules, place them in this directory. Follow the file and naming conventions for icons as specified in the *UI Pattern Library*, located on Support Center.
{custom kit}	You custom kit directories are automatically created in the **stage/src** directory. You may have as few or as many kits as you need to organize your content. Each of your custom module directories are located in one of your kit directories.
solution	In this directory, you will find the Active Workspace **kit.json** file and all of the workspace definitions.

© 2025 Siemens

> Caution:
>
> Siemens Digital Industries Software recommends not modifying any of the OOTB files in the **solution** directory.

stage/src/repo

This directory contains the declarative source code for Active Workspace, and is required when building the application.

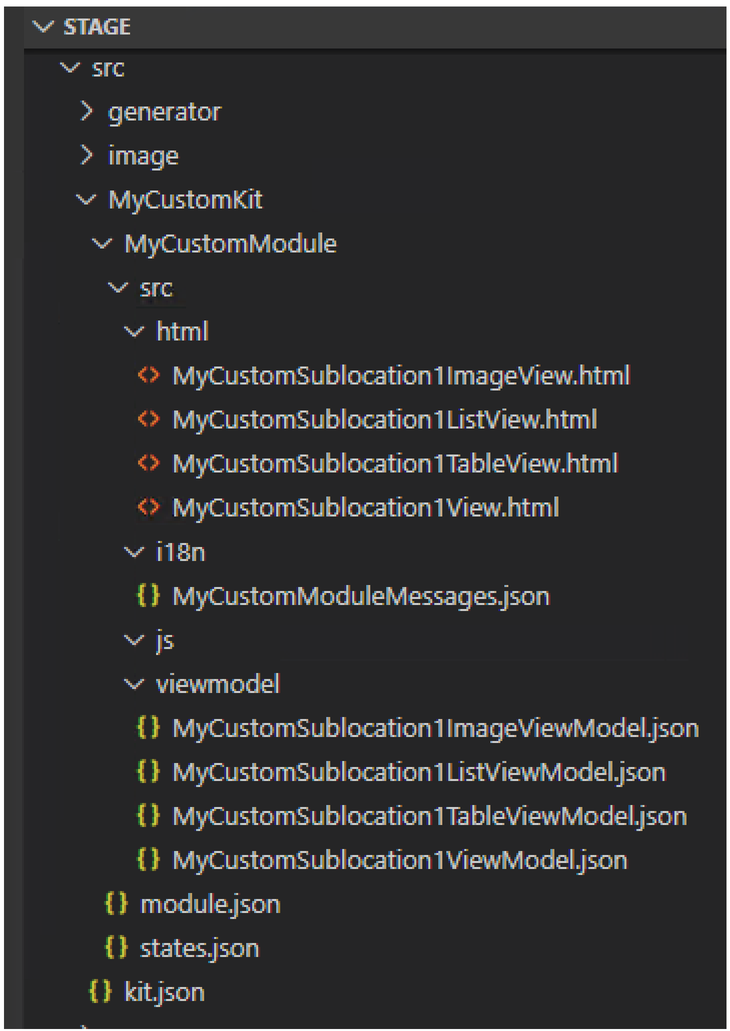
You can use the declarative source definitions in this directory for reference; all of the installed OOTB kits are located here in their respective directories.

> Caution:
>
> Siemens Digital Industries Software recommends not modifying any of the files in the **repo** directory. Instead, create your own modules and add your customizations there.

The declarative customization project

Your custom module must follow the declarative file and directory structure. Following is an example of a module with several customizations (yours will differ):

© 2025 Siemens



> Tip:
>
> Use the **aw add** script to create boilerplate files for you.

stage/src/MyCustomKit

You custom kit directory. You can create multiple kits to help organize your customizations.

stage/src/{kit}/MyCusomModule

This is an example of a base directory for your custom module. It is located in the **STAGE***src* folder. Your **module.json** file (required) is located here. Depending on your customizations, you may also have the following files:

- **aliasRegistry.json**

  Assign type icons to business objects in this file.

© 2025 Siemens

- **typeIconsRegistry.json**

  Assign business objects conditionally to icons in this file.

- **commandsViewModel.json**

  Define new commands here, including handlers and placements.

- **states.json**

  Define your custom locations or sublocations here.

- **indicators.json**

  Assign visual indicators in this file.

- **typeProperties.json**

  Specify any *additional* properties that you want the client to pre-load. This can be useful when defining visual indicators, for example.

src

This directory contains the main organizational directories for your module: *html*, *i18n*, *js*, and *viewmodel*. If you created a theme, you will see your SCSS file here.

html

Your view definitions are located in this directory. The definitions have the following naming convention:

sublocationNameView.html

sublocationNameListView.html

sublocationNameTableView.html

i18n

All your localization keys for your module are located in this directory. The English translation file is named **moduleNameMessages.json**. Additional language files will have a suffix distinguishing them.

> Example:
>
> **moduleNameMessages_de.json** for German
>
> **moduleNameMessages_ja_JP.json** for Japanese

For more examples, refer to the provided source files in the *components\activeworkspace\repo\kit* directory.

js

If you created a command, the **JSFunction** method will be located in this directory. The default name of the file is **commandNameService.js**

viewmodel

This directory contains your view model files. Each view (html) file must have an associated view model (json) file.

Development scripts

If you choose to use the command line to create custom Active Workspace components, you must work within an environment configured for Active Workspace development. Siemens Digital Industries Software provides a script to assist in the development of component modules and provide several other services. Unless otherwise stated, all scripts must be run from the **stage** directory.

initEnv.cmd

In a Windows environment, run this script to configure the command-line environment variables necessary for Active Workspace development.

aw

This provides every capability you need to create, populate, build, and publish your custom declarative module. Use the **help** argument to get more information.

```
aw help
```

Argument	Description
add	Add declarative artifacts to your module.
audit	Audit the current project.
build	Build the current project.
clean	Clean up temporary files generated during the build process.
doc	Search for a specific string in the documentation.
help	Display the help for the aw script.
info	Display information about the specified artifact.
migrate	Perform a code migration or automated refactoring on the whole project.
report	Generate reports.

© 2025 Siemens

Argument	Description
search	Search for declarative artifacts containing the specified string.
set	Set or clear a specific setting, such as the default module or kit to work on.
start	Start a site.
test	Run unit tests for the current solution.
upgrade	Upgrade the current project to a newer version of Siemens Web Framework.

aw add

This argument assists you in creating boilerplate files and directories for individual components.

```
aw help add
```

Use...	To add a new...
aw add authenticator	Authenticator
aw add clipboard	Clipboard
aw add command	Command
aw add component	Component (view and view model)
aw add i18n	i18n entry to the i18n section of a view model
aw add kit	Kit
aw add location	Location route (state)
aw add module	Module
aw add property	ViewModelProperty to the data section of a view model
aw add route	Route (state)
aw add sublocation	Sublocation route (state)
aw add theme	Theme
aw add type	Type icon for a type (aliasRegistry)
aw add workspace	Workspace

aw audit

Check your declarative definitions against the *stated* schema version. It reports any errors it encounters, which you must fix before continuing. The current schema file is available in the **UI Pattern Library** on Support Center.

For example, if a view model file contains `"schemaVersion" : "1.0.0"`, then it will be audited versus schema version.1.0.0, even if 1.0.1 were current.

© 2025 Siemens

aw start

Launch a **temporary application server for declarative development**.

aw build

Performs a clean build and minification of the Active Workspace application from **STAGE**/*src* and **STAGE**/*repo* into the **STAGE**/*out/site* directory.

> Caution:
>
> This also clears out the declarative artifact repository, so any UI builder changes you have made since your last **exportToSrc** will be lost.

aw publish

Publishes the Active Workspace application to the gateway deployment specified in **STAGE***tem.properties*.

The target gateway can be overridden by running `npm run publish` **<gateway deployment URL>**.

aw genSoaApi

Use this to generate a local site documenting all of the SOA APIs available in Active Workspace. The output directory is **STAGE**/*out/soa/api*.

aw exportToSrc

When you are done making your changes in the UI using **Active Architect** and you want to commit them, you need to export them into your local source repository.

All of the changes that were made with **Command Builder**, **Panel Builder**, and so on, are stored in the declarative artifact service, but any time Active Workspace (the artifact service specifically) is rebuilt, that history and all changes are cleared. Think of exporting back to source as similar to checking in code to source control.

```
aw exportToSrc <Gateway Client URL> -- --moduleName=<name of module>
```

- The *Gateway Client URL* defaults to **http://localhost:3000**, if not provided.
- The *name of module* is the name of the existing module in which the changes should be stored. It will not create one. You can create a custom module using the **generateModule** script.

After running the tool, you can rebuild Active Workspace as much as you want without losing those changes. Any future changes will be stored in the artifact service until you run this script again. This process is similar to checking your code into a source control manager.

© 2025 Siemens

aw convertTemplates

Converts your deployed custom SOA templates from the Business Modeler IDE for use in Active Workspace. You must provide the location of the deployed templates which is normally the **TC_DATA_MODEL** directory. The converted templates are added to the *out/soa/json* folder and will override any files there. These converted files are used for the generation of the schema used for service input validation and defaults. Use **genSoaApi** to create a new set of documentation.

Once your testing is complete, move the files from the *out/soa/json* directory to the *src/soa/json* directory to avoid upgrade issues.

Example:

To run this command from the Windows operating system:

```
aw convertTemplates D:\teamcenter\tcdata\model
```

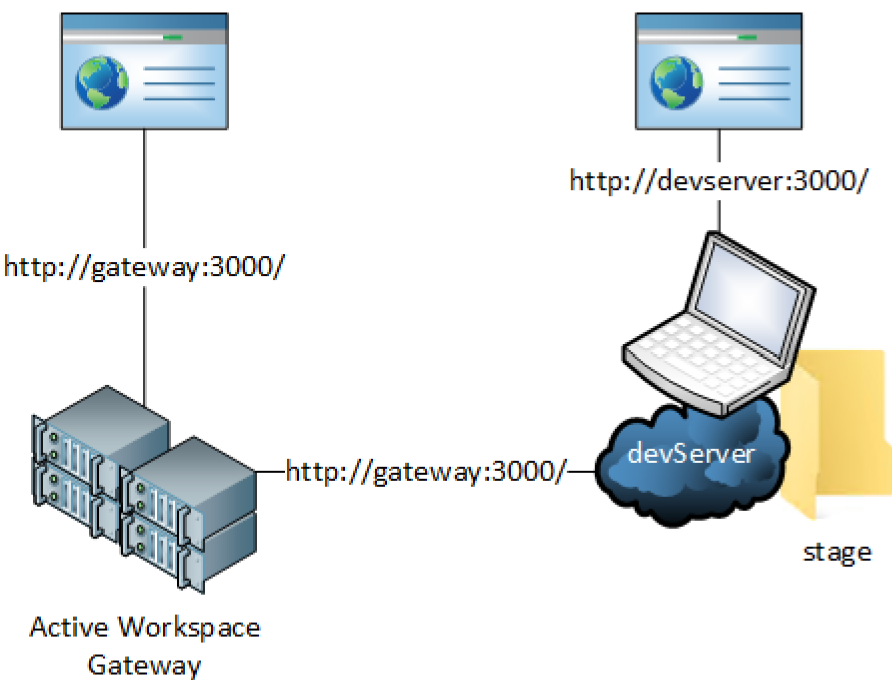
Verify your customizations using the development server

The Active Workspace development server

Use the development server to temporarily host the Active Workspace application locally in order to quickly check your declarative module changes without affecting the main server or having to redeploy the application.

When you run the *development server*, a web application server runs *locally* using the build from your **STAGE** directory and connects to an existing Active Workspace site. This allows you to connect to the production site with or without your customizations.

© 2025 Siemens



Staring the development server

Open an Active Workspace development command prompt from the **stage** directory.

Run the development server using **aw start**. This must be done using an Active Workspace development environment and from the **stage** directory. Use `aw help start` for more information on available arguments.

> Example:
>
> In this example, you set the dev server port to 3001, set the gateway to **10.16.68.16** on port **3000**.
>
> **aw start --port=3001 --gateway=http://10.16.68.16:3000**
>
> After the script lists all kits and modules, the following message is displayed and a web browser opens pointing to the server.
>
> ```
> info: + Starting the development server...
> ```

If necessary, the source code is compiled (which may take several minutes). Once the compile completes, the following message is displayed and your development server is ready to use.

```
info: + webpack compiled
```


© 2025 Siemens

Stopping the development server

The development server runs in the command-line window until you stop it with **CTRL-C**. While it is running, it will detect any changes made to your custom native modules and update itself accordingly; there is no need to run refresh after each change. This allows you to validate your changes before running a full build.

> Note:
>
> Stop your development server before performing a full build.

Active Workspace development examples

Active Workspace development environment

What do I need to know about the development environment?

- **Learn how the development environment works with the gateway architecture.**

  The gateway and microservices architecture are an important component in the development environment. You must configure the gateway to read from your local site instead of the file repository so you can see your command-line development changes.

- **Learn the environment directories**

  NPM is the package manager for Active Workspace. The **STAGE** directory can be loaded by a JavaScript project IDE, such as Microsoft's Visual Studio Code.

- The **provided scripts** help you define, build, and publish your customizations.

- Your **custom module's file structure** must be followed, and is created and maintained by the scripts.

What are these examples for?

These examples demonstrate how to use the **generateModule** script to create various declarative Active Workspace customizations from the command line. For example, adding a new sublocation or creating a new theme.

> Tip:
>
> Siemens Digital Industries Software recommends using the **command builder** page to create and modify commands.

© 2025 Siemens

What do I need to make these examples work?

These examples assume you:

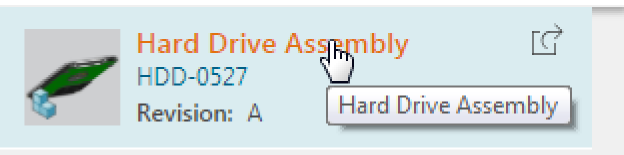
- Have write access to the **TC_ROOT***aws2\stage* directory.
- Use a **JavaScript**-capable project editor, like **Visual Studio Code** for example.

> Tip:
>
> If you are using a Linux environment, you will need to install the version of **Node.js** specified in the Hardware and Software Certifications knowledge base article on Support Center.
>
> The Active Workspace development environment on Windows includes **Node.js**.

Example: Make cell titles clickable

This example only requires that you rebuild and deploy. It does not require you to create a module.

In this example, you make the cell titles (normally the object name) clickable by adding an optional section to the *kit.json* file.



1. Locate and open Active Workspace's *kit.json* file.

   This file is located in the **STAGE***src\solution* directory.

2. Add the following section to the **solutionDef** portion of the file.

   ```
   "clickableCellTitleActions": {
      "click": "Awp0ShowObjectCellTitle",
      "ctrlClick": "Awp0OpenInNewTabCellTitle",
      "shiftClick": "Awp0OpenInNewWindowCellTitle",
      "doubleClick": "Awp0ShowObjectCellTitle"
      }
   ```

   The **solutionDef** section is an array, so you will have to add a comma either before or after this section, depending on where you insert it.

   ```
   {
     "name": "tc-aw-solution",
     ...
   ```


© 2025 Siemens

```
        "solutionDef": {
            "solutionId": "TcAW",
            "solutionName": "Active Workspace",
            ...
            "commandVisibility": "js/tcCommandVisibilityService",
            "clickableCellTitleActions": {
                "click": "Awp0ShowObjectCellTitle",
                "ctrlClick": "Awp0OpenInNewTabCellTitle",
                "shiftClick": "Awp0OpenInNewWindowCellTitle",
                "doubleClick": "Awp0ShowObjectCellTitle"
             },
            "bundler": { ...
            }
        },
        ...
        "defaultDragAndDropHandlers": { ...
        }
    }
```

3. Build the application, deploy, and test.

Siemens Web Framework

This functionality is part of the Siemens Web Framework upon which Active Workspace is built. Refer to *Solution configuration* for more information on **clickableCellTitleActions** within the Digital Engineering Services product on Support Center.

Create boilerplate declarative artifacts

Active Workspace uses kits and modules to contain your custom declarative artifact definitions. Each of your kits consist of a **stage\srckitName** directory. This kit directory contains a **kit.json** file which is dependent upon the core Active Workspace **kit.json**.

Each module you create belongs to one of your kits. The module directory consists of a **module.json** file, a **states.json** file and a **src** directory.

The Active Workspace solution kit file is located in the *stage\src\solution* directory.

> Caution:
>
> Siemens Digital Industries Software recommends that you do not modify this file.

Command line and Active Architect

The Active Workspace command-line development environment works together with the UI-based tools of **Active Architect**. While you can quickly iterate using the UI-based tools, ignoring the command

© 2025 Siemens

line environment, you must eventually use the command line to build and publish your changes to a production server.

The changes made using UI tools are stored in a declarative repository independent of your kits and modules. Similarly, changes made in your kits and modules will not appear in the UI tools.

When you want to finalize your UI-based changes, use **exportToSrc** to move the declarative definitions from the declarative repository into a kit and module you choose.

Example: Create a declarative project space

Create a custom declarative kit and module using the Active Workspace development environment.

Prerequisites

You must have access to the Active Workspace **STAGE** directory.

Procedure

This procedure assumes that you have not defined any customizations yet.

1. Open an normal operating system command prompt (not a Teamcenter command prompt) and navigate to the **STAGE** directory.

   This is the **aw2/stage** in your *TC_ROOT* directory.

2. If you are using Windows, run the **initenv.cmd** script.

   This is not necessary if you use Linux.

   You now have an environment configured for Active Workspace development.

3. Create a custom kit to hold your Active Workspace customizations.

   ```
   aw add kit
   ```

   Specify the name of your kit: **MyCustomKit**

4. Add a new module to your kit.

   ```
   aw add module
   ```

   Specify the name of your module: **MyCustomModule**

   Your new module is added to the current kit.

© 2025 Siemens

Results

You now have an Active Workspace development environment project.

The following are created in the *stage/src* directory.

MyCustomKit

This directory is created in the **stage/src** directory to contain all the files related to your custom kit.

This is your kit directory.

kit.json

The kit file is automatically created in your kit directory. It contains the name of your custom kit and its dependency on the core Active Workspace kit, along with a reference to your custom module.

```
{
    "name": "MyCustomKit",
    "level": 7,
    "kitDeps": [
        "tc-aw-solution"
    ],
    "modules": [
        "MyCustomModule"
    ]
}
```

MyCustomModule

This directory is created in your kit directory to contain all the files related to your custom module.

This is your module directory.

module.json

This file is created in your module directory and populated with the basic information about your custom module.

```
{
    "name": "MyCustomModule",
    "description": "MyCustomModule module"
}
```

states.json

This file is created in your module directory and will hold the definitions for any new states (pages, tabs, locations, and sub locations).

This file contains no information yet.

src

© 2025 Siemens

This directory is created in your module directory to contain all the files related to your custom module. It contains several directories for organizing your view, view model, localization, and other files.

This is your module's source directory.

Postrequisites

You can now add new client-side workspace definitions, views, view models, view model properties, and so on. You can also create new modules in this kit to organize your custom declarative artifacts, or create new kits for an additional organization level.

Use **aw set** to change which kit or module is the default when you create new declarative artifacts.

© 2025 Siemens

4. Working with platform customizations

Integrating Teamcenter platform customizations

What are platform customizations?

The following are considerations for the Active Workspace client when you perform platform customizations:

- **Enabling your custom business object.**
- **Enabling your custom preferences.**
- **Integrating your custom workflow handlers.**

What do I need to make these examples work?

These examples assume you have already installed Active Workspace. In addition, you will need:

- Access to the *TC_ROOT\aws2\stage* directory.

Enable a custom business object in Active Workspace

CREATE
- Create BMIDE project
- Create custom Item type
- Add custom properties
- Package your project

DEPLOY
- Copy package to Deployment Center
- Add package to environment
- Generate the deployment scripts
- Deploy and run the scripts

FINISH
- Add custom type icon
- Display custom properties
- Create access rules

1. Perform the following steps to create the business object in the Business Modeler IDE:

    a. Import the Active Workspace (**aws2**) template into your Business Modeler IDE project.

© 2025 Siemens

b. Create the new business object and create custom properties on the business object.

c. Use the **Operation Descriptor** tab to ensure the custom properties appear on the creation page in Active Workspace.

d. Ensure that the custom business object and its properties are included in searches by applying the **Awp0SearchIsIndexed** business object constant and property constant.

   If you have installed Active Content Structure and you want to make the custom business object available on the **Content** tab, add the custom business object to the **Awb0SupportsStructure** global constant.

e. Install the custom template to the server.

For details about using the Business Modeler IDE, see *BMIDE for Data Model Design* in the Teamcenter help.

2. Update the search indexer by merging the Teamcenter schema with the Solr schema and re indexing.

3. Most custom object types will automatically appear in the **Create** panel. However, if you create an unusual type of object and it does not appear, add your object type to the **AWC_DefaultCreateTypes** preference.

4. If you want a unique page layout for the business object, set up **style sheets** for the business object's create page, summary page, and information panel.

5. Optionally, to add special permissions restrictions to the business object, use the access manager to create access control lists or projects and programs.

Adding custom type icons

You can use three main ways to assign icons to business object types. These are static methods, meaning the icon assignment does not change.

> Note:
>
> If you want to register icons using conditions which includes type icons and indicators, then you must **use configuration builder to register type icons**.

Adding an icon to a type which does not have one

You can add a new icon to a type that does not already have its own. This can be either your custom item type or an OOTB type. Types that do not have their own icon inherit icon of their parent.

© 2025 Siemens

Active Workspace uses a naming convention for its icons. As long as you follow this convention, it is easy to add icons for your custom business object types. The icons that get compiled into the web application are located in the **STAGE***src**image* directory. Add your new icon into this directory, and then build the Active Workspace application.

The naming pattern for type icons is:

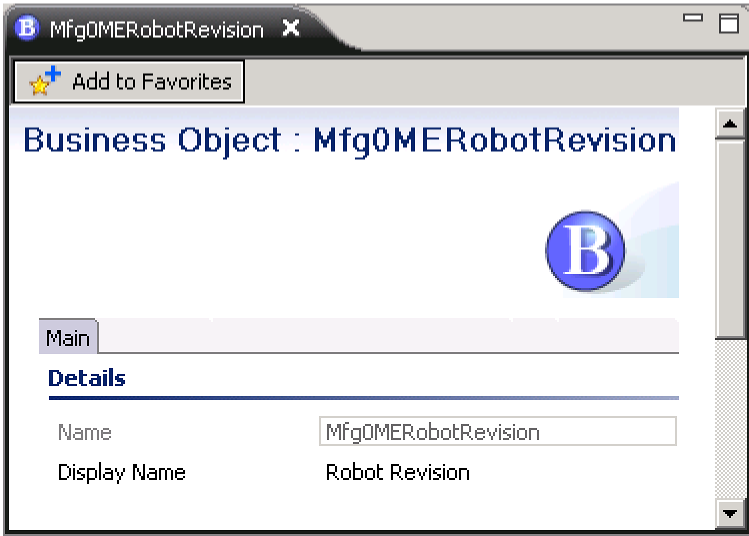
```
typetypename48.svg
```

To use this method, *typename* must be the database name of the object type.

Example:

For example, if you want to add a new icon for the **Robot Revision**, you need to find the database name of that type:

The database names of the objects can be located using the Business Modeler IDE.



You find that **Mfg0MERobotRevision** is the database name of the **Robot Revision** business object. So you would name your icon:

```
typeMfg0MERobotRevision48.svg
```

Custom business objects follow the same rule. For example, if you have created a new business object called **C9BoltRevision** with a display name of **Bolt Revision**, you would add a new icon with the following name:

```
typeC9BoltRevision48.svg
```


© 2025 Siemens

> Note:
>
> If a business object type does not have an icon specifically assigned to it either by an appropriately-named icon in the source directory or by an advanced method, then Active Workspace will display the default icon for that class of objects.

After making your change and replacing the icons in the *stage\src\image* directory, rebuild and redeploy Active Workspace.

Replacing an OOTB icon but keeping the same name

If you want to *replace* an OOTB icon, you *must* modify the *build.json* file located in your **stage directory**.

Add a JSON object listing the names of the icons you have replaced. Adding icon names here will force the build to re-evaluate those icons, otherwise the system will assume they have already been processed.

> Example:
>
> The following forces the **typeItem48.svg** and **typeItemRevision48.svg** icons to be re-evaluated during the build.
>
> ```
> "customImages": [
>    "typeItem48",
>    "typeItemRevision48"
> ]
> ```

After making your change and replacing the icons in the *stage\src\image* directory, rebuild and redeploy Active Workspace.

Assign icons to types with mismatched names

This method requires that you to use the **development environment** and create a custom module.

The **aliasRegistry** contains the icon and type pairing when the names do not match.

The *aliasRegistry.json* file is a sibling to your custom *module.json*. Use the **generateModule script** to create this file automatically.

This file must contain a JSON object listing icons and the business objects to which they are assigned.

> Example:
>
> The following registers the **typemyTypeIcon48.svg** to the **myCustomTypeRev** business object.

© 2025 Siemens

```
{
    "typemyTypeIcon48": [
        "myCustomTypeRev"
    ]
}
```

The following registers two different icons to a total of three business objects.

```
{
    "typemyTypeIcon48": [
        "myCustomTypeRev",
        "myOtherCustomTypeRev"
    ],
    "typemyNewCustomIcon48": [
        "myNewCustomTypeRev"
    ]
}
```

Registering icons using conditions

Type icons configure the icon of a **ViewModelObject**.

Indicators are used to render an icon on a **ViewModelObject** based on some condition, like whether it is checked out or checked in, has a certain release status attached, and so on.

There can be multiple indicators assigned to an object, but it can only be represented by a single type icon. They have a few similarities, though.

- Given a list of types and given a condition that matches, then pick the specified indicator or type icon.
- Given a mechanism to traverse from one reference object to another, and given a condition has been met, pick the given indicator or type icon.

Declarative type icon assignment

For real examples of type icon definitions, examine the existing files shown in the **Configuration Builder**, or use your JSON editor, like Visual Studio Code for example, to search for **"indicators"** in the *STAGE\src\repo* directory. Following in an outline of what an indicator definition looks like. Within the **indicators** object, there may be an array of objects defined.

```
{
    "$schema": "...",
    "typeIconsRegistry": {
    "...": {
      "type": {
        "names": [
          "..."
        ],
```


© 2025 Siemens

```
          "prop": {
            "names": [
              "..."
            ],
            "type": {
              "names": [
                "..."
              ],
              "prop": {
                "names": [
                  "..."
                ],
                "condition": "..."
              }
            }
          }
        },
        "iconId": "..."
      }
    }
  }
```

Declarative Indicator icon assignment

For real examples of indicator definitions, examine the existing files shown in the **Configuration Builder**, or use your JSON editor, like Visual Studio Code for example, to search for **"indicators"** in the *STAGE\src\repo* directory. Following in an outline of what an indicator definition looks like. Within the **indicators** object, there may be an array of objects defined.

```
{
    "$schema": "...",
    "indicators": {
      "...": {
        "tooltip": {
          "showPropDisplayName": true,
          "propNames": [
            "...",
            "..."
          ]
        },
        "type": {
          "names": [
            "...",
            "...",
            "..."
          ],
          "prop": {
            "names": [
              "..."
            ],
            "type": {
              "names": [
                "..."
              ],
              "prop": {
                "names": [
                  "..."
                ],
```


© 2025 Siemens

```
                "condition": "..."
              }
            }
          }
        },
        "iconId": "..."
      }
    }
  }
```

Functionality

Following are rules for creating the registration JSON objects.

- **"type"**

  Is the starting point for indicator and type icon evaluation. Intermediate level "**type**" nodes are considered to be evaluated using a new reference model object.

  Must have an array of type names.

- **"prop"**

  Is an optional property of "**type**".

  Must have a name property along with either condition or another type of property

```
"prop": {
        "name": ["aaaa"],
        "condition": "ctx.test"
    }
OR
    "prop": {
        "name": ["aaaa"],
        "type": {
            "names": [
                "ref_obj_type"
            ]
        }
    }
}
```

  If the prop has a condition, then it must be the leaf level node.

Enabling your custom preferences in Active Workspace

For any custom Teamcenter preference that you want to use in Active Workspace, you must add it to the **AWC_StartupPreferences** preference. Otherwise, the preference will not be loaded by Active Workspace.

© 2025 Siemens

The **AWC_StartupPreferences** preference defines the list of preferences to be retrieved at startup by the Active Workspace client from the Teamcenter server. Each entry in the list is a valid Teamcenter preference name.

Enable your custom workflow handler in Active Workspace

If you have a custom workflow handler, Siemens Digital Industries Software recommends that you create a corresponding JSON file describing the handler.

Why should I to do this?

Without a corresponding JSON file, the Workflow Designer handler panel offers no assistance to the user for that handler.

The Active Workspace client reads the handler JSON files to create a handler panel populated with available arguments and value hints, instead of making the user type them in manually, reducing errors and increasing efficiency. The handler panel:

- Provides a drop-down list of optional arguments.
- Enforces required arguments so the user cannot forget to add the
- Provides drop-down lists for possible values.
- Tracks mutually exclusive or dependent arguments.

Where do I put the handler JSON file?

Add your custom JSON file to the provided files in the **TC_DATA***workflow_handlers* directory. The filename must match the handler name. Reference the OOTB files as examples when creating your own.

Example:

The handler named **EPM-auto-assign-rest** has a file:

TC_DATA*workflow_handlers\EPM-auto-assign-rest.json*

How is the handler defined?

In the JSON file, you must define all arguments as either **mandatory** or **optional**. For any arguments you defined, you may also specify if they are dependent upon each other, if they are mutually exclusive, required, allowed to be left blank, or never take any value at all. The following syntax allows you to create all of these possibilities.

© 2025 Siemens

Syntax

The JSON file may contain any of the following attributes:

```
{
  "mandatory": [...],
  "optional": [...],
  "dependent": [...],
  "mutex": [...],
  "required_one_of": [...],
  "nullable": [...],
  "nullvalue": [...],
  "undefined_arg_value": [...]
}
```

Or, if the handler takes no arguments, then you can define it as follows:

```
{
  "no_arguments": true
}
```

If a handler has no corresponding JSON file, or if the definitions are blank, then the user can (must) manually type in any arguments and values they want.

Attributes

mandatory

Define any arguments which are mandatory, along with the list of available value hints for each argument. Required arguments will appear pre-populated in the handler panel. If no value hints are provided, the user must type in the values.

Example:

Shown are two mandatory arguments: **-name** which has three value hints to choose from, and **-assignee** (which requires the user to manually enter a value).

```
"mandatory": [
    {
      "-name": [
        "PROPOSED_RESPONSIBLE_PARTY",
        "ANALYST",
        "CHANGE_SPECIALIST"
      ]
    },
    {
      "-assignee": [ ]
    }
  ]
```


© 2025 Siemens

Tip:

dynamic hints and **multiselect** are available when defining argument values.

optional

Define any arguments which are optional, along with the list of available value hints for each argument.

Example:

Shown are two optional arguments: **-from_attach** (which has three values to choose from) and **-target_task** (which requires the user to manually enter a value).

```
"optional": [
  {
    "-from_attach": [
      "target",
      "reference",
      "schedule_task"
    ]
  },
  {
    "-target_task": [ ]
  }
]
```

Tip:

dynamic hints and **multiselect** are available when defining argument values.

dependent

Specify which arguments that are dependent upon other arguments being chosen. This is a one-way relation. If you have arguments that are dependent upon each other, you need to specify all combinations.

Example:

Shown are two sets of dependent arguments. If the user selects **-latest** or **-targetstatus**, then **-status** is required. However, if there are no other restrictions, the user could select **-status** by itself.

```
"dependent": [
  {
      "-latest":["-status"]
  },
  {
      "-targetstatus":["-status"]
```


© 2025 Siemens

```
    }
  ]
```

mutex

Specify which arguments are *mut*ually *ex*clusive. Each group is processed separately.

Example:

Shown are two sets of mutually exclusive arguments. If the user selects **-primary_type** they cannot also select **-secondary_type**, and vice-versa. The same with **-check_only_for_component** and **-check_only_for_assembly**, the user can only choose one of them.

```
  "mutex": [
    {
      "-primary_type": "",
      "-secondary_type": ""
    },
    {
      "-check_only_for_component": "",
      "-check_only_for_assembly": ""
    }
  ]
```

required_one_of

Specify arguments from which at least one *must* be chosen.

Example:

```
"required_one_of": [
  {
    "-allowed_status": "",
    "-include_related_type": "",
    "-relation": ""
  }
]
```

You can specify multiple sets of required arguments. In the following example, the user must choose either **arg1** or **arg2** and also choose either **argA** or **argB**.

```
  "required_one_of": [
      {
          "-arg1": "",
          "-arg2": ""
      },
      {
          "-argA": "",
          "-argB": ""
```


© 2025 Siemens

```
        }
    ]
```

nullable

Specify which arguments *may* have a value. The user is allowed to provide a value, but it is not required.

Example:

```
"nullable": [
  "-ce",
  "-auto_complete",
  "-clear_signoffs",
  "-check_first_object_only",
  "-required"
]
```

nullvalue

Specify which arguments *can not* have a value. The user can not add any value to the arguments.

```
"nullvalue": [
     "-bypass",
     "-check_first_target_only"
]
```

undefined_arg_value

If a JSON file has this specified, then the user would be allowed to add any argument name and a corresponding value for it of his\her choice. The handler panel will have a free text to provide any argument name as well as argument value that may not be specified in the JSON mandatory or optional sections.

Dynamic hints

Within the argument definition sections (**mandatory** and **optional**), you can add *dynamic hints*. The server will replace the following keywords with a list of values before sending to the client. Supported keywords are:

To get a list of...	use...
Relations	DYNAMIC_HINT_RELATIONS
Supported types	DYNAMIC_HINT_TYPES
LOVs	DYNAMIC_HINT_LOV
Release statuses	DYNAMIC_HINT_STATUS
Available workflow process templates	DYNAMIC_HINT_TEMPLATE
All ACLs	DYNAMIC_HINT_ACLS
Workflow ACLs	DYNAMIC_HINT_WORKFLOW_ACLS

© 2025 Siemens

To get a list of...	use...
System ACLs	DYNAMIC_HINT_SYSTEM_ACLS
BMIDE conditions	DYNAMIC_HINT_CONDITIONS
Group names	DYNAMIC_HINT_GROUPS
User names	DYNAMIC_HINT_USERS
Revision rules	DYNAMIC_HINT_REVISION_RULES
Form type names	DYNAMIC_HINT_FORMS
Queries	DYNAMIC_HINT_QUERIES
Note types	DYNAMIC_HINT_NOTETYPES
Participant types	DYNAMIC_HINT_PARTICIPANTS
Single participant types	DYNAMIC_HINT_SINGLE_PARTICIPANTS

Example:

The following provides a list of all relations when the user selects the **-from_relation** argument.

```
{
  "-from_relation": [
    "DYNAMIC_HINT_RELATIONS"
  ]
}
```

Multiselect

Within the argument definition sections (**mandatory** and **optional**), you can allow *multiselect*.

Use **multiselect=true** to allow the user to pick multiple values for the argument. This should be present as the first value in the list of values for a given handler argument.

Example:

The user is allowed to choose several values from the list of types.

```
{
  "-to_include_type": [
    {
      "multiselect": true
    },
    "DYNAMIC_HINT_TYPES"
  ]
}
```

Examples

Explore all the provided JSON files for examples. Following are some examples chosen from the provided JSON files. They each have interesting content to examine for sample content.

© 2025 Siemens

EPM-assert-targets-checked-in

As there are no arguments for this handler, this JSON file is empty. A blank template.

EPM-check-target-object

An example of using a combination of multiselect, explicit values, and dynamic hints.

EPM-attach-related-objects

An example of several mutually exclusive groups of arguments.

© 2025 Siemens

5. Active Workspace Customization Reference

Troubleshooting

Open source software attributions

Open source software (OSS) attribution for Active Workspace can be found in the **OSSAttributionInfo.json** file. You can find this file in the **config** folder within the web application's assets directory.

In the development environment, you can find this file in the *TC_ROOT***aws2\stage\out\war\assets***xxxx***\config** folder.

Examine the application context

The Active Workspace maintains an application context object, **ctx**, used to hold state. Application data like the current displayed page, selected objects, user's id, group, role, and so on. Even a list of Teamcenter preferences from the user's perspective. The data in this object can be used by conditions for the declarative UI.

Following is a brief look at the Redux DevTools. For more information, search for Redux in the Siemens Web Framework documentation.

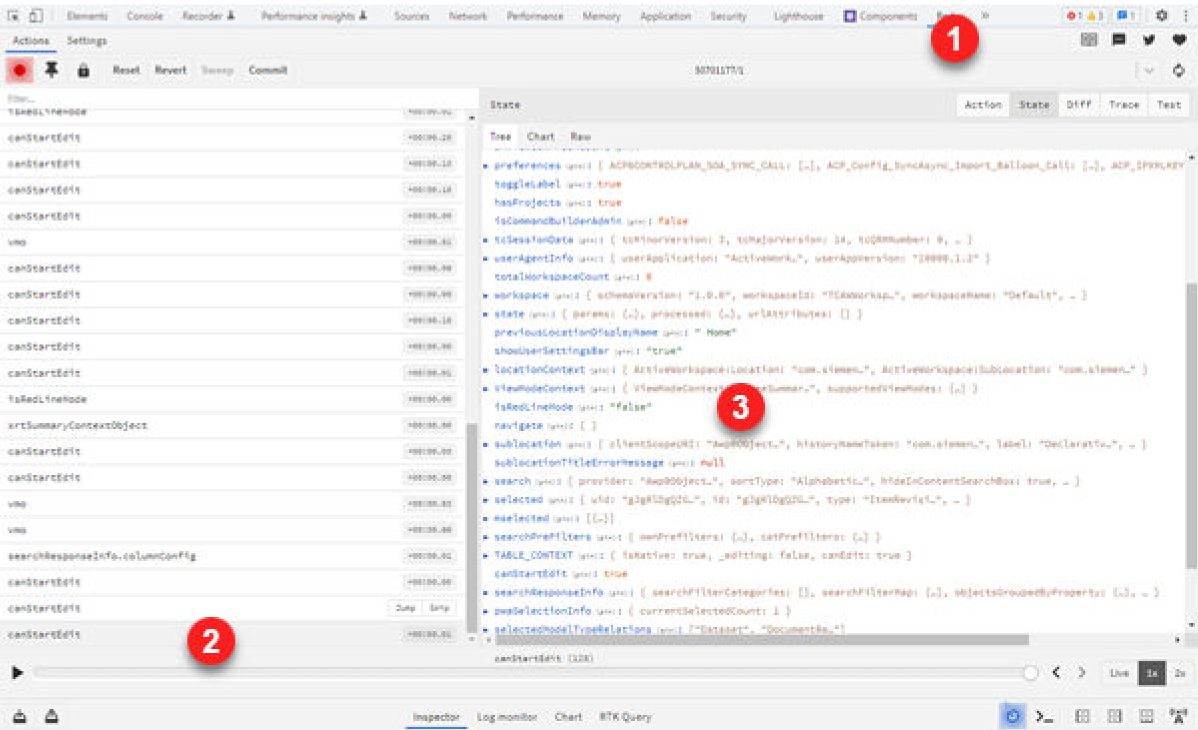
Redux DevTools extension

To view the contents of the **ctx** object, you can use the Redux DevTools extension for the Chrome or Firefox browser.

The Redux UI

To access the Redux interface, you must open the browser's developer tools while logged in to Teamcenter using the Active Workspace client.

© 2025 Siemens



1. Choose the **Redux** tab in the developer tools window.

2. In the left-side window, select an action.

3. In the right-side windows, observe the state tree as it existed during that action.

Exploring the CTX object

Expand the various nodes to discover the information available. The contents of this object changes each time the interface is used, so your results vary depending on where you are and what you are doing at the time.

© 2025 Siemens

```
▸ sublocation (pin): { clientScopeURI: "Awp0Object…", historyNameToken: "com.siemen…", label: "Declarativ…", … }
  sublocationTitleErrorMessage (pin): null
▸ search (pin): { provider: "Awp0Object…", sortType: "Alphabetic…", hideInContentSearchBox: true, … }
▾ selected (pin)
    uid (pin): "gJgRlDgQZGeBtC"
    id (pin): "gJgRlDgQZGeBtC"
    type (pin): "ItemRevision"
    displayName (pin): "033438"
    levelNdx (pin): 1
    childNdx (pin): 0
    iconURL (pin): "assets/image/typeItemRevision48.svg"
    visible (pin): true
    $$treeLevel (pin): 1
    isLeaf (pin): true
  ▾ modelType (pin)
      abstract (pin): false
      displayName (pin): "Item Revision"
      name (pin): "ItemRevision"
      parentTypeName (pin): "WorkspaceObject"
      primary (pin): true
    ▸ typeHierarchyArray (pin): ["ItemRevisi…", "WorkspaceO…", "POM_applic…", "POM_object", …]
      typeUid (pin): "TYPE::ImanType::ImanType::POM_object"
      uid (pin): "TYPE::ItemRevision::ItemRevision::WorkspaceObject"
    ▸ constantsMap (pin): { Fnd0AllowReviseOperation: "true", Fnd0WhereUsed: "true", ReviseInput: "ItemRevisi…", … }
    ▸ propertyDescriptorsMap (pin): { view_SEWeldment: {…}, view_SEDraft: {…}, view: {…}, … }
```

There is a lot of information available in the **ctx** object. Be sure to examine it fully. For example, while the class name of a selected object type is found at

```
ctx.selected.modelType.name
```

the entire class hierarchy is found as an array at

```
ctx.selected.modelType.typeHierarchyArray
```

Following are some examples of declarative conditions using the **ctx** object.

```
"conditions":
{
    "haveSearchResultsAndEditNotActive":
    {
        "expression": "ctx.search.totalFound > 0 && true"
    },
    "alertItemRevisionCommandVisible":
    {
        "expression": "ctx.selected"
    },
    "alertItemRevisionCommandActive":
    {
        "expression": "ctx.selected.type === 'ItemRevision'"
    },
    "alertItemRevisionSpecialCommandActive":
    {
        "expression": "ctx.selected.type === 'ItemRevision' && ctx.selected.props.object_name.dbValue === 'qwerty'"
    }
},
```


© 2025 Siemens

Siemens Web Framework

This functionality is part of the Siemens Web Framework upon which Active Workspace is built. Refer to *Redux Development Tools extension for Google Chrome* within the Digital Engineering Services product on Support Center for more information.

Use logical objects to consolidate properties

Logical Objects

Why would I use a logical object?

Use a logical object to

- Gather properties from various related objects into a single place.
- Eliminate the need for the end user to know the data model by presenting a flat list of properties.
- Share specified properties without exposing others.

What is a logical object?

A logical object is a runtime object designed to consolidate properties. If you have related business objects in Teamcenter and they contain useful properties, you might want to see them all in a single place.

The logical object allows you to:

- Define a set of objects related to a specific business object.
- Define a list of properties found on those objects.

Why would I use a logical object instead of compound properties?

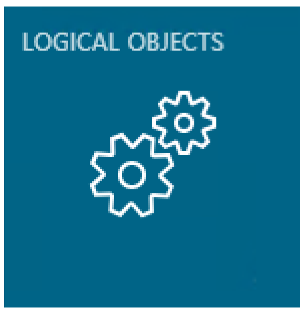
Logical objects are runtime objects that organize properties.

- Logical objects can be created and maintained from within Active Workspace.
- Dynamic compound properties are a display capability. While flexible, they are not real properties nor real objects to be queried or exported.
- Traditional static compound properties are a change to the Teamcenter schema, and must be created using the Business Modeler IDE, and then deployed.

© 2025 Siemens

How do I configure a logical object?

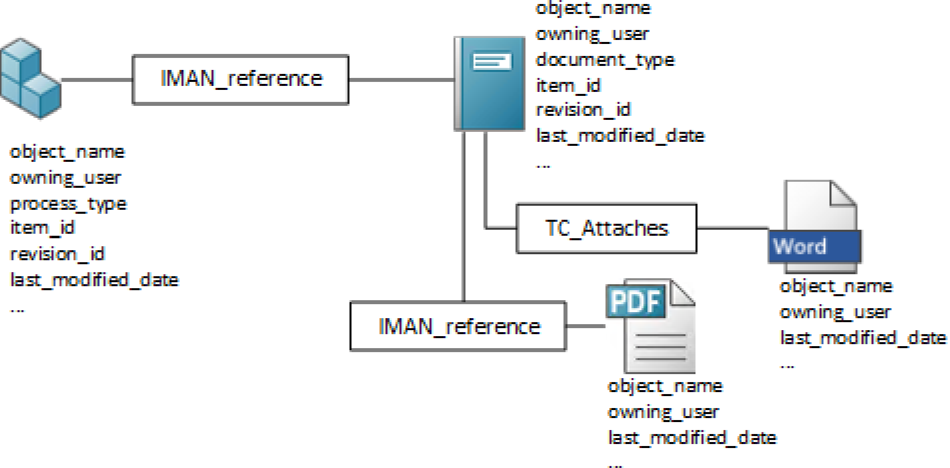
You must use the **Logical Objects** administrative tool in Active Workspace in order to create or modify a logical object. This tile is only visible to administrators.



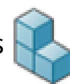
Logical object configuration

Example scenario

In this example, you have a process object which has a reference relation to a document object. That document object has two attached files, one word and one PDF.



You want to consolidate four properties from three of those objects.

- **object_name** and **process_type** from the process 
- **document_type** from the attached document

© 2025 Siemens

- **object_name** from the PDF attached to the document

No properties are desired from the word object , since this is internal to your department.

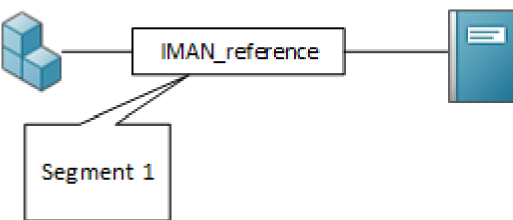
Create a new logical object

1. Log in to Active Workspace with administrator privileges and go to the **Logical Object Configuration** page.

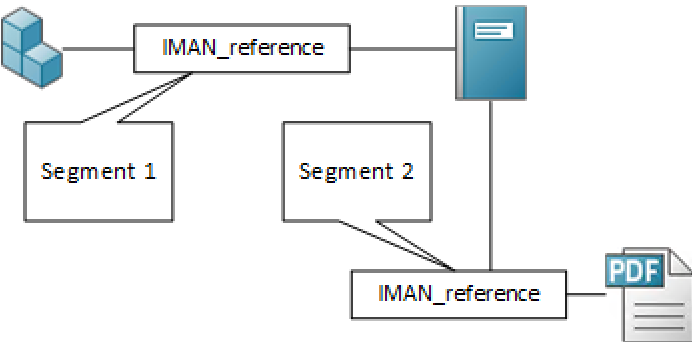
2. Define ⊕ a new logical object, specifying the process object as the root object. In this example you do not want to inherit configurations from other logical objects, so select the **Fnd0LogicalObject** as the parent.

3. Add ⊕ the two **Members** by specifying their relationship and object segments. All member objects must be defined from the root object.

   When adding the document only a single segment is required.

   However, the PDF object is two segments away from the root, and so you must define both segments.

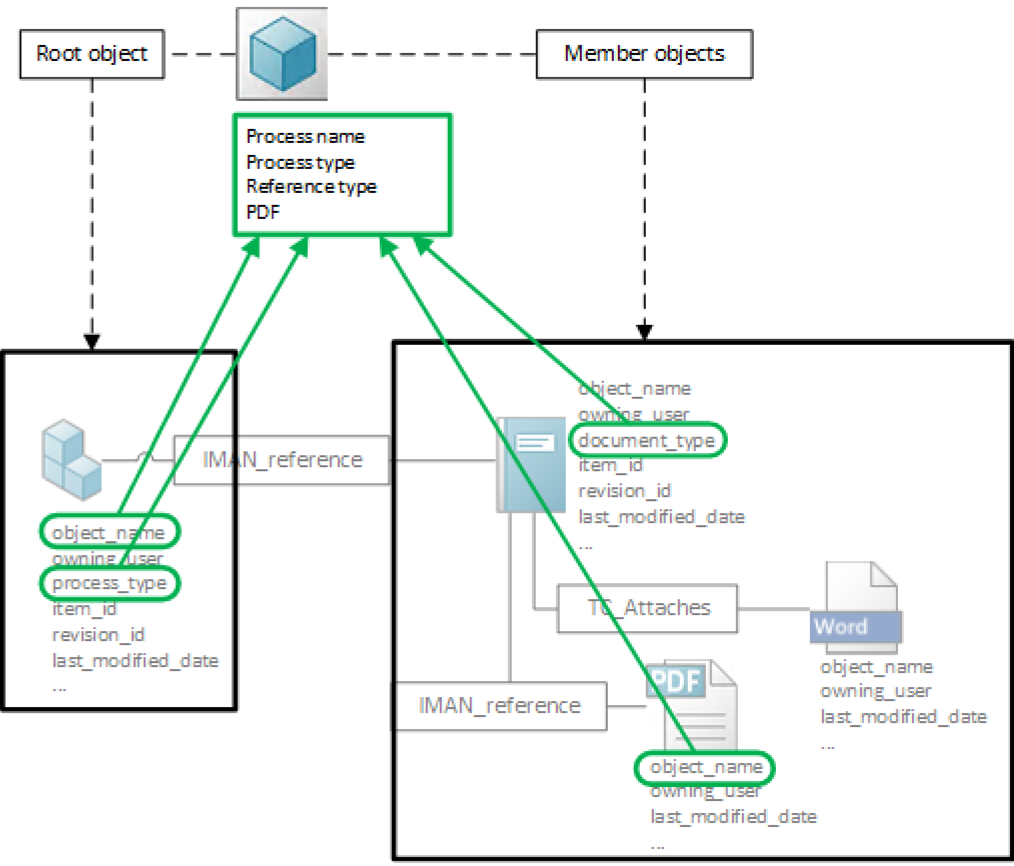
© 2025 Siemens

In this example, there is no need to add the word object.

4. Add the four **Presented Properties**, specifying from which **Member** object they will be retrieved.

Active Workspace automatically saves your progress at each step, so you are done. The new logical object is available.

Result



Edit an existing logical object

You can edit existing logical objects from the **Logical Object Configuration** page.

- To add a new member or property, select the logical object you want to edit and select **Add Member** or **Add Property** near the respective table.
- To modify an existing member, select the row in the **Members** table you want to edit, and then select **Edit Member** .

© 2025 Siemens

The page configuration changes based on what you have selected, so if you do not see the icons you expect, check your table selections.

Destination criteria

What are destination criteria?

A destination criterion is a way for the system to filter out destination objects at runtime.

What destination criteria are available?

Depending on the destination object type, you may have a list of one or more to choose from. Choose only one. If you wish to use more than one criterion, you will need to define a second rule with the same relation and destination object. Following is a list of your choices based on what type of object you choose as your destination object:

- **Type of WorkspaceObject**

  Current user session project

- **Type of Item revision**

  Current user session project

  Choice of configuration contexts

  Choice of revision rules

- **Runtime business object**

  No options

If your destination object is another logical object, then its root type is considered for this purpose, even though logical objects are runtime objects.

What do the criteria types do?

- **Current user session project**

  The destination object will be chosen at runtime based on the user's current project.

- **Choice of configuration contexts**

  The destination object will be chosen at runtime based on the specific revision rule chosen.

© 2025 Siemens

- **Choice of revision rules**

  The destination object will be chosen at runtime based on the specific configuration context chosen. If you chose the **Configuration Context** option, then it will be based on the user's current configuration context.

- **No options**

  The destination object will be chosen at runtime with no special considerations.

How do I use it?

When adding a new **Member** to a logical object or editing an existing member, each segment asks for **Destination Criteria**.

Compound logical objects

What is a compound logical object?

A compound logical object is a logical object that displays the properties from other logical objects.

Why would I use it?

Since it is possible for a single logical object to contain as many properties as you require, and from many target business objects, you may wonder why you would need to reference another logical object at all.

One reason is compartmentalization. Imagine a scenario where several groups are each contributing properties to an overall logical object. Each group would have to edit the members and presented properties of the same logical object. This could become a logistical nightmare.

If instead, each group creates a single logical object from which to present their properties, then a single corporate-wide logical object could include those logical objects from the individual groups. This provides a single overall object for reference while still maintaining each group's ability to modify their own object as needed.

Another reason is to create a base logical object that contains a set of properties that you want several logical objects to share. Then the other logical objects can inherit that configuration, but each add their own properties.

There are two methods to retrieve properties from other logical objects that can be used as needed:

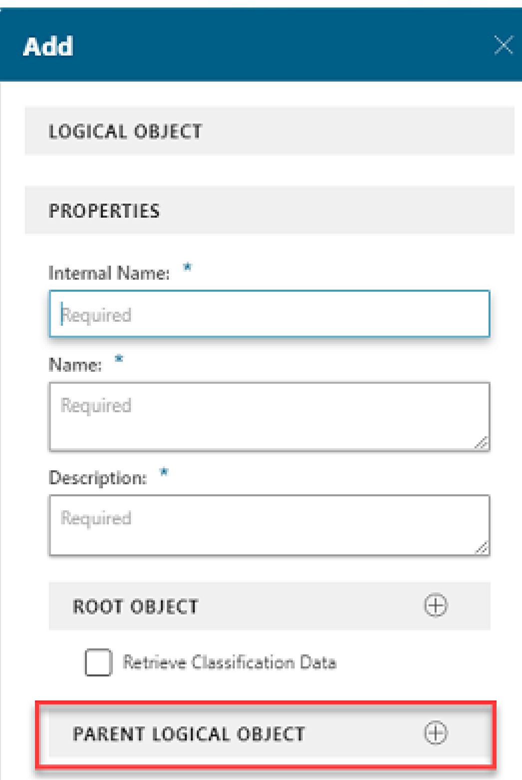
- During creation, specify a **parent logical object** from which to inherit a base set of properties.

- After creation, choose some **included logical objects** and which of their properties to present.

© 2025 Siemens

Parent logical object

When creating a logical object, you may choose one other logical object to be its parent.

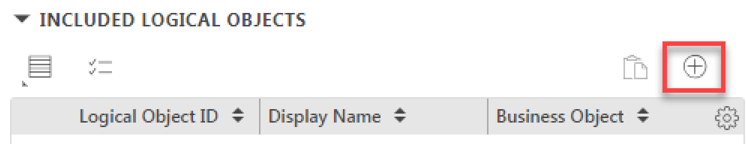


Your new logical object will inherit its configuration from this parent.

Included logical objects

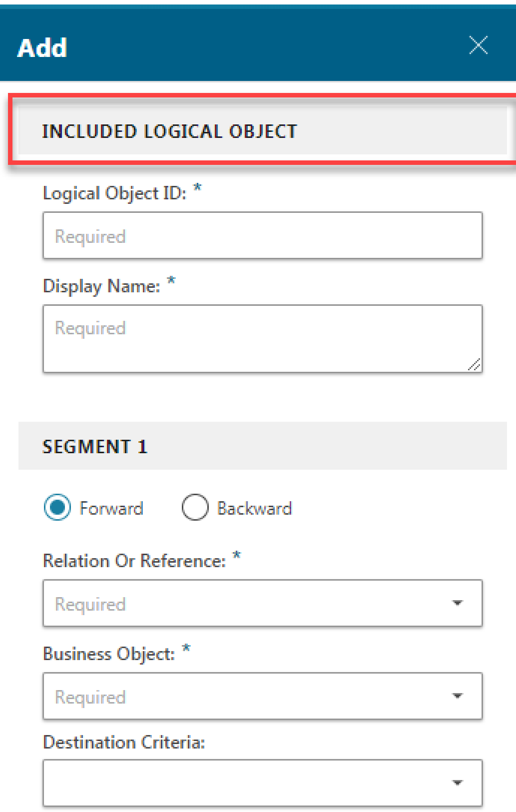
Consolidate properties from other logical objects onto an existing one.

When viewing a logical object, use the **Add** command ⊕ from the **Included Logical Objects** table.



Specify the target logical object as the **Business Object**.

© 2025 Siemens



Workspaces

Learn about workspaces

What are workspaces?

A workspace is a UI configuration that is independent of the Teamcenter organization. Traditionally, Teamcenter uses groups and roles for user organization as a way to control data security, command visibility, workflow tasks, and so on. But in many cases, these groups and roles are defined by corporate standards or maintained by **LDAP** and cannot be manipulated as needed to achieve the desired security and UI configurability. In other cases, the work of configuring duplicate settings for each group and role combination becomes tedious. Workspaces allow you to configure for a role or task once, and then assign it as needed.

What are the benefits?

You can:

- Create reusable configurations, independent of the organization.
- Design them according to real-world roles, tasks, or skills.

© 2025 Siemens

- Assign them to Teamcenter groups and roles.
- Determine which pages and commands are available.
- Control which style sheets are used.
- Decide how property columns in declarative tables are arranged.
- Control home page tile availability.

How do I use workspaces?

- **Define** and **modify** your workspace using a declarative module.
- **Map a workspace to the organization** using *import_wsconfig*.
- Optionally, **Assign style sheets to the workspace** with preferences.
- Optionally, **Assign tile collections to the workspace** with preferences.
- Optionally, **Map column configurations to the workspace** using *import_uiconfig*.
- You can also **Remove a workspace**, if necessary.

What workspaces are available?

Following are some of the workspaces available to the Active Workspace client. This is not a complete list; certain features will install more workspaces as needed.

These workspaces are provided as examples that you can use as-is or as reference to create your own. Siemens Digital Industries Software recommends that you create your own *exclusive* workspaces for your users, with a task-based organization in mind.

- **Author - TcAuthorWorkspace**

  This is an *exclusive* workspace which limits users to a select group of commands and pages for creating content. Assign this workspace to your CAD designers, simulation engineers, and so on.

  > Tip:
  >
  > This is the default workspace. You may change the default workspace for your site by using the **AWC_Default_Workspace** preference.

- **Consumer - TcConsumerWorkspace**

© 2025 Siemens

This is an *exclusive* workspace which limits users to a select group of commands and pages for viewing content. Assign this workspace to your users that are not authors and are assigned a consumer license. For example shop floor personnel, sales team members, and so on.

- **Active Amin - TcActiveAdminWorkspace**

  This optional workspace is an *exclusive* workspace which limits users to a select group of administrative commands and pages.

- **Active Architect - TcActiveArchitectWorkspace**

  This optional workspace is an *exclusive* workspace which limits users to a select group of commands and pages for modifying the user interface.

- **Simplified - TcXSimplifiedWorkspace**

  This optional workspace is an *exclusive* workspace which provides a streamlined UI for basic document management users who have no access to other Teamcenter applications or integrations.

- **Default - TCAWWorkspace**

  An *inclusive* workspace, meaning that it has access to *all* pages and commands. This workspace is not meant for use in your production environment, but is provided for use in a development environment so you can explore content right away without having to do an initial configuration.

Create a custom workspace definition

There are two parts to a workspace: client and server. The client-side portion of a workspace is defined using a **JSON** file which is created in the **stage\src\solution** directory.

Create a new workspace

In this example, you use the *aw* script to create a new workspace **myExclusiveWorkspace**. Open a command prompt, navigate to your **stage** directory, and run **initenv** to create an Active Workspace development environment. From that environment, run the following command.

```
aw add workspace
? Specify the ID of the workspace myExclusiveWorkspace
? Specify the name of the workspace Example
{select from list} Exclusive
{select from list} myTasks
✔ Added workspace 'Example' (myExclusiveWorkspace).
```

The utility automatically creates the workspace definition file and registers it in Active Workspace's **kit.json** file.

© 2025 Siemens

kit.json

The OOTB kit file is automatically updated with the name of your custom module and your workspace ID.

```
"solutionDef": {
    "solutionId": "TcAW",
    "solutionName": "Teamcenter",
    [illegible]
    "workspaces": [
        "TCAWWorkspace",
        "TcAuthorWorkspace",
        "TcConsumerWorkspace",
        "TcActiveAdminWorkspace",
        "TcActiveArchitectWorkspace",
        "myExclusiveWorkspace"
    ],
```

Optionally, use the **includeInSolutions** entry in your workspace definition file to register your workspace without the need to modify the *kit.json* file.

> Note:
>
> Siemens Digital Industries Software recommends against using *inclusive* workspaces except for exploration and testing. Create and use only *exclusive* workspaces in your production environment.
>
> Active Workspace ignores the **defaultWorkspace** entry in *kit.json* and instead uses the **AWC_Default_Workspace** preference. This allows you to easily change the default workspace without having to rebuild the application.

workspace_{workspaceName}.json

This file contains the definition of your workspace. It must follow the naming convention shown and be next to the kit file. If you used the *aw* utility, this is created automatically.

© 2025 Siemens

```
workspace_myExclusiveWorkspace.json
solution > workspace_myExclusiveWorkspace.json > ...
1  {
2      "schemaVersion": "1.0.0",
3      "workspaceId": "myExclusiveWorkspace",
4      "workspaceName": "Example",
5      "workspaceType": "Exclusive",
6      "defaultPage": "myTasks",
7      "availablePages": [
8          "myTasks"
9      ]
10 }
```

Tip:

There are many OOTB workspace files to use as reference.

schemaVersion (Mandatory)

Version of the declarative schema to which this workspace definition complies.

workspaceId (Mandatory)

A unique identifier string for the workspace.

workspaceName (Mandatory)

A JSON structure providing lookup details for the localizable string of the workspace name. The **source** attribute points to the message file, and the **key** attribute tells the system which definition to use.

workspaceType (Mandatory)

A string value specifying the type of workspace. There are two choices.

- Exclusive: The user will only be able to see the UI elements which are exclusively mapped in the **availablePages** attribute. This is the recommended selection.
- Inclusive: The user will see *all* UI elements defined within the solution. This is *not* recommended for your production environment.

defaultPage (Mandatory)

The page which will be shown by default when the user logs in or changes workspace. The value is one of the *states* defined by the solution or in your custom *states.json*.

availablePages (Optional)

An array of pages which can be navigated by the user while working within the workspace. Each value is one of the *states* defined by the solution or in your custom *states.json*. You may also include all commands from an entire kit by specifying "**kit::kitname**" in the list.

© 2025 Siemens

includeInSolutions (Optional)

Instead of adding the **workspaceId** to the *kit.json* file, it is possible to register your workspace indirectly from in the workspace itself. The solution name for Active Workspace is **TcAW**.

```
"schemaVersion" : "1.0.0",
    "workspaceId": "..." ,
    "workspaceName": {
        "source": "...",
        "key": "..."
    },
    "workspaceType": "...",
    "includeInSolutions": [
        "TcAW"
    ],
    "defaultPage": "...",
```

Modify your custom workspace definition

Since the main purpose of a workspace is to control which pages and commands are available to the user, you will want to configure your workspace definition to fit your needs.

Following are the three main sections of the workspace definition file that control which UI elements are available or are restricted. None of these have any effect in an **inclusive** workspace definition.

> Tip:
>
> Workspaces only control the *declarative* interface. Any commands that are provided using style sheets are outside the control of the workspace and may still be available to the user. Be certain to review your style sheets, and register custom style sheets to your workspace if desired.

availablePages

List the pages that you want the user to have access to. For convenience, you can include all pages from a kit as well by specifying **kit::** in front of the kit name.

For a list of kits, see the *STAGE\repo* directory in your **development environment**.

> Example:
>
> To include all pages defined within the **workflow** kit, use:
>
> ```
> "availablePages": [
>     "kit::workflow"
> ]
> ```


© 2025 Siemens

includedCommands

If you use this option, the workspace will only include the commands listed. Similar to pages, you can include all commands from a kit by specifying **kit::** in front of the kit name.

Example:

The following includes a list of eight specific commands, plus all the commands defined within the **workflow** kit.

```
"includedCommands": [
    "Awp0ObjectInfo",
    "Awp0GoBack",
    "cmdSignOut",
    "cmdViewProfile",
    "Awp0ManageGroup",
    "Awp0HelpGroup",
    "Awp0Help",
    "Awp0HelpAbout",
    "kit::workflow"
]
```

excludedCommands

There are two ways you can use this option:

- If you *also* specify included commands, then the workspace will subtract these excluded commands *from that included list*.
- If you *do not* specify any included commands, then the workspace will *automatically include all commands* in the workspace, and then subtract these.

Example:

The following excludes two specific commands from the workspace. They will not be available through the declarative interface. Remember however, that a style sheet could still present them.

```
"excludedCommands": [
    "Awp0SuspendTask",
    "Awp0AbortTask"
]
```

Similar to pages, you can exclude all commands from a kit by specifying **kit::** in front of the kit name.

© 2025 Siemens

Modify an OOTB workspace definition

Using a workspace contribution file, you can modify which commands and pages are available to users of an out-of-the-box (OOTB) workspace.

It is not recommended to modify the existing workspace definition files that are provided OOTB. Instead, create a workspace contribution file in the same directory (**STAGE***src**solution*). This file allows you to contribute the same definition sections as if they were in the target workspace definition file.

File name pattern

Workspace contribution file names must begin with *workspace_contribution*, but the rest of it is up to you. It is recommended that you also add the name of the target workspace somewhere in the file name.

Mandatory fields

schemaVersion

The version of the declarative schema to which this workspace contribution definition complies.

workspaceId

The target workspace to be modified.

Optional fields

These are the same fields as you will find in the workspace definition file. Full explanations are found in **modifying your custom workspace**.

availablePages

An array of pages (declarative states) which can be navigated by user working in the workspace. When a sub location is specified, the associated location must also be included.

includeCommands

An array of commands IDs which *can* be used by a user working in the workspace.

excludedCommands

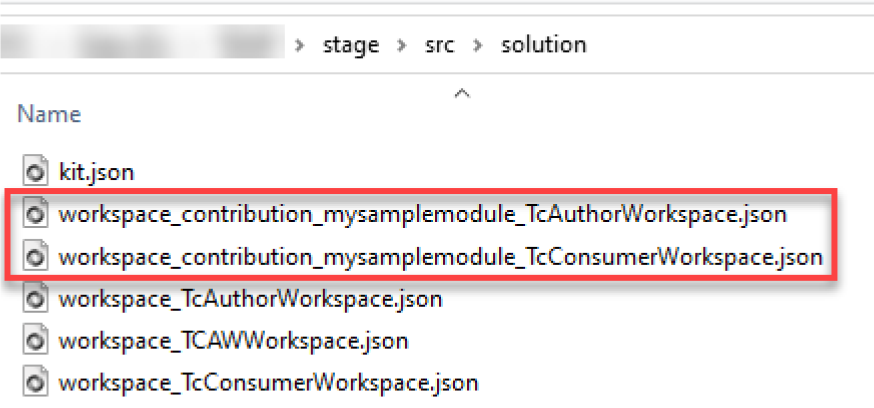
An array of commands IDs which *can not* be used by a user working in the workspace.

Example

You want to add your custom sub location (**mySubLocation**) to the **Author** and **Consumer** workspaces.

1. You must create a workspace contribution file for each target workspace. You decide to add the name of your custom declarative module (**mysamplemodule**) as well as the name of the target workspace to the file names.

© 2025 Siemens



2. Add the sub location ID and its associated location ID in the list of available pages.

   workspace_contribution_mysamplemodule_TcAuthorWorkspace.json

```
{
    "schemaVersion": "1.0.0",
    "workspaceId": "TcAuthorWorkspace",
    "availablePages": [
        "mySubLocation",
        "myLocation"
    ]
}
```

   workspace_contribution_mysamplemodule_TcConsumerWorkspace.json

```
{
    "schemaVersion": "1.0.0",
    "workspaceId": "TcConsumerWorkspace",
    "availablePages": [
        "mySubLocation"
        "myLocation"
    ]
}
```

3. Build your environment as appropriate.

Your custom sub location will now be available in the two workspaces.

Create or update workspace mappings

You can create workspace mappings by first exporting the list of existing workspace mappings provided with Active Workspace, adding your new workspace, and then importing the list. You can also update existing workspaces with these utilities.

© 2025 Siemens

Export a list of existing workspaces

Use the export_wsconfig utility to export workspace definitions from Teamcenter. This is a Teamcenter platform command, and must be run in a Teamcenter command-line environment. Information on how to *manually configure the Teamcenter environment* can be found in the Teamcenter *Utilities Reference* documentation.

In the following example, all existing workspace definitions are exported.

```
export_wsconfig -u=xxx -p=xxx -g=dba -file=c:\temp\ws.xml
```

You can use the `-for_workspace` option to export a specific workspace.

Modify the workspace list

Following is an example of what your exported file might look like.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Import>

  <Workspace id="TcActiveAdminWorkspace">
    <WorkspaceMapping group="dba" role=""/>
    <WorkspaceMapping default="true" group="dba" role="DBA"/>
  </Workspace>

  <Workspace id="TcAuthorWorkspace">
    <WorkspaceMapping default="true" group="Simulation" role="engineer"/>
    <WorkspaceMapping default="true" group="Product" role="desginer"/>
    <WorkspaceMapping group="dba" role="DBA"/>
  </Workspace>

   ...

</Import>
```

Many of these examples are mapped to organization groups and roles. Some are labeled as `default=true`. Use the exported examples and the following rules to create your own workspaces.

You may automatically map a workspace to all subgroups of a group by using an asterisk. You must map the parent group first.

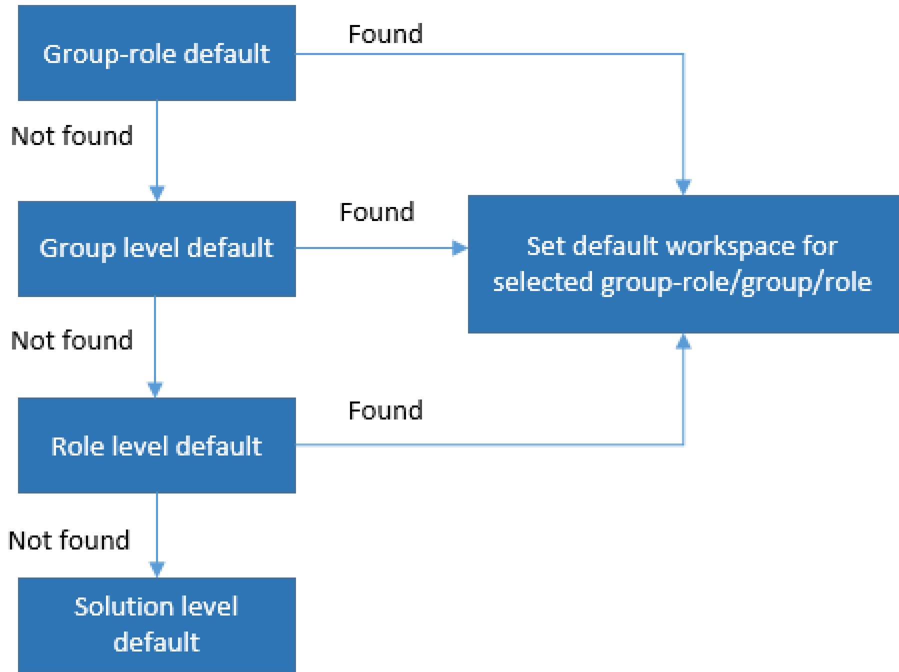
> Example:
>
> In this example, you map a workspace to the Engineering group, and then map it to all Engineering subgroups.

© 2025 Siemens

```
    <WorkspaceMapping group="Engineering"/>
    <WorkspaceMapping group="Engineering.*"/>
```

- If `group=""` and `role` is assigned to something, then this workspace will apply to that role regardless of group.
- If `role=""` and `group` is assigned to something, then this workspace will apply to every role within that group.
- If both `group` and `role` are assigned, then this workspace will only apply to that combination.
- If `default="true"`, then this workspace takes precedence over others for that group or role.

If one or more workspaces are set to `default="true"` for a given group or role, then when a user logs in, the default workspace is picked in following order:



Solution level default

For the selected group-role combination, if there is no workspace defined by Teamcenter organization mappings, then the value of the **AWC_Default_Workspace** preference is used. The **defaultWorkspace** setting in Active Workspace's solution *kit.json* file is ignored.

Unique default workspace

The *import_wsconfig* utility only allows unique entry of a default workspace for the same group, role, or group-role combination. If you want to override an existing default workspace for specified

© 2025 Siemens

group, role, or group-role, then you need to set the existing default workspace to false, the new workspace to true, and then re-import the mappings.

Import your custom workspace

You use the *import_wsconfig* utility to import your custom workspace definitions.

In the following example, you import your custom file *ws.xml*, which contains your workspace definitions.

```
import_wsconfig -u=xxx -p=xxx -g=dba -file=c:\temp\ws.xml
```

Assign style sheets to a workspace

You can associate XML rendering templates (XRT), also known as style sheets, to a workspace similar to how you assign them to a group or role. When a user is in a specific workspace, Active Workspace will look for the workspace-specific XRT to render. If it cannot find one, then it will proceed as normal.

The **XRTEditor** does not support workspace-based style sheets at this time.

To associate an XRT to a workspace, follow the same process you would normally use for groups or roles, with the following differences.

Preference name syntax and precedence

To associate a style sheet with a workspace, use the workspace name in addition to **AWC**. The preference name syntax is similar to regular, non-workspace, style sheet preference naming.

workspace XRT preference name

AWC_*workspaceId*.*typeName*.*location*.*sublocation*.*stylesheetType*

non-workspace XRT preference name

AWC_*typeName*.*location*.*sublocation*.*stylesheetType*

Following is the precedence for these workspace preferences. As usual, the more specific assignments will take priority over the less specific.

- AWC_*workspaceId*.*typeName*.*location*.*sublocation*.*stylesheetType*
- AWC_*workspaceId*.*typeName*.*location*.*stylesheetType*
- AWC_*workspaceId*.*typeName*.*stylesheetType*
- If none of these are found, then it will proceed as normal, looking for preferences starting with AWC_*typeName* per the normal rules.

© 2025 Siemens

For example, you would create the following preference to assign an XRT to the summary page of a revision when it is part of an assembly when the user is in the **myWorkspace** workspace.

```
AWC_myWorkspace.ItemRevision.showObjectLocation.OccurrenceManagementSubLocation.SUMMARYRE
NDERING
```

Assign a tile collection to a workspace

You can configure a tile collection with a workspace as its scope.

This is preliminary functionality.

How do I assign a workspace as a tile collection scope?

The general process to assign a workspace as a scope instead of a group or role, for example, is essentially no different; the tile collection object references the scope object with its **awp0Scope** property.

Using the platform command-line utility

You can use a Teamcenter platform command-line utility to perform the association of a workspace to a tile collection scope. This command reads an XML file that contains the definition for the tile collection scope assignments.

> Caution:
>
> Siemens Digital Industries Software does not support the use of this utility for any other purpose.

```
aws2_install_tilecollections -u=xxx -p=xxx -g=dba -mode=add
-file=c:\temp\mytilecollections.xml
```

At this time, there is no export functionality, only import. If you wish to use this utility, you must copy this XML content and modify it.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE ActiveWorkspaceGateway SYSTEM
"Awp0aws2ActiveWorkspaceGateway.dtd" >
<ActiveWorkspaceGateway version="1.0">
    <!-- Tile Template definitions begins here -->
    <TileTemplate templateId="myTileTemplate">
        <ThemeIndex index="1" />
        <Icon>homefolder</Icon>
        <Action>myAction1</Action>
        <ActionType type="3" />
    </TileTemplate>

    <!-- Tile definitions begins here -->
```


© 2025 Siemens

```
        <Tile tileId="myTile" templateId="myTileTemplate">
            <Name>My Custom Tile</Name>
        </Tile>
        <!-- Tile collection definitions begins here -->
        <TileCollection>
            <WorkspaceScope id="myCustomizerWorkspace" />
            <CollectionTiles tileId="myTile" groupName="main"
    size="0" ></CollectionTiles>
        </TileCollection>
    </ActiveWorkspaceGateway>
```

Tip:

To assign the tile collection to all users, replace

```
<WorkspaceScope id="..." />
```

with

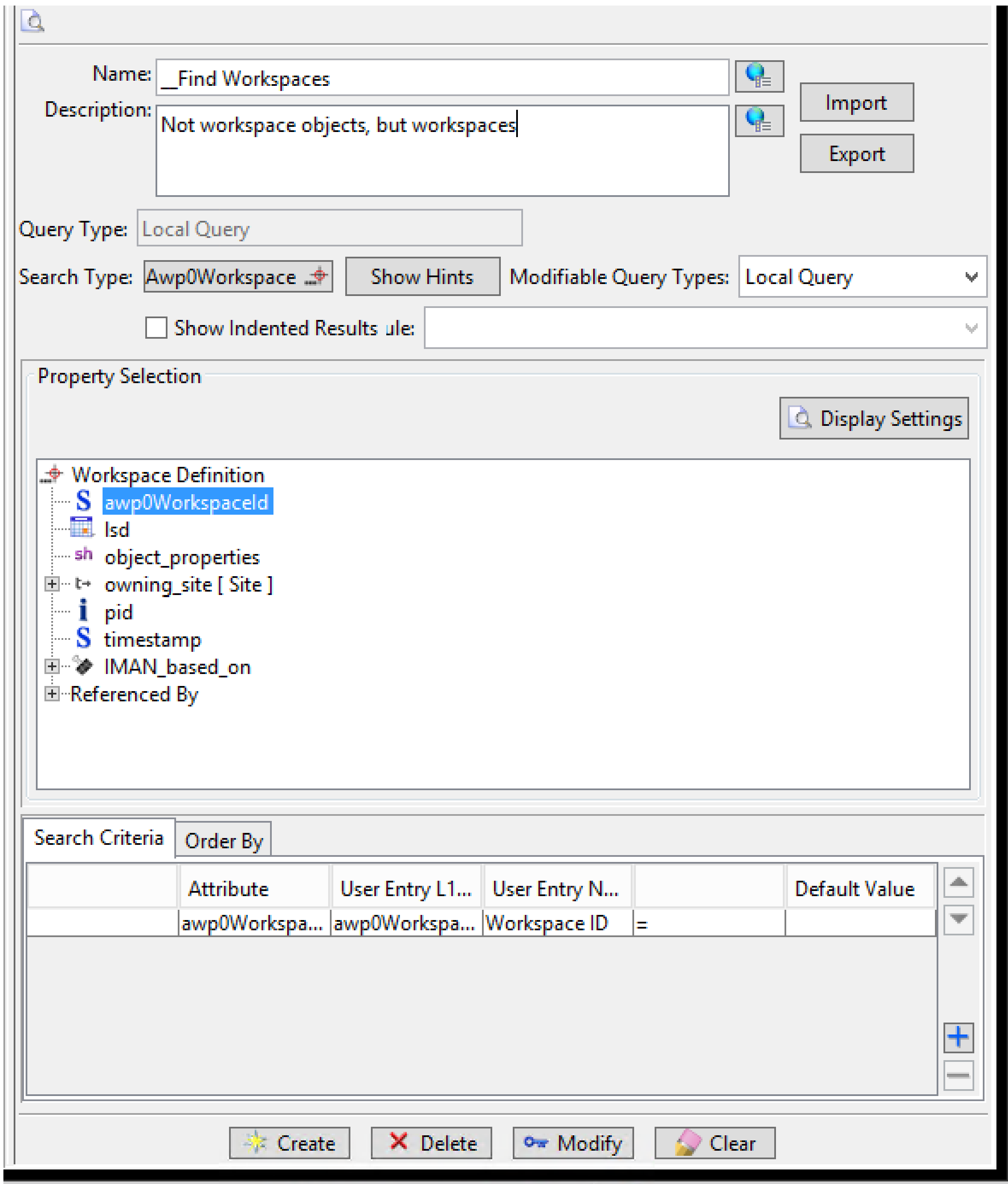
```
<SiteScope/>
```

instead.

Using the rich client

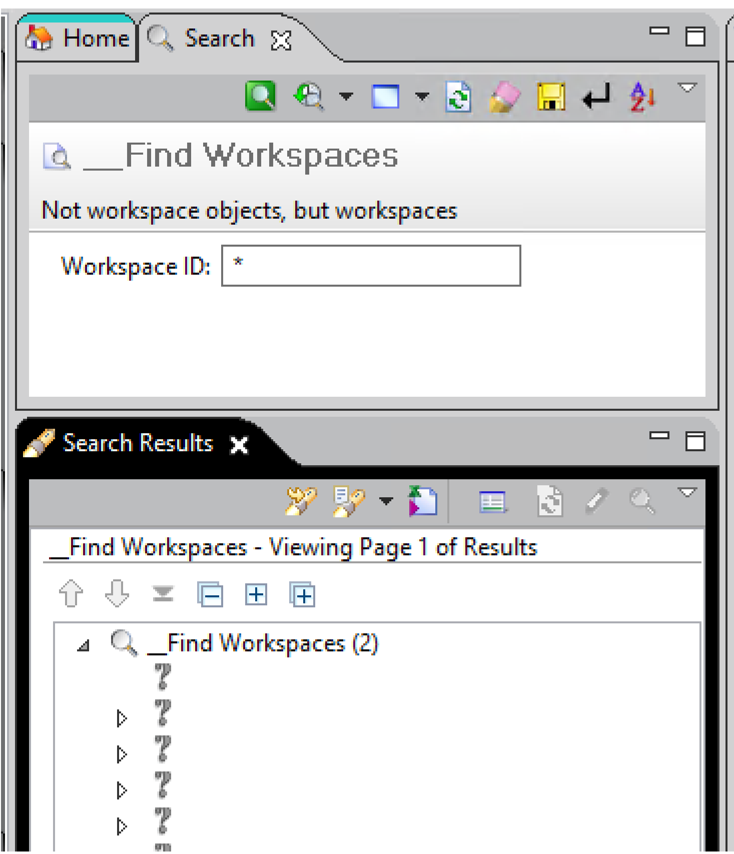
The rich client was never designed to display the newly designed **Awp0Workspace** object type. The only way to find them using the rich client is to create a custom query designed specifically to find **Awp0Workspace** objects, and the search results will appear very strange.

© 2025 Siemens

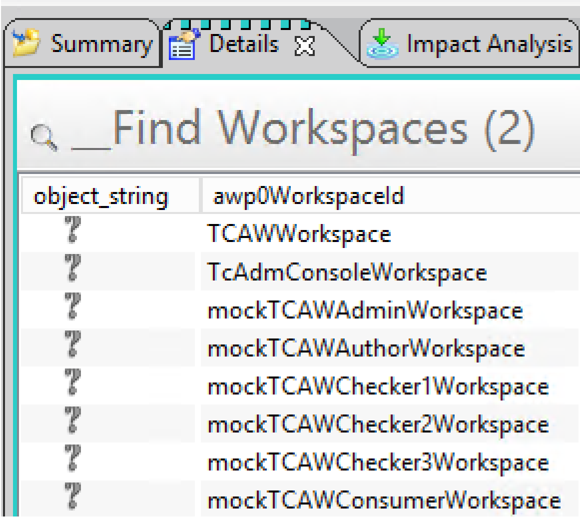


Since **Awp0Workspace** is not a child of **WorkspaceObject** it has neither an **object_name** property nor a default icon. The rich client relies on the **object_string** property, which is based on **object_name**.

© 2025 Siemens



In order to see which workspace is which, you need to examine their **awp0WorkspaceId** property. You can add this property to the column layout in the **Details** view to make it easier to find which workspace is which.



Now you can copy and paste them like you would a group, or role, and so on.

© 2025 Siemens

Create or modify column configuration for a workspace

You can use the **export_uiconfig** and **import_uiconfig** utilities to create column configurations associated with a workspace.

Export an existing column configuration

You use the export_uiconfig utility to export a column configuration from Teamcenter. This is a Teamcenter platform command, and must be run in a Teamcenter command-line environment. Information on how to *manually configure the Teamcenter environment* can be found in the Teamcenter *Utilities Reference* documentation.

In the following example, existing column configurations are exported.

```
export_uiconfig -u=xxx -p=xxx -g=dba -file=c:\temp\ui.xml
```

You can use the `-for_workspace` option to export the column configuration for a specific workspace.

Modify the column configuration

Following is an example of what your exported file might look like.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Import>

  <Client abbreviation="AWClient" name="AWClient">
    <ClientScope hostingClientName="" name="ReuseView" uri=...
      <ColumnConfig columnConfigId="reuseViewConfig" sortBy=...
        <ColumnDef columnName="pgp1CurrentModel" objectType=...
        ...
        <ColumnDef columnName="pgp1TargetType" objectType=...
      </ColumnConfig>
    </ClientScope>
  </Client>

  <Client abbreviation="AWClient" name="AWClient">
    <ClientScope hostingClientName=""
name="Ret0RetailLinePlanAllViewURI" uri=...
      <ColumnConfig columnConfigId="lineplanAllColumnView" sortBy=...
        <ColumnDef columnName="plp0Product" objectType=...
        ...
        <ColumnDef columnName="ret0TotalNoOfColors" objectType=...
      </ColumnConfig>
    </ClientScope>
  </Client>

...
```


© 2025 Siemens

```
</Import>
```

Use the exported examples as a guide to create your own column configuration file (*myui.xml* for example) or modify the existing one.

Import your custom column configuration

You use the import_uiconfig utility to import your custom column configuration file.

In the following example, you import *your custom* workspace definition file, *myui.xml*, for use in your custom workspace.

```
import_uiconfig -u=xxx -p=xxx -g=dba -for_workspace=myCustomizerWorkspace
-file=c:\temp\myui.xml
```

Precedence

Column configurations are retrieved with the following precedence:

- GroupMember
- Workspace
- User
- Role
- Group
- Site

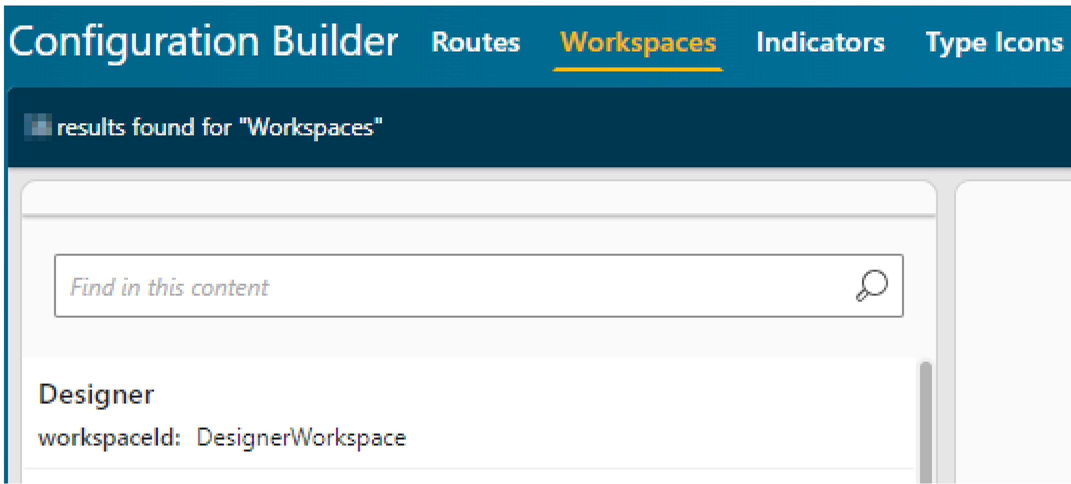
The only way a column configuration with the **GroupMember** scope is created is if a user manually modified theirs using the UI. The user can remove their custom column configurations by using the **Reset** command in the column configuration panel of the UI.

Verifying your new workspace

You can verify that your workspace has been successfully created by viewing the **Workspaces** tab in the **Configuration Builder**. If you have administrative permissions, the **Config Builder** command is displayed on the global navigation toolbar (**aw_globalNavigationbar**).

This location contains a list of the **Workspaces** which are available in your environment.

© 2025 Siemens



Remove a workspace

You can remove a custom workspace by removing each component.

- **Client-side**

  Remove the client-side portion of the workspace by removing the declarative definition. This is a JSON file located in your **STAGE***src**solution* folder.

  If your workspace was added to the **solutionDef** portion of the *kit.json* file, remove its entry.

```
"solutionDef": {
  "solutionId": "TcAW",
  "solutionName": "Active Workspace",
...
  "workspaces": [
      "TCAWWorkspace",
      "TcAuthorWorkspace",
...
      "MyCustomWorkspace"
  ],
  "defaultWorkspace": "TcAuthorWorkspace",
```

- **Server-side**

  Delete the server-side configuration by using the **-action=delete** argument of the import_wsconfig utility.

© 2025 Siemens

XRT element reference

Active Workspace style sheet elements

You can control the layout of certain portions of the declarative interface by using XML rendering templates (XRT), also called style sheets. These XML files are stored in the Teamcenter database and are read as needed, so changes made to these rendering templates are reflected in the UI without the need to build or deploy an application file.

You will need to validate the positioning and usage of these elements manually. There is no schema for XRT.

Top elements

One of these elements must be the overall element for the XML file.

<rendering>

The overall wrapper element for the panel's XML file.

<subRendering>

Used instead of **rendering** when the XML file is *injected* into another XRT.

Main elements

The main rendering of the view typically consists of a single header followed by one or more pages. Headers typically contain property elements, and pages typically consist of any number of the property, container, or layout elements.

<header>

Displayed at the top of the rendered view.

<page>

The way to organize properties onto multiple pages.

Property elements

These elements display information to the user. They are the reason for the rendering. All other elements are for organization and ease-of-consumption by the user.

<property>

Displays a single property by name. This is the database name of the property, not the localized display name. You can choose a property on the selected object, or a related object.

<objectSet>

© 2025 Siemens

Displays a table of properties from related objects.

<tableProperty>

Displays a special property that is a table. Contrast this with the **<objectSet>**, which is a *collection of individual properties* in a table format, whereas the **tableProperty** is a *single property* that contains a table of information.

<classificationTrace>

Displays the classification hierarchy information for an object, if present.

<classificationProperties>

Displays the classification properties and the hierarchy information for an object, if present.

Container elements

These elements help you group and organize your property elements.

<column>

Creates a column of properties.

<section>

Creates a visible, collapsible grouping.

<content>

Provides logical grouping, typically for conditional content.

<inject>

Allows you to insert additional XRT content from another file.

<htmlPanel>

Allows you to insert HTML content into the panel.

Layout elements

These elements help you separate and highlight your property elements.

<separator>

Inserts a visible separator between other elements.

<break>

Inserts empty space between other elements.

<label>

Insert a static text string between other elements.

© 2025 Siemens

Modifying style sheets using the XRT Editor

You can use the **XRTEditor** to view and edit the style sheets that are used to render certain content within Active Workspace.

This editor allows you to directly edit the XRT controlling the current page's layout. The editor will automatically find the associated **XMLRenderingStylesheet** dataset, and present it for view and edit.

Starting the editor

By default, administrative users have access to the **XRT EDITOR** tile on the home page.

The editor will open in a new browser tab. For convenience, you may move this tab to its own browser window.

Using the editor

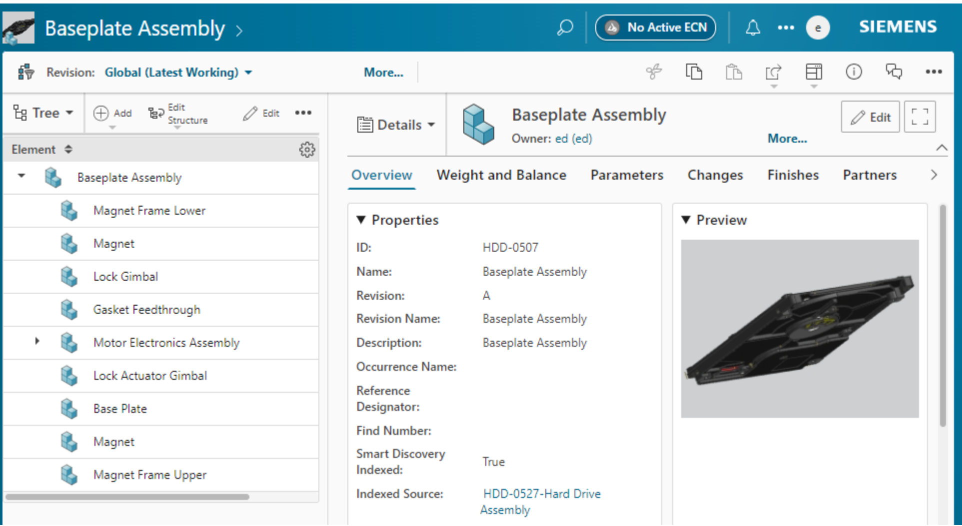
The editor in the secondary tab is now linked to your primary tab. The editor will follow your navigations in the primary tab, displaying the style sheet used to render each page when applicable.

Since editor window is linked to the first window, the editor will follow your navigations in the first window, displaying the style sheet used to render each page when applicable.

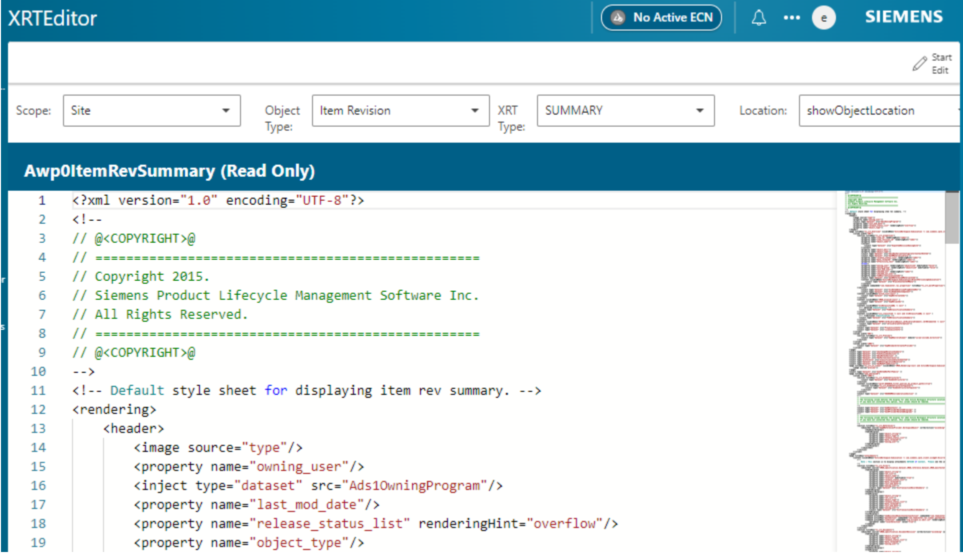
Example: Item revision summary

In this example, you have navigated to an assembly with your primary browser tab.

© 2025 Siemens



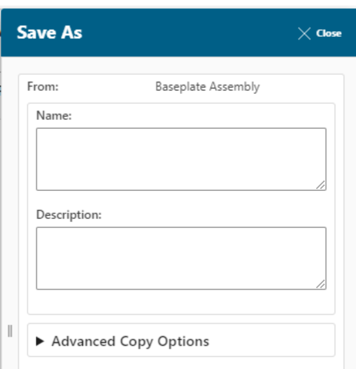
Meanwhile, the editor in the secondary browser tab has detected that a style sheet is used, and is displaying the XRT contents of the style sheet as well as the registration information.



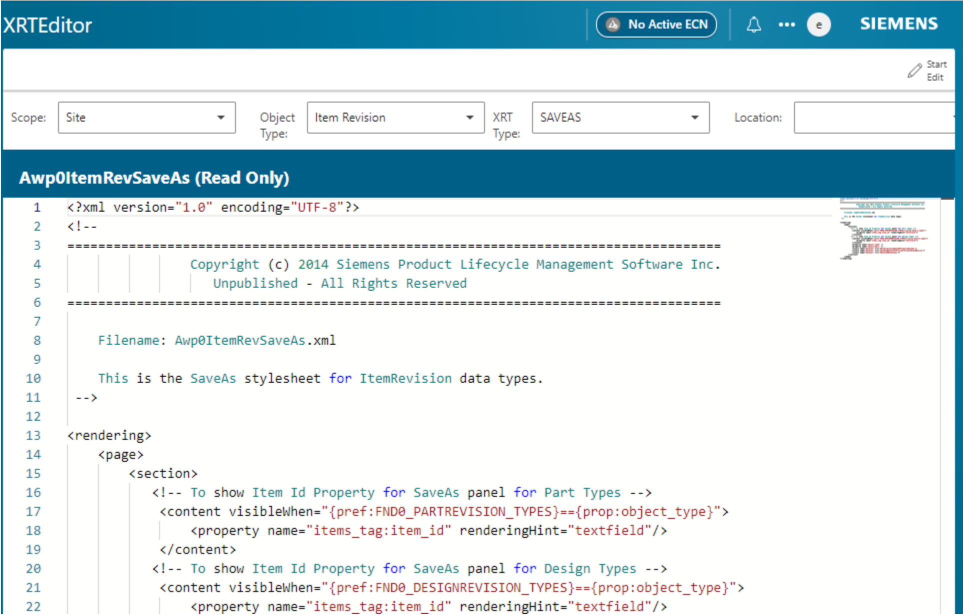
Example: Item create

In this example, you have opened the **Save As** panel with your primary browser tab.

© 2025 Siemens

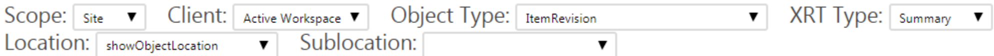


Meanwhile, the editor in the secondary browser tab has detected that a style sheet is used, and is displaying the XRT contents of the style sheet as well as the registration information.



Alternate usage

It is also possible to use the drop-down menus and the **Load** button to select an XRT by its registration.



This does not require navigation, and can be used to make edits as needed if you already know what you are looking for.

© 2025 Siemens

break

You can insert a non-visible break between elements. This appears as additional space.

USAGE

This element is typically used alongside the various property elements.

ATTRIBUTES

None.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<break>** element:

```
<section titleKey="tc_xrt_properties">
   ...
   <property name="effectivity_text" renderingHint="label"/>
   <inject type="dataset" src="Cm1AuthoringChange"/>
   <break/>
   <property name="owning_user" renderingHint="objectlink" modifiable="false"/>
   <property name="owning_group" renderingHint="objectlink" modifiable="false"/>
   ...
</section>
```

Effectivity:
Authoring Change:

Owner: Engineer,Ed (ed)
Group ID: demo

© 2025 Siemens

classificationTrace

You can display the classification trace (hierarchy) of the object. For example:

TC Classification Root > Classification Root > Material Families > Bar Families > Solid Bar > Rectangular Bar.

USAGE

This element is typically used within the header element, but may be used anywhere the various property elements can be used.

ATTRIBUTES

None.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

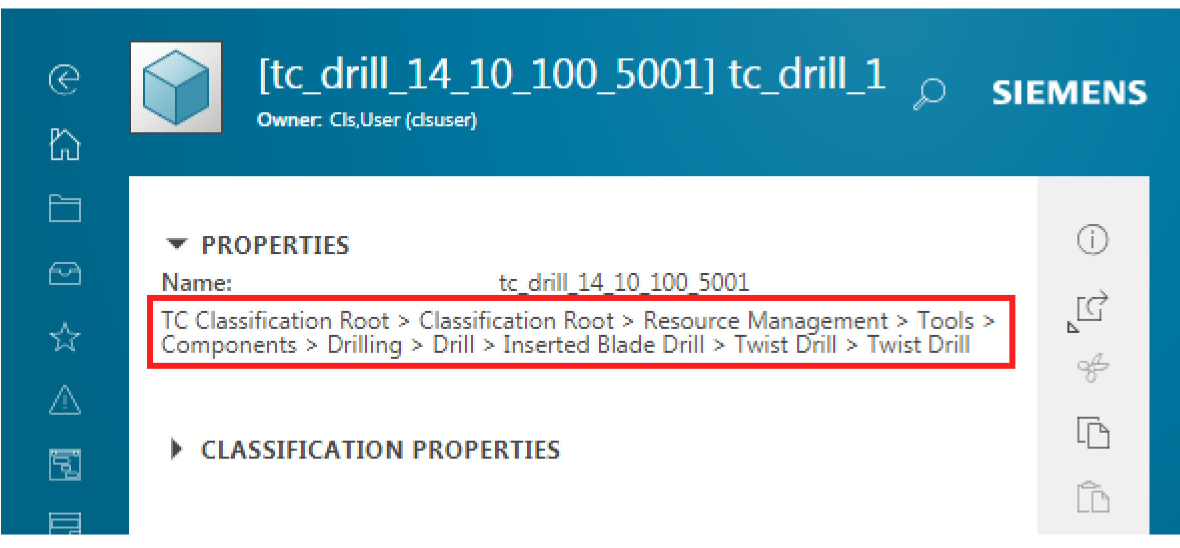
Summary

EXAMPLE

Following style sheet snippet shows the **<classificationTrace>** element:

```
<column>
    <section titleKey = "tc_xrt_properties">
        <property name="object_name"/>
        <classificationTrace />
    </section>
    <section titleKey="tc_xrt_ClassificationProperties">
        <classificationProperties/>
    </section>
</column>
```


© 2025 Siemens


[tc_drill_14_10_100_5001] tc_drill_1
Owner: Cls,User (clsuser)
SIEMENS
PROPERTIES
Name: tc_drill_14_10_100_5001
TC Classification Root > Classification Root > Resource Management > Tools > Components > Drilling > Drill > Inserted Blade Drill > Twist Drill > Twist Drill
CLASSIFICATION PROPERTIES

classificationProperties

You can display the classification properties of the object, including the classification trace. Properties and their values are rendered as name/value pairs in static text.

USAGE

This element may be used anywhere the various property elements can be used.

ATTRIBUTES

None.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary

EXAMPLE

Following style sheet snippet shows the **<classificationProperties>** element:

```
<content visibleWhen="ics_classified!=null">
   <section titleKey="tc_xrt_ClassificationProperties">
      <classificationProperties/>
   </section>
</content>
```


© 2025 Siemens

© 2025 Siemens

column

The column element is a container element that can help organize your style sheet content.

In the XRT hierarchy, sections and columns must not be siblings. Typically, columns are children of the page element and help make the page easier to read due to the typical wide-screen layout.

ATTRIBUTES

width (optional)

This is a percentage of the overall screen width, even if the percent sign is not used.

If the column percentages total less than 100%, the empty space will not fill.

If the column percentages total more than 100%, overflow columns are placed on a new row.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

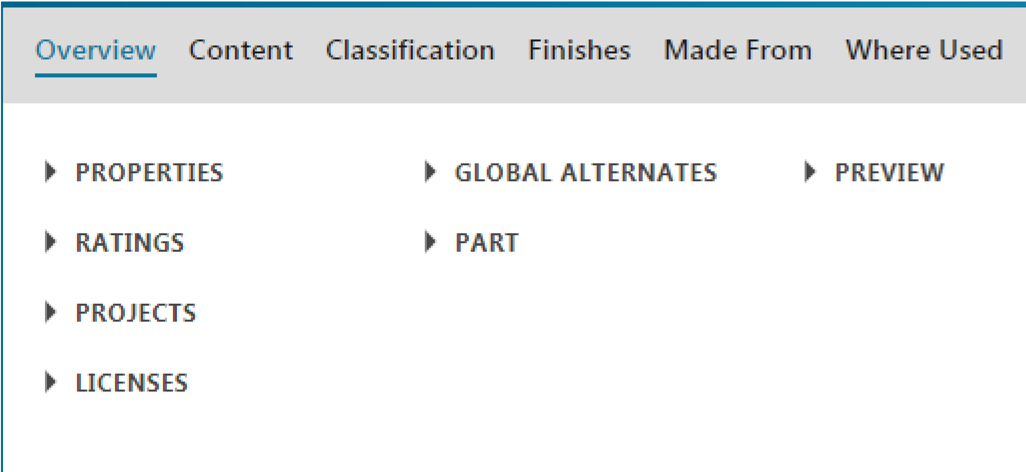
Summary

EXAMPLE

Following style sheet snippet shows the **<column>** element:

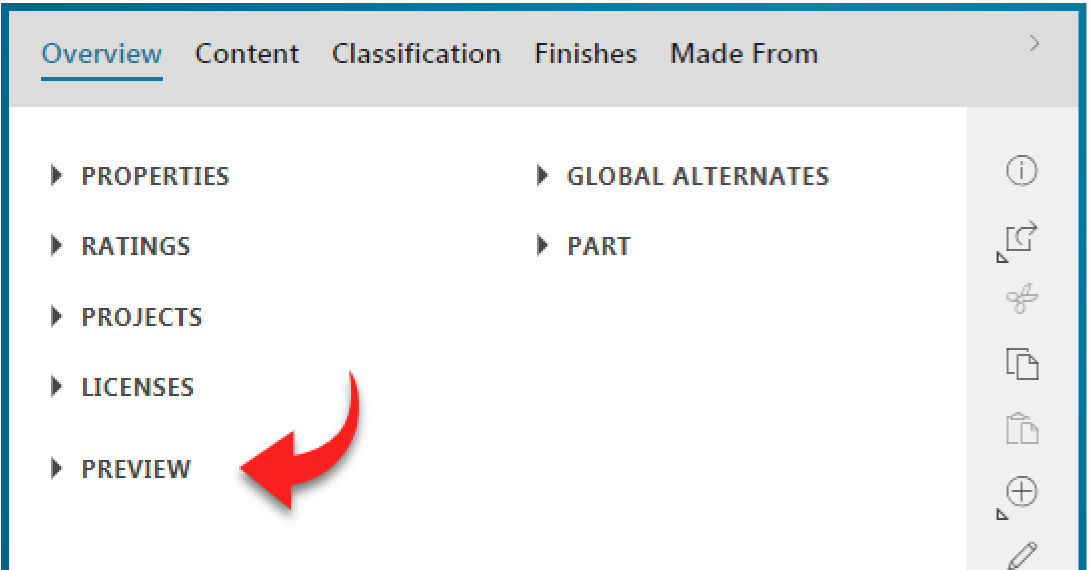
```
<page titleKey="tc_xrt_Overview" ...>
    <column width="30%">
        <section titleKey="tc_xrt_properties">
            ...
        </section>
        <inject type="dataset" src="S2clScalarRatingOverview"/>
        <inject type="dataset" src="ProjectListInfo"/>
        <inject type="dataset" src="LicenseListInfo"/>
    </column>
    <column width="25%">
      <inject type="dataset" src="Awp0GlobalAlternatesProvider"/>
      <inject type="dataset" src="Fgb0AlignedPartsProvider"/>
    </column>
    <column width="45%">
        <section titleKey="tc_xrt_Preview">
            ...
        </section>
    </column>
</page>
```


© 2025 Siemens



ADDITIONAL INFORMATION

As the browser tab narrows, the contents of extra right-hand columns will move onto the end of the first column.



command

Specifies a command representation to be displayed on an object set table. Use of the `<command>` element anywhere else on an Active Workspace style sheet is not supported.

The **Cut** command is not supported as an XRT element.

ATTRIBUTES

commandId

Specifies the command to be executed. The attribute value must be a key into a property file and must be a valid command ID.

THE <PARAMETER> ELEMENT

Following are some commands that utilize the parameter element.

Copy

localSelection = **true** (required)

You must provide a target for the **Copy** command, which is passed from the UI when local selection is **true**.

```
<command commandId="com.teamcenter.rac.copy" ... >
   <parameter name="localSelection" value="true"/>
</command>
```

Add

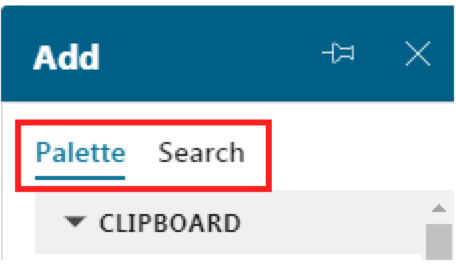
visibleTabs = {**new | palette | search**} (optional)

You can specify which panels are available when the user creates an object. If this parameter is not used, all available tabs will appear.

Following is an example of displaying only the palette and search tabs, effectively hiding the new tab.

```
<command commandId="com.teamcenter.rac.common.AddNew">
   <parameter name="visibleTabs" value="palette,search"/>
</command>
```


© 2025 Siemens



> Caution:
>
> Do not leave spaces in the comma-separated list of tabs.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary

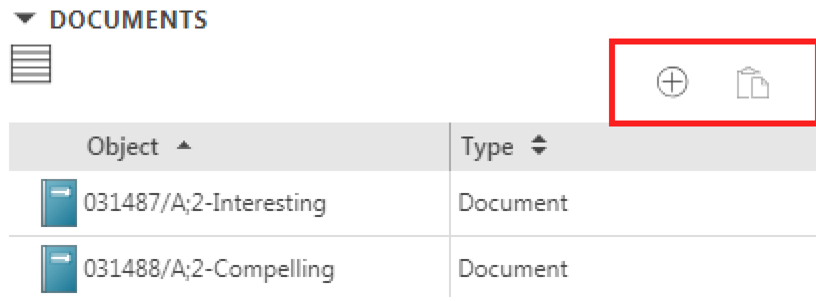
The **command** tag is ignored in the **header** section.

EXAMPLE

Following style sheet snippet shows the **command** element:

```
<objectSet source="..." sortdirection="..." sortby="..." defaultdisplay="...">
    <tableDisplay>
        <property name="..."/>
        <property name="..."/>
    </tableDisplay>
    <thumbnailDisplay/>
    <listDisplay/>
    <command commandId="com.teamcenter.rac.common.AddNew"/>
    <command commandId="com.teamcenter.rac.viewer.pastewithContext"/>
    </command>
</objectSet>
```

In this example, the **command** element adds the **Add New** and **Paste** buttons in the object set.



© 2025 Siemens

content

The content element is a logical container element that can help organize your XRT elements. Use this element if you want conditional control the display of a property, a group of properties, a section, and so on.

USAGE

You may use the content element to group all other elements, including other content elements, but not rendering elements.

ATTRIBUTES

visibleWhen

Uses a condition to determine if the element is displayed. You can specify many types of conditions with the visibleWhen attribute.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary

EXAMPLE

Following style sheet snippet shows the **<content>** element:

```
<column>
   <section titleKey="...">
      <property name="..."/>
      <property name="..."/>
      ...
      <break/>
      <content visibleWhen="...">
         <section titleKey="DCP Properties">
            <property name="..." titleKey="..."/>
            <property name="..." titleKey="..."/>
            ...
         </section>
      </content>
   </section>
   <content visibleWhen="...">
      <section titleKey="Custom Properties">
         <property name="..."/>
         <property name="..."/>
         ...
      </section>
   </content>
   ...
</column>
```


© 2025 Siemens

EXAMPLE

Following is sample code from the **ItemSummary.xml** XML rendering style sheet showing the **section** element:

```
<page titleKey="tc_xrt_Overview">
    <column>
        <section titleKey="tc_xrt_AvailableRevisions">
        </section>
        <section titleKey="tc_xrt_ItemProperties">
        </section>
        <section titleKey="tc_xrt_ClassificationProperties">
        </section>
    </column>
    <column>
        <section titleKey="tc_xrt_Preview">
        </section>
        <section titleKey="tc_xrt_actions" commandLayout="vertical">
        </section>
    </column>
</page>
```


© 2025 Siemens

header

Specifies the content of the header area.

USAGE

This is typically at the top level of the style sheet, a direct child of the rendering or subrendering elements.

ATTRIBUTES

None.

SUPPORTED STYLE SHEETS

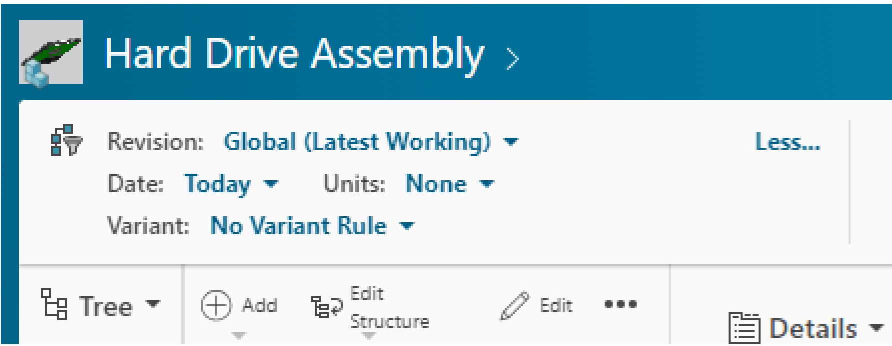
This element is only used on the following style sheets:

Summary

EXAMPLE

Following style sheet snippet shows the **<header>** element:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<rendering>
   <header>
      <image source="type"/>
      <property name="owning_user"/>
      <property name="last_mod_date"/>
      <property name="release_status_list"/>
      <property name="object_type"/>
   </header>
...
</rendering>
```



© 2025 Siemens

ADDITIONAL INFORMATION

The **<header>** element is optional. If it is not included, or if it does not contain any elements, the header is automatically populated with the **object_string** property as a label. This label is not selectable.

The following elements may be contained within a **<header>**:

- `<image>`
- `<property>`
- `<inject>`

  The injected dataset can only contain property elements.

© 2025 Siemens

htmlPanel

You can insert HTML code into your style sheet.

ATTRIBUTES

Use these attributes to define where the indirect HTML is located.

declarativeKey

The name of a declarative **View** file to insert.

src

The URL of a web page to insert.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE DECLARATIVEKEY

Following style sheet snippet shows how to implement a declarative view as your html content. In this example, the **Rv1RelationsBrowserView.html** declarative view will be inserted. Just like any declarative view, there must be a corresponding view model file, and any other supporting files that may be needed.

Optionally, use the **enableresize** attribute to control the user's ability to change the size of the declarative panel.

```
<htmlPanel declarativeKey="Rv1RelationsBrowser" enableresize="true"/>
```

EXAMPLE SRC

Following style sheet snippet shows an inserted web page from the customer's site, including a reference to a property. You must specify any properties used in the URL by using the **<property>** element inside the HTML panel element. When used in this way the property element is not displayed in the UI, it is only present to ensure the property is loaded in the client.

```
<htmlpanel src="www.customersite.com/parts/app/
{{selected.properties['item_id'].dbValue}}">
   <property name="item_id" />
</htmlpanel>
```


© 2025 Siemens

inject

You can break up larger XRT content into smaller, more manageable, logical groups of elements, and then inject them back into the main rendering file. You can also inject HTML content instead.

The target dataset must either be an **XMLRenderingDataset** or **HTML** dataset.

Avoid using a large amount of **<inject>** elements in your XML rendering templates. It can negatively impact the performance of the client.

ATTRIBUTES

These attributes define the way the dataset is located.

type

Specify whether you will give the name of a dataset, or the name of a preference that contains the name of the dataset.

dataset	Use this option if you will *directly* specify the name of the dataset.
preference	Use this option if you will *indirectly* specify the name of the dataset through a preference. This allows you to take advantage of the ability to have the value of a preference vary based on a user's credentials.

src

Specify the name of the dataset or preference.

- If `type="dataset"`, this is the name of the dataset that contains the code to be injected.
- If `type="preference"`, this is the name of the preference that contains the name of the dataset that contains the code to be injected.

visibleWhen

Uses a condition to determine if the element is displayed. You can specify many types of conditions with **the visibleWhen attribute**.

> Note:
>
> Do not use the **visibleWhen** attribute with the inject element to check the object type. Use multiple style sheets, each registered to a different object instead.
>
> Do not attempt to create a single, over-arching XRT for all object types.

© 2025 Siemens

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<inject>** element:

```
<page titleKey="tc_xrt_Overview" ...>
   <column width="30%">
         ...
      <inject type="dataset" src="S2clScalarRatingOverview"/>
      <inject type="dataset" src="ProjectListInfo"/>
      <inject type="dataset" src="LicenseListInfo"/>
   </column>
   <column width="25%">
      ...
   </column>
   <column width="45%">
      ...
   </column>
</page>
```

Dataset: **S2clScalarRatingOverview**

```
<subRendering>
    <section titleKey="tc_xrt_Ratings">
        <htmlPanel declarativeKey="ratingOverViewPanel"/>
    </section>
</subRendering>
```

Dataset: **ProjectListInfo**

```
<subRendering>
    <section titleKey="tc_xrt_Projects">
        <property name="owning_project" renderingHint="label"/>
        <property name="project_list"/>
    </section>
</subRendering>
```

Dataset: **LicenseListInfo**

```
<subRendering>
    <section titleKey="tc_xrt_Licenses">
```


© 2025 Siemens

```
        <property name="license_list"/>
    </section>
</subRendering>
```


© 2025 Siemens

label

ATTRIBUTES

These attributes define

textKey

Specifies the key used for localization. If it is not defined or otherwise found, the string defined by the **text** attribute is used.

text (optional)

Specifies a static string of the title for this tab. This attribute is used when the **textKey** is not found by localization.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

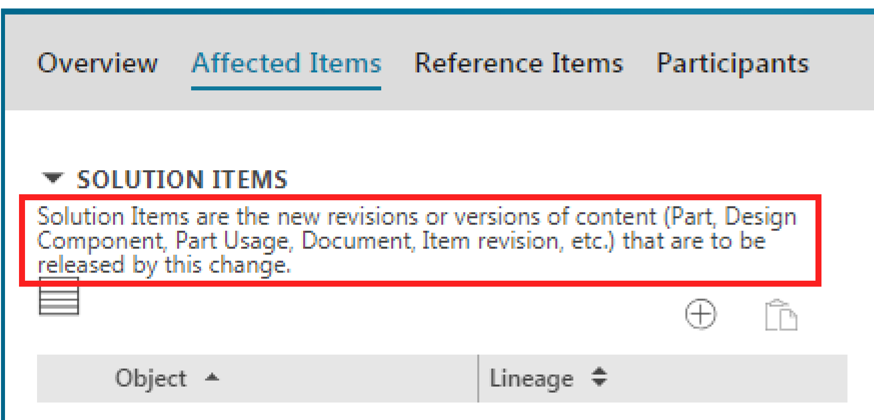
Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<label>** element:

```
<page title="Affected Items" titleKey="tc_xrt_AffectedItems">
   <section titleKey="tc_xrt_ProblemItems" title="Problem Items">
      <label textKey="ProblemItemsInfo" text="..."/>
         <objectSet source = "..." defaultdisplay = "..." sortdirection = "..." sortby =
"...">
            ...
         </objectSet>
   </section>
   <section titleKey="tc_xrt_ImpactedItems" title="Impacted Items">
      ...
   </section>
   ...
</page>
```


© 2025 Siemens



USING A URL IN A LABEL

If you add a URL address in the **text** attribute, it will be automatically rendered.

Example:

```
<section titleKey="tc_xrt_properties">
    ...
    <property name="object_type"/>
    <separator/>
    <label text="***My label URL http://www.siemens.com ***" />
    <separator />
    ...
</section>
```

Because a URL is included in the **label** tag, it is automatically rendered on the page.

PROPERTIES

ID: 031486

Revision: A

Name: Test Item

Description:

Type: Item Revision

***My label URL
http://www.siemens.com
***

Release Status:

Date Released:

Effectivity:

© 2025 Siemens

nameValueProperty

Name-value properties are a specialized form of the table property designed for name-value pairs.

To display a name-value property in Active Workspace, use the **nameValueProperty** tag instead of the **objectSet** tag in your custom summary style sheet.

Attributes

name (required)

The name of the name-value property you want to display.

source (optional)

Specifies the related object from which to retrieve the name-value property. You must **use dynamic compound properties to specify a source**.

sortBy (optional)

Specifies the object property to sort the set of objects by prior to rendering. If this is not specified, the rows will be displayed in the order they were created.

sortDirection (optional)

Specifies the direction in which the set of objects on the **nameValueProperty** should be sorted.
Valid values are **ascending** or **descending**.
The default value is **ascending**.

enablePropEdit (optional)

Set this to **false** to disable the user's ability to automatically start editing on this table. While this is disabled, If the user wants to make changes to values on this table, they must start editing manually.

maxRowCount (optional)

The maximum number of rows displayed in the **nameValueProperty** regardless of available vertical space.

The <property> element

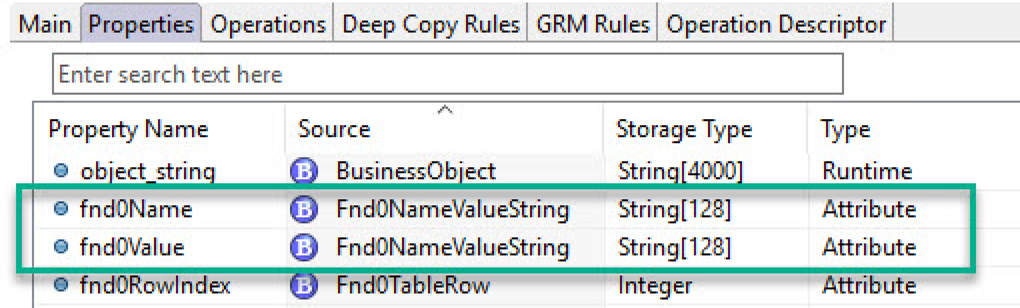
Use the **<property>** element inside the **<nameValueProperty>** element. You *must* use the **fnd0Name** and **fnd0Value** properties.

```
<section title=...>
   <nameValueProperty
name="myNVProp">
      <property name="fnd0Name"/>
      <property name="fnd0Value"/
>
```


© 2025 Siemens

```
    </nameValueProperty>
</section>
```

OOTB name-value pair properties use **fnd0Name** and **fnd0Value**.



Supported style sheets

This tag is supported on the following types of style sheets:

Summary

Examples

Example:

This example shows a name-value property on the selected object.

```
<nameValueProperty name="aw2_NameValue_Pair">
  <property name="fnd0Name"/>
  <property name="fnd0Value"/>
...
</nameValueProperty>
```

Example:

This example uses DCP to show a name-value property on a related object.

```
<nameValueProperty source="REF(IMAN_specification,AW2_NameValueRevision)"
               name="aw2_NameValue_Pair">
  <property name="fnd0Name"/>
  <property name="fnd0Value"/>
...
</nameValueProperty>
```


© 2025 Siemens

objectSet

You can use an object set in your style sheets to place a table of properties in the UI. These properties can be from the selected object, or from related objects. The initial definition of the table columns is defined by the **objectSet**, but any changes a user makes, like arranging, hiding, or showing columns, for example, are tracked in an auto-created column configuration.

ATTRIBUTES

defaultdisplay (required)

Specifies the default format to use when displaying the set of objects. The value must be any valid **display element**.

source (required)

Specifies the comma-delimited set of run-time properties or relations that return the desired set of objects. The format for the attribute value is *relation.type*, where *relation* is the name of a relation, run-time, or reference property or a GRM relation, and *type* represents the type of objects to be included. You can also **use dynamic compound properties to specify a source**. Multiple sources can be specified as a comma-separated list.

Subtypes of the specified object type are recognized, so specifying the **ItemRevision** type will also return all **DocumentRevision** objects, **Design Revision** objects, and so on. When using a relation however, you must *explicitly* specify the relation. Subtypes of the specified relation are not considered.

The source definition also determines which properties are available to the display types. If you specify a parent object, but the property you are interested in is defined on one of the child objects, then the property may not display properly. Add the child object as a source as well.

showConfiguredRev (optional)

If any of the defined *sources* is an item type, then setting this attribute to **true** uses revision rules to determine which of the item's revisions to display instead of the item itself.

maxRowCount (optional)

The maximum number of rows displayed in the **tableDisplay** regardless of available vertical space.

sortby (optional)

Specifies the object property to sort the set of objects by prior to rendering.

sortDirection (optional)

Specifies the direction in which the set of objects on the **tableDisplay** should be sorted. Valid values are **ascending** or **descending**.
The default value is **ascending**.

enablePropEdit (optional)

© 2025 Siemens

Set this to **false** to disable the user's ability to automatically start editing on this table. While this is disabled, If the user wants to make changes to values on this table, they must start editing manually.

filterable (optional)

Allow this column to be filtered by the user using the UI. The type of filter shown to the user will depend on the data type of the column. dates and numbers are automatically detected and will allow a range filter, but everything else is treated as a string. Set to **false** to prevent filtering.

The default is **true**.

You can set **filterable=false** on individual properties in the table display, or set it on the source to prevent filtering for all columns.

Example:

Filtering will be disabled on all columns, regardless of their individual settings.

```
<objectSet source= ... filterable="false">
    <tableDisplay>
        <property name= ... />
        <property name= ... />
        ...
    </tableDisplay>
    ...
</objectSet>
```

Example:

Only the second column will have filtering disabled.

```
<objectSet source= ... >
    <tableDisplay>
        <property name= ... />
        <property name= ... filterable="false"/>
        ...
    </tableDisplay>
    ...
</objectSet>
```

hidden (optional)

Allow this column to be hidden by default. The benefit of hiding a column is that it allows the user to expose the property if they want, but otherwise the client will not load the information which saves time when loading a page. Set to **true** to hide the column.

The default is **false**.

Explicitly setting **hidden=false** serves no purpose.

© 2025 Siemens

Example:

The second property will not be displayed on the table, but a user could arrange their table to show it.

```
<objectSet source= ... >
    <tableDisplay>
        <property name= ... />
        <property name= ... hidden="true" />
        ...
    </tableDisplay>
    ...
</objectSet>
```

DISPLAY ELEMENTS

The following elements are defined within an object set to determine the availability of the display types.

<tableDisplay>

<listDisplay>

<imageDisplay>

<thumbnailDisplay>

The **<tableDisplay>**'s columns can be defined using a list of **<property>** elements. If a user customizes the table by arranging or hiding columns, a column configuration will automatically be created for that user to store their changes, and then any property elements in the style sheet will be ignored. If that user ever resets their columns, the table will revert back to the object set's property elements.

You can remove the user's ability to arrange the columns by setting the **enableArrange** attribute to **false**.

```
<tableDisplay enableArrange="false">
```

MODIFY AVAILABLE COMMANDS

Use the Command Builder to customize the list of commands that are displayed on an object set. One common uiAnchor for object sets is **aw_objectSet_right**.

EXAMPLE OF SETTING OBJECTSET SOURCE

In these examples, you are creating the source definition for an object set on *folders*.

- You want to display any common object that is in a folder. The **Folder** object uses the **contents** property to maintain this list. The **WorkspaceObject** is the parent of all common user-facing objects.

© 2025 Siemens

```
<objectSet source="contents.WorkspaceObject">
```

- You only want to display documents. You decide that this would include the **DocumentRevision**, **MSWordX**, and **PDF** object types. This time, even though the folder might contain other objects, they are ignored and the table only displays your documents. This will also include children of the **DocumentRevision**, like the **Specification Revision** object, for example.

```
<objectSet source="contents.DocumentRevision,contents.PDF,contents.MSWordX">
```

FULL OBJECTSET EXAMPLE

The following example was taken from an OOTB style sheet

```
<objectSet source="IMAN_specification.Dataset,IMAN_reference.Dataset,
IMAN_specification.Dataset,IMAN_reference.Dataset,
IMAN_manifestation.Dataset,IMAN_Rendering.Dataset,
TC_Attaches.Dataset,IMAN_UG_altrep.Dataset,
IMAN_UG_scenario.Dataset,IMAN_Simulation.Dataset"
   defaultdisplay="listDisplay"
   sortby="object_string"
   sortdirection="ascending">
   <tableDisplay>
      <property name="object_string"/>
      ...
   </tableDisplay>
   <thumbnailDisplay/>
   <listDisplay/>
</objectSet>
```


© 2025 Siemens

objectSetSection

You can use an object set section in your style sheets to place a table of properties in the UI. These properties can be from the selected object, or from related objects. The initial definition of the table columns and any changes a user makes, like arranging, hiding or showing columns for example, are tracked in a column configuration.

ATTRIBUTES

titlekey (optional)

Specifies the key used for localization. If it is not defined or otherwise found, the string defined by the **title** attribute is used.

title (optional)

Specifies a static string of the title for this tab. This attribute is used when the **titleKey** is not found by localization.

initialstate (optional)

Specifies whether the view or section should be expanded or collapsed on initial rendering. Valid values are **expanded** or **collapsed**. The default value is **expanded**. This attribute is optional.

You may also use the **collapsed** attribute, but Siemens Digital Industries Software recommends that you use **initialstate** instead to avoid confusion.

`collapsed="true"` is the same as `initialstate="collapsed"`
`collapsed="false"` is the same as `initialstate="expanded"`

defaultdisplay (required)

Specifies the default format to use when displaying the set of objects. The value must be any valid **display element**.

source (required)

Specifies the comma-delimited set of run-time properties or relations that return the desired set of objects. The format for the attribute value is *relation.type*, where *relation* is the name of a relation, run-time, or reference property or a GRM relation, and *type* represents the type of objects to be included. You can also **use dynamic compound properties to specify a source**. Multiple sources can be specified as a comma-separated list.

Subtypes of the specified object type are recognized, so specifying the **ItemRevision** type will also return all **DocumentRevision** objects, **Design Revision** objects, and so on. When using a relation however, you must *explicitly* specify the relation. Subtypes of the specified relation are not considered.

The source definition also determines which properties are available to the display types. If you specify a parent object, but the property you are interested in is defined on one of the child objects, then the property may not display properly. Add the child object as a source as well.

© 2025 Siemens

showConfiguredRev (optional)

If any of the defined *sources* is an item type, then setting this attribute to **true** uses revision rules to determine which of the item's revisions to display instead of the item itself.

maxRowCount (optional)

The maximum number of rows displayed in the **tableDisplay** regardless of available vertical space.

sortby (optional)

Specifies the object property to sort the set of objects by prior to rendering.

sortDirection (optional)

Specifies the direction in which the set of objects on the **tableDisplay** should be sorted. Valid values are **ascending** or **descending**.
The default value is **ascending**.

enablePropEdit (optional)

Set this to **false** to disable the user's ability to automatically start editing on this table. While this is disabled, If the user wants to make changes to values on this table, they must start editing manually.

filterable (optional)

Allow this column to be filtered by the user using the UI. The type of filter shown to the user will depend on the data type of the column. Dates and numbers are automatically detected and will allow a range filter, but everything else is treated as a string. Set to **false** to prevent filtering.

The default is **true**.

You can set **filterable=false** on individual properties in the table display, or set it on the source to prevent filtering for all columns.

Example:

Filtering will be disabled on all columns, regardless of their individual settings.

```
<objectSetSection source= ... filterable="false">
    <tableDisplay>
        <property name= ... />
        <property name= ... />
        ...
    </tableDisplay>
    ...
</objectSetSection>
```

Example:

Only the second column will have filtering disabled.

```
<objectSetSection source= ... >
    <tableDisplay>
```


© 2025 Siemens

```
            <property name= ... />
            <property name= ... filterable="false"/>
            ...
        </tableDisplay>
        ...
    </objectSetSection>
```

hidden (optional)

Allow this column to be hidden by default. The benefit of hiding a column is that it allows the user to expose the property if they want, but otherwise the client will not load the information which saves time when loading a page. Set to **true** to hide the column.

The default is **false**.

Explicitly setting **hidden=false** serves no purpose.

Example:

The second property will not be displayed on the table, but a user could arrange their table to show it.

```
    <objectSetSection source= ... >
        <tableDisplay>
            <property name= ... />
            <property name= ... hidden="true" />
            ...
        </tableDisplay>
        ...
    </objectSetSection>
```

DISPLAY ELEMENTS

The following elements are defined within an object set section to determine the availability of the display types.

<tableDisplay>

<listDisplay>

<imageDisplay>

<thumbnailDisplay>

The **<tableDisplay>**'s columns can be defined using a list of **<property>** elements. If a user customizes the table by arranging or hiding columns, a column configuration will automatically be created for that user to store their changes, and then any property elements in the style sheet will be ignored. If that user ever resets their columns, the table will revert back to the object set section's property elements.

© 2025 Siemens

You can remove the user's ability to arrange the columns by setting the **enableArrange** attribute to **false**.

```
<tableDisplay enableArrange="false">
```

MODIFY AVAILABLE COMMANDS

Use the Command Builder to customize the list of commands that are displayed on an object set section. One common uiAnchor for object set sections is **aw_objectSetSection**.

EXAMPLE OF SETTING OBJECTSETSECTION SOURCE

In these examples, you are creating the source definition for an object set section on *folders*.

- You want to display any common object that is in a folder. The **Folder** object uses the **contents** property to maintain this list. The **WorkspaceObject** is the parent of all common user-facing objects.

  ```
  <objectSetSection source="contents.WorkspaceObject">
  ```

- You only want to display documents. You decide that this would include the **DocumentRevision**, **MSWordX**, and **PDF** object types. This time, even though the folder might contain other objects, they are ignored and the table only displays your documents. This will also include children of the **DocumentRevision**, like the **Specification Revision** object, for example.

  ```
  <objectSetSection source="contents.DocumentRevision,contents.PDF,contents.MSWordX">
  ```

FULL OBJECTSETSECTION EXAMPLE

The following example was taken from an OOTB style sheet.

```
<objectSetSection titleKey="tc_xrt_Files"
source="IMAN_specification.Dataset,IMAN_reference.Dataset,IMAN_manifestation.Dataset,IMAN
_Rendering.Dataset,TC_Attaches.Dataset,IMAN_UG_altrep.Dataset,IMAN_UG_scenario.Dataset,IM
AN_Simulation.Dataset,Mci0PMIXmlRel.Dataset" defaultdisplay="listDisplay"
sortby="object_string" sortdirection="ascending">
   <tableDisplay>
      <property name="object_string"/>
      ...
   </tableDisplay>
   <thumbnailDisplay/>
   <treeDisplay>
      <property name="object_string"/>
      ...
   </treeDisplay>
   <listDisplay/>
</objectSetSection>
```


© 2025 Siemens

parameter

Use this element is not used on its own, rather it is used to pass values to other elements. The other elements will specify when and where you can use the parameters element.

ATTRIBUTES

name

The name of the parameter being passed.

value

The value of the parameter being passed.

SUPPORTED STYLE SHEETS

This tag is not specifically supported on a style sheet, but rather can only be used in conjunction with another element.

EXAMPLE

Other elements will provide the specific parameter names and possible values available for use in each case.

Following is the generic syntax for **<parameter>**:

```
<paramter name="paramName" value="paramValue"/>
```


© 2025 Siemens

page

Presents a tab panel in a dialog box or view. If the **page** element is not defined in the XML file, a default page is created.

USAGE

This is typically at the top level of the style sheet, a direct child of the rendering or subrendering elements, although it may be the child of a content element.

ATTRIBUTES

These attributes define

titleKey	Specifies the key used for localization. If it is not defined or otherwise found, the string defined by the **title** attribute is used.
title (optional)	Specifies a static string of the title for this tab. This attribute is used when the **titleKey** is not found by localization.
visibleWhen	Uses a condition to determine if the element is displayed. You can specify many types of conditions with the visibleWhen attribute.

> Note:
>
> If there is only one page defined, Active Workspace ignores the **visibleWhen** condition and displays the page.

SUPPORTED STYLE SHEETS

This tag is supported on all types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<page>** element:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<rendering xmlns:xsi=...
           xsi:noNamespaceSchemaLocation=...>
```


© 2025 Siemens

```
   <header>
      ...
   </header>
   <page titleKey=...>
   </page>
   <page titleKey=...>
   </page>
   <inject type=.../>
</rendering>
```

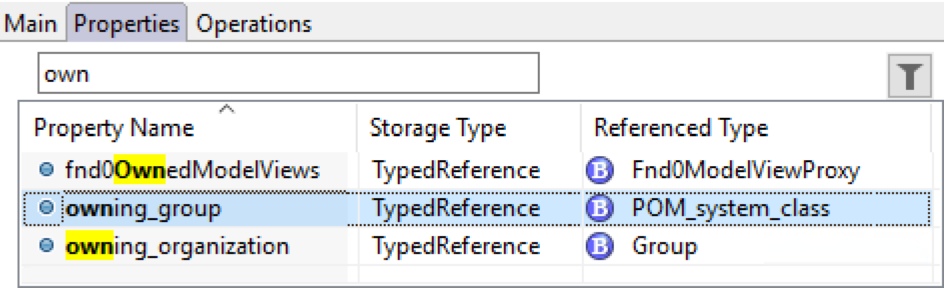

© 2025 Siemens

property

Use this element to display a property name and value from the selected object. You must include at least one property in the XML definition, otherwise, the system displays an empty panel. Note the following:

- You cannot add the same property multiple times in the same style sheet.
- Using style sheets to edit reference properties and reference property arrays which utilize non-workspace objects (not a child of **WorkspaceObject** in the BMIDE) is not supported.

  For example, **owning_group** is a **TypedReference** property whose **Referenced Type** is **POM_system_class**, which is not a workspace object type.



ATTRIBUTES

border

Determines whether the border is displayed. Valid values are **true** and **false**. This works only with the titled style.

modifiable

Set this to *false* to prohibit the user from editing a property that is normally modifiable. For example, if you want users to see but not change the object's description on this sheet, then use:

```
<property name="object_desc" modifiable="false"/>
```

isAutoAssign

Default: **true**

Set this to *false* on a property for which you do not want the client to ask the server for an auto assigned value until the user selects the **Assign** button. There is no need to use this attribute for auto generators to work

name

Specify the database name of a property on the object, not the display name. This is a required attribute.

© 2025 Siemens

```
<property name="object_name"/>
```

You can also use dynamic compound properties to specify a property from a related object.

```
<property name="GRM(IMAN_specification,DocumentRevision)
                        .item_revision_id"/>
```

Note:

When using this attribute on a **create** style sheet, there is additional functionality. You can specify a property from another object that is related to the original object by the **revision**, **IMAN_master_form**, or **IMAN_master_form_rev** relations. To do this, specify the relation trail followed by the name of the property on the destination related object, separated by a colon.

For example, if a **create** style sheet is registered for an item,

- to display the revision ID, you would use

  ```
  name=revision:item_revision_id
  ```

- to display the **project_id** property from the item's master form.

  ```
  name=IMAN_master_form:project_id
  ```

- to display the **serial_number** property from the item revision master form, you need to traverse from the item to the revision, and then to the revision's master form.

  ```
  name=revision:IMAN_master_form_rev:serial_number
  ```

renderingHint

Specify the component used to render this property. This is an optional attribute. If not defined, the default renderer is used based on the property type.

renderingStyle

Define the rendering style used in the rendering component. There are three styles: headed, headless, and titled.

- **Headed**

  This is the default rendering style. The property name is displayed on the left followed by the property value renderer.

- **Headless**

  This style renders only the property value without displaying the property name in front of it.

© 2025 Siemens

- **Titled**

  The property name is displayed on the top of the property value renderer.

THE <PARAMETER> ELEMENT

searchFilter	Use this parameter to configure a search prefilter on the list of objects presented to the user when they add a row to the table.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<property>** element:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<rendering xmlns:xsi=...
           xsi:noNamespaceSchemaLocation=...>
   <header>
      ...
   </header>
   <page titleKey=...>
      <property name="object_desc"/>
      <property name="effectivity_text" renderingHint="label"/>
      <break/>
      <property name="owning_user" renderingHint="objectlink" modifiable="false"/>
   </page>
   <page titleKey=...>
   </page>
   <inject type=.../>
</rendering>
```


© 2025 Siemens

rendering

This is the root element of a style sheet. All XRT elements *must* be contained within this root element. This line must not be modified.

ATTRIBUTES

These attributes, if present, define the schema namespace attributes for the style sheet.

xmlns:xsi

Specifies the default **xsi** prefix.

xsi:noNamespaceSchemaLocation

Active Workspace style sheets do not have a specific schema namespace.

SUPPORTED STYLE SHEETS

This tag is required on all types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **rendering** element:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<rendering xmlns:xsi=...
           xsi:noNamespaceSchemaLocation=...>
   <header>
      ...
   </header>
   <page titleKey=...>
   </page>
   <page titleKey=...>
   </page>
   <inject type=.../>
</rendering>
```


© 2025 Siemens

section

The section element is a container element that can help organize your style sheet content.

In the XRT hierarchy, sections and columns must not be siblings. Typically, sections are placed within columns to further divide up the properties into collapsible groupings.

ATTRIBUTES

titlekey (optional)	Specifies the key used for localization. If it is not defined or otherwise found, the string defined by the **title** attribute is used.
title (optional)	Specifies a static string of the title for this tab. This attribute is used when the **titleKey** is not found by localization.

initialstate	Specifies whether the view or section should be expanded or collapsed on initial rendering. Valid values are **expanded** or **collapsed**. The default value is **expanded**. This attribute is optional.
	
	You may also use the **collapsed** attribute, but Siemens Digital Industries Software recommends that you use **initialstate** instead to avoid confusion.
	
	`collapsed="true"` is the same as `initialstate="collapsed"`
	`collapsed="false"` is the same as `initialstate="expanded"`

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary

EXAMPLE

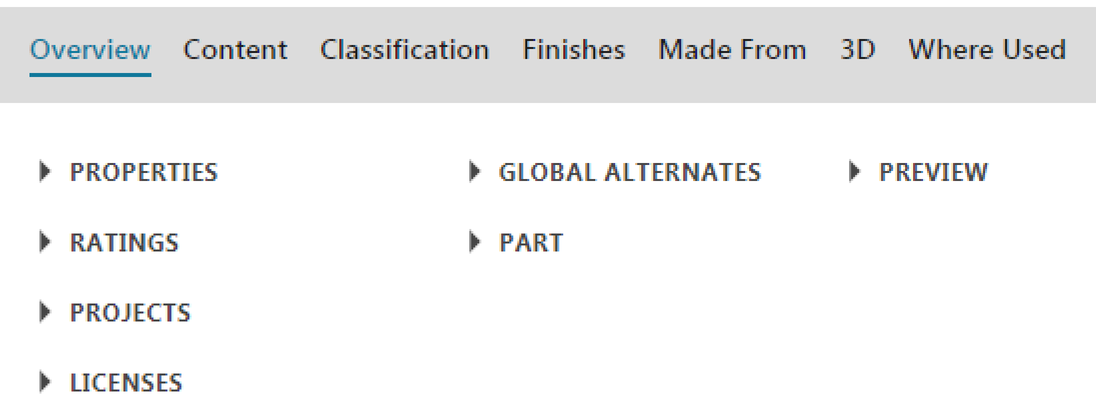
Following style sheet snippet shows the **<section>** element:

```
<page titleKey=...>
   <column>
      <section titleKey=...>
      ...
      </section>
      <section titleKey=...>
      ...
      </section>
   </column>
   <column>
      <section titleKey=...>
      ...
      </section>
```

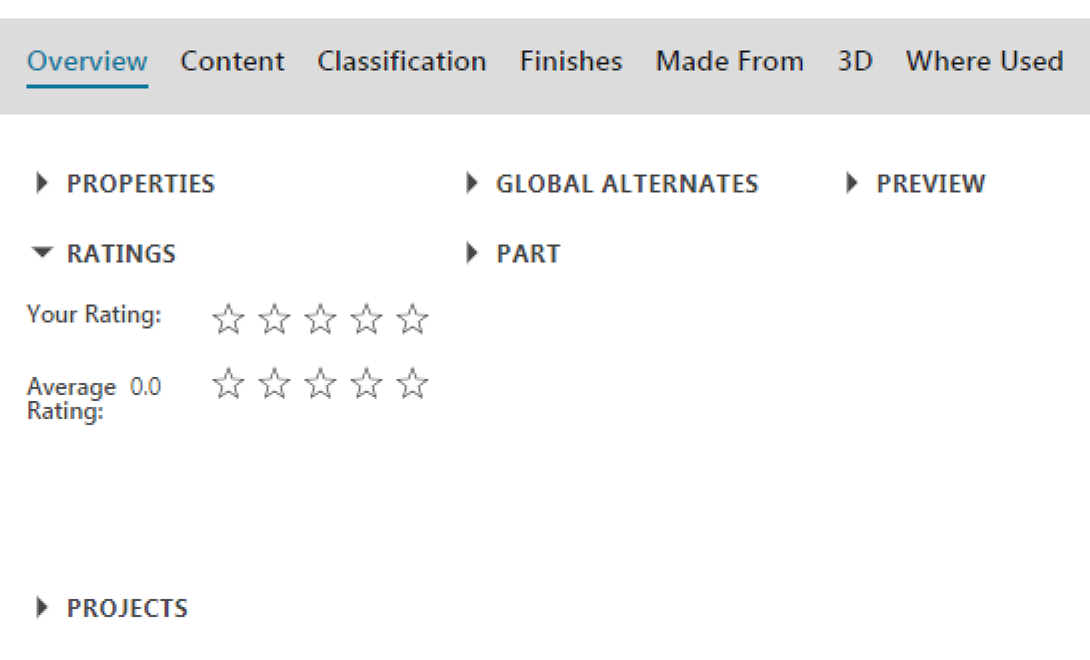

© 2025 Siemens

```
        <section titleKey=...>
        ...
        </section>
    </column>
</page>
```

Following are seven collapsed sections divided among three columns.



Following shows the ratings section expanded.



© 2025 Siemens

separator

You can insert a visible break between elements. This appears as a thin line.

USAGE

This element is typically used alongside the various property elements.

ATTRIBUTES

None.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<separator>** element:

```
<section titleKey="tc_xrt_properties">
   ...
   <property name="effectivity_text" renderingHint="label"/>
   <inject type="dataset" src="Cm1AuthoringChange"/>
   <separator/>
   <property name="owning_user" renderingHint="objectlink" modifiable="false"/>
   <property name="owning_group" renderingHint="objectlink" modifiable="false"/>
   ...
</section>
```

© 2025 Siemens

subRendering

This element must be a child of the rendering element on full style sheets. Use this element as the root element on a style sheet that is being *injected* into another.

ATTRIBUTES

None.

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary
Create
Information
Revise
Save As

EXAMPLE

Following style sheet snippet shows the **<subRendering** element:

```
<?xml version="1.0" encoding="UTF-8"?>
...
<subRendering>
   <page titleKey=...>
   </page>
   <page titleKey=...>
   </page>
   <inject type=.../>
</subRendering>
```

Then, in the main style sheet, use the **<inject>** element to insert this XML.

© 2025 Siemens

tableProperty

ATTRIBUTES

name (required)

The name of the table property you want to display.

source (optional)

Specifies the related object from which to retrieve the table property. You must **use dynamic compound properties to specify a source**.

sortBy (optional)

Specifies the object property to sort the set of objects by prior to rendering. If this is not specified, the rows will be displayed in the order they were created.

sortDirection (optional)

Specifies the direction in which the set of objects on the **tableProperty** should be sorted. Valid values are **ascending** or **descending**.
The default value is **ascending**.

enablePropEdit (optional)

Controls whether properties can be edited directly without using the **Start Edit** command. It does not control the general ability to edit this property.

Set this to **false** to disable the user's ability to automatically start editing on this table. While this is disabled, If the user wants to make changes to values on this table, they must start editing manually.

maxRowCount (optional)

The maximum number of rows displayed in the **tableProperty** regardless of available vertical space.

Get more information about how *table properties* **are not tables**.

THE <PROPERTY> ELEMENT

Use the **<property>** element inside the **<tableProperty>** element to choose which of the columns from the table you wish to display, and in which order.

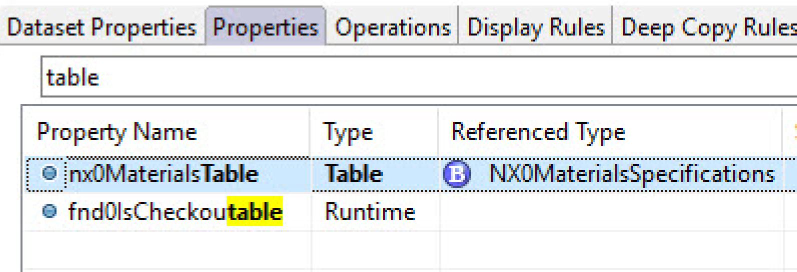
```
<tableProperty name="..">
   <property name="..."/>
   <property name="..."/>
   ...
</tableProperty>
```

The property names are defined on the **Fnd0TableRow** object associated with the **<tableProperty>**.

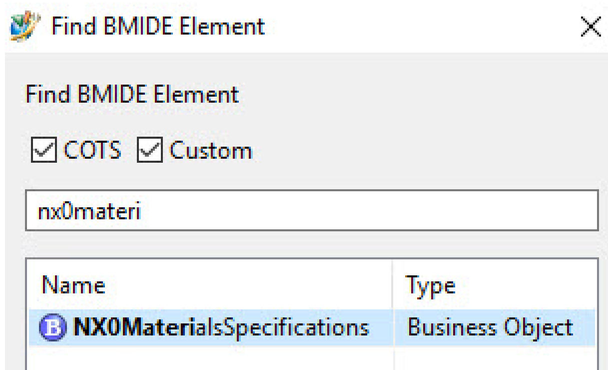
© 2025 Siemens

In this example, you want to display the **nx0MaterialsTable** from the **UGMaster** object.

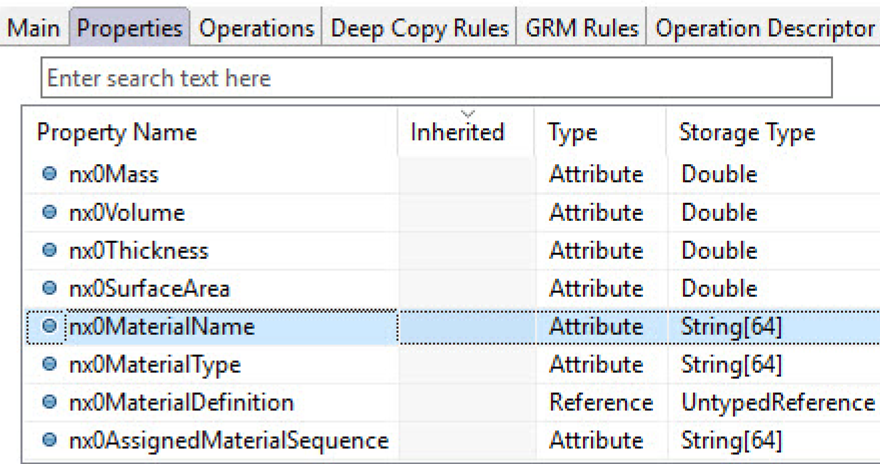
1. Using the BMIDE, locate the **nx0MaterialsTable** property on the **UGMaster** object.



2. Find the referenced type.



3. Choose from the available properties.



© 2025 Siemens

SUPPORTED STYLE SHEETS

This tag is supported on the following types of style sheets:

Summary

EXAMPLES

Example:

This OOTB example shows a table property on the selected object.

```
<tableProperty name="cae0KPITable">
  <property name="cae0KPITableName"/>
  <property name="cae0KPITableValue"/>
...
</tableProperty>
```

Example:

This OOTB example uses DCP to show a table property on a related object.

```
<tableProperty source="GRM(IMAN_specification,UGMASTER)"
               name="nx0MaterialsTable">
  <property name="nx0MaterialAssignmentType"/>
  <property name="nx0MaterialName"/>
...
</tableProperty>
```


© 2025 Siemens

The visibleWhen attribute

THE VISIBLEWHEN ATTRIBUTE

Use the **visibleWhen** attribute to specify when the element is displayed based on a condition. The value checked can be **null** or a string, and you can compare that value to the value of a:

- **Property on the selected object**

```
visibleWhen="property-name==xxx"
visibleWhen="property-name==xxx"
```

- **Property on a related object**

  Use dynamic compound properties.

- **Teamcenter preference**

```
visibleWhen="{pref:preference-name} == xxx"
visibleWhen="{pref:preference-name} !== xxx"
```

- **BMIDE global constant**

```
visibleWhen="{const:global-constant-name} == xxx"
visibleWhen="{const:global-constant-name} !== xxx"
```

  xxx can be a string or **{prop:property-name}** or null (representing the given constant is not defined).

- **BMIDE type constant**

```
visibleWhen="{const:type-name:type-constant-name} == xxx"
visibleWhen="{const:type-name:type-constant-name} !== xxx"
```

  type-name must be either a BMIDE type name or equal to the static string **ContextType**, which represents the current business object type.

- **BMIDE property constant**

```
visibleWhen="{const:type-name:property-name:property-constant-name} == xxx"
visibleWhen="{const:type-name:property-name:property-constant-name} == xxx
```

  type-name must be either a BMIDE type name or equal to the static string **ContextType**, which represents the current business object type.

- **Location or sub location**

```
visibleWhen="ActiveWorkspace:Location == xxx"
```


© 2025 Siemens

```
visibleWhen="ActiveWorkspace:SubLocation == xxx"
```

Find the last part of the URL for the location or sublocation you want to check.

- **XRT context**

```
visibleWhen='ActiveWorkspace:xrtContext=={"context_field":xxx}'
```

You can:

- Compare a single string to a string.
- Compare a single string to a list of strings. Array matching is not supported.

  For example: If myProp=[string1, string2, string3]

  - `visibleWhen "myProp==string1"` would work, because string1 is in the myProp array.
  - `visibleWhen "myProp==string1, string2, string3"` would *not* work.
- Compare the following non-string types as single values (non-array).

  int, short, double, float, char, logical
- Check the following property types for null and not null only.
  - typed and untyped reference
  - typed and untyped relation
  - external reference
  - date
- Use the logical **AND** operation to perform multiple checks. Use of **OR** is not supported.

EXAMPLE: CHECK PROPERTY

When the property is on the selected object:

```
visibleWhen="awb0ServiceAdapter == 4G"

visibleWhen="awb0UnderlyingObjectType == Aqc0CharElementRevision"

visibleWhen="awb0Parent!=null"
```


© 2025 Siemens

```
visibleWhen="ps_parents!=null"
```

When the property is on a related object:

```
visibleWhen="REF(att0CurrentValue,Att0MeasureValue).owning_user!=null"

visibleWhen="REF(root_task, EPMJob).root_target_attachments!=null"

visibleWhen="GRMS2PREL(IMAN_based_on,CAW0Defect).primary_object != NULL"
```

EXAMPLE: CHECK PREFERENCE

```
visibleWhen="{pref:CM_ReverseTreeAvailable}==true"

visibleWhen="{pref:AWB_ShowMarkup}==true"
```

EXAMPLE: CHECK BMIDE CONSTANT

```
visibleWhen="{const:ContextType:Att0EnableComplexValue} == true"

visibleWhen="{const:Att0ParameterPrj:ItemRevision} != null””
```

EXAMPLE: CHECK LOCATION OR SUBLOCATION

```
visibleWhen="ActiveWorkspace:Location!
=com.siemens.splm.clientfx.tcui.xrt.showObjectLocation"

visibleWhen="ActiveWorkspace:SubLocation !=
com.siemens.splm.client.occmgmt:OccurrenceManagementSubLocation"

visibleWhen="ActiveWorkspace:SubLocation == qualityFmeaSublocation"

visibleWhen="ActiveWorkspace:SubLocation != showObject"
```

EXAMPLE: CHECK XRT CONTEXT

```
visibleWhen='ActiveWorkspace:xrtContext=={"interfaceTable":"visible"}'

visibleWhen='ActiveWorkspace:xrtContext!={"showLimitedPlanInfo":true}'
```

EXAMPLE: USING AND

```
visibleWhen="{pref:Pma0IsReleaseAtLeast141}==true and
{pref:Pma0IsMaturitySupportedForDesign}==true"

titleKey="tc_xrt_AuditLogs"
```


© 2025 Siemens

```
visibleWhen="{pref:TC_audit_manager_version}==3 and
{pref:AWC_show_audit_logs}==true"

titleKey="tc_xrt_Overview" visibleWhen="structure_revisions==null
and ActiveWorkspace:SubLocation !=
com.siemens.splm.client.occmgmt:OccurrenceManagementSubLocation"

titleKey="tc_xrt_jt_viewer" visibleWhen="structure_revisions==null
and IMAN_Rendering!=null and ActiveWorkspace:SubLocation !=
com.siemens.splm.client.occmgmt:OccurrenceManagementSubLocation"

titleKey="tc_xrt_Overview" visibleWhen="structure_revisions==null
and ActiveWorkspace:SubLocation !=
com.siemens.splm.client.occmgmt:OccurrenceManagementSubLocation"

visibleWhen="REF(last_release_status,ReleaseStatus).object_name !=
Approved and REF(last_release_status,ReleaseStatus).object_name !=
Obsolete"
```


© 2025 Siemens

© 2025 Siemens

6. Architecture concepts

Learning Active Workspace architecture

What is Active Workspace?

The Active Workspace client framework presents Teamcenter and its applications, as well as other data sources, in an intuitive user interface, rather than the traditional interfaces and applications that targeted expert users.

How does it work?

Active Workspace is a web application that connects to Teamcenter and other data sources in order to present a consistent interface for the user. With Teamcenter, it communicates with the web tier of the four-tier architecture, relaying user interaction and presenting the results. In this capacity, it can replace other Teamcenter clients.

What configuration mechanisms are there?

The user interface (UI) includes few main mechanisms for easy extension.

- **Declarative locations**
- **Declarative commands**
- **XML rendering templates** (XRT, also known as style sheets)

What do I need to do before configuring?

The Active Workspace interface consists of many components. Learn the Active Workspace user interface terms for these components.

Learn how a declarative panel works with a **declarative panel walk-through**.

Visit the **UI Pattern Library** in the Active Workspace section of the Support Center.

Use the **Active Architect** to change the UI layout quickly and easily.

> Note:
>
> Many of the more involved platform modifications require the use of the Business Modeler IDE. Check the Teamcenter platform documentation to learn what is required.

© 2025 Siemens

How Active Workspace saves changes to properties

The Active Workspace client uses two different methods to save your users' property edits, explicit checkout and optimistic locking. Both of these methods are available for your users to use as they choose. You can restrict your users' access to either of these methods by modifying the visibility of the commands.

Explicit checkout

When your user selects the **Check Out** command, the object is explicitly checked out to them on the server, preventing anyone else from modifying the object. This lasts until the user explicitly checks the object back in using the **Check In** command. During this time, the user can make changes without concern about the object being modified while they are editing.

Optimistic locking

When your user selects the **Start Edit** command, the client allows them to edit the displayed properties, but does not check out the object for exclusive editing. This editing only occurs on the client. Then, after your user makes their changes and they select the **Save Edits** command, the client sends the changes to the server which checks the last saved date of the object (using the **lsd** property).

- If the object has not been modified since the user started editing, then the server quickly locks, modifies, and unlocks the object.
- However, if the object had been modified while your user was editing, then their changes are lost, and the Active Workspace client displays a message stating that the object has been modified and they should refresh it before editing.

Learn declarative contributions

Declarative action: navigate

You can use the **Navigate** action to take the user to a specific page.

The **UI Pattern Library** on Support Center maintains the up-to-date syntax and options.

Example: Zero-compile command example for Open

In this example, you examine the OOTB command handler for the **Open** command. This command handler references the **showObject** action, defined in the **actions** section.

© 2025 Siemens

```
"commandHandlers": {
    "Awp0ShowObjectCommandHandler": {
        "id": "Awp0ShowObject",
        "action": "showObject",
        "activeWhen": {
            "condition": "conditions.cmdOpenBaseActiveCondition"
        },
        "visibleWhen": {
            "condition": "conditions.cmdOpenBaseVisibleCondition"
        }
    }
}
```

The **showObject** action is defined as being a **Navigate** action type which will **navigateTo** the **com_siemens_splm_clientfx_tcui_xrt_showObject** page, and it will send along the UID of the selected object as a parameter.

```
"actions": {
    "showObject": {
        "actionType": "Navigate",
        "navigateTo": "com_siemens_splm_clientfx_tcui_xrt_showObject",
        "navigationParams": {
            "uid": "{{ctx.selected.uid}}"
        }
    }
}
```

The base **activeWhen** and **visibleWhen** condition expressions are shown for reference.

```
"conditions": {
    "cmdOpenBaseVisibleCondition": {
        "expression": "ctx.selected
        && ( 'com.siemens.splm.clientfx.tcui.xrt.showObjectLocation'
        !== ctx.locationContext['ActiveWorkspace:Location']
        || (ctx.locationContext.modelObject
        && ctx.selected.uid !== ctx.locationContext.modelObject.uid))
    "
    },
    "cmdOpenBaseActiveCondition": {
        "expression": "ctx.selected"
    }
}
```

Example: Override the Open command

In this example, when a project object is selected the OOTB **Open** command is overridden so that it takes the user to **ProjectContents** instead. This command handler references the **TcProjectShowDelegatedObject** action, defined in the **actions** section.

© 2025 Siemens

```
"commandHandlers": {
    "TcProjectShowObjectCommandHandler": {
        "id": "Awp0ShowObject",
        "action": "TcProjectShowDelegatedObject",
        "activeWhen": {
            "condition": "conditions.TcProjectOpenConditionActive"
        },
        "visibleWhen": {
            "condition": "conditions.TcProjectOpenConditionActive"
        }
    }
}
```

The **TcProjectShowDelegatedObject** action is defined as being a **Navigate** action type which will **navigateTo** the **ProjectContents** page, and it will send along the UID of the selected object as a parameter.

```
"actions": {
    "TcProjectShowDelegatedObject": {
        "actionType": "Navigate",
        "navigateTo": "ProjectContents",
        "navigationParams": {
            "uid": "{{ctx.selected.uid}}"
        }
    }
}
```

You must never broaden an existing command condition. Include the original condition to use as the base in order to ensure you are more specific. In this example, the original condition is ANDed with the new condition to check the selected object to see if it is a **TC_Project** type. This ensures that this new delegate command handler does not allow the **Open** command to be run outside of its normal design.

```
"conditions": {
    "TcProjectOpenConditionActive": {
        "expression": "conditions.cmdOpenBaseActiveCondition
        && (ctx.selected.type === 'TC_Project'
        || ctx.selected.modelType.typeHierarchyArray.indexOf('TC_Project') > -1)"
    }
}
```

Using the navigateIn attribute

You can use the **navigateIn** attribute to open the new page in either a new browser tab or a new browser window instead of the normal behavior of replacing the current browser contents.

newTab	Opens the new page in a new browser tab.

© 2025 Siemens

```
"openInNewTab": {
    "actionType": "Navigate",
    "navigateTo": "com_siemens_splm_clientfx_tcui_xrt_showObject",
    "navigationParams": {
        "locale": "{{ctx.userSession.props.fnd0locale.dbValues[0]}}",
        "uid": "{{ctx.selected.uid}}"
    },
    "navigateIn": "newTab"
}
```

newWindow Opens the new page in a new browser window. You can specify attributes for the new window using the **options** attribute.

```
"openInNewWindow": {
    "actionType": "Navigate",
    "navigateTo": "com_siemens_splm_clientfx_tcui_xrt_showObject",
    "navigationParams": {
        "locale": "{{ctx.userSession.props.fnd0locale.dbValues[0]}}",
        "uid": "{{ctx.selected.uid}}"
    },
    "navigateIn": "newWindow",
    "options": {
        "top": 10,
        "left": 10
    }
}
```

Declarative conditions

You can use conditions to provide logic in your view model.

Conditions:

- Evaluate to either true or false.
- Can refer to other conditions.
- Evaluate live data.
- Can leverage Boolean operations.

Expressions

Condition expressions can be expressed as a simple string,

```
"expression": "ctx.mselected && ctx.mselected.length > 1"
```

or as a JSON object.

© 2025 Siemens

```
"expression": {
    "$source": "ctx.mselected.length",
    "$query": {
        "$gt": 2
    }
}
```

Operators

The following operators are supported with expression definition objects.

$source	Indicates the reference on the data context node to be used as starting point for evaluation.
$query	Defines the query to be executed on the **$source**.
$all	Applicable when the **$source** is an array. Indicates that query result should be valid on all instances of the array elements.
$and	A logical AND of each query result.
$or	A logical OR of each query result.
$adapt	The resulting **$source** should be adapted before evaluating the query. Active Workspace sometimes uses intermediary runtime objects that represent other objects. An example of this is the **Awp0XRTObjectSetRow** object in **objectSet** tables. When you use **$adapt** the condition uses the target object instead of the intermediary object.
$in	The query should match with at least one of the value from the array.
$notin	The query should *not* match with any value from the array.
$eq	Equal.
$ne	Not equal. If you want to test for blanks, use: • `"$ne": []` to check if an *array* property does not contain an empty array. • `"$ne": ""` to check if a *single string* property does not contain an empty string. Use one of these with `$notnull` to ensure you find both blanks and nulls.
$notnull	Null value. Pair this with `$ne` to ensure you find both blanks and nulls.
$gt	Greater than.
$gte	Greater than or equal to.
$lt	Less than.
$lte	Less than or equal to.

© 2025 Siemens

Example:

Use of **$and**: Enable this command handler when the selected object type is **Folder** AND **object_name** is **Newstuff**

```
"expression": {
    "$source": "ctx.selected",
    "$query": {
        "$and": [{
                "$source": "modelType.typeHierarchyArray",
                "$query": {
                    "$in": ["Folder"]
                }
            },
            {
                "$source": "props.object_name.dbValue",
                "$query": {
                    "$eq": "Newstuff"
                }
            }
        ]
    }
}
```

Example:

Use of **$or**: Enable this command handler when the selected object type is **ItemRevision** OR **object_name** is **Newstuff**

```
"expression": {
    "$source": "ctx.selected",
    "$query": {
        "$or": [{
                "$source": "modelType.typeHierarchyArray",
                "$query": {
                    "$in": ["ItemRevision"]
                }
            },
            {
                "$source": "props.object_name.dbValue",
                "$query": {
                    "$eq": "Newstuff"
                }
            }
        ]
    }
}
```

Example:

Use of **$and** and **$or**: Enable this command handler when the selected object type is **ItemRevision** OR (**object_name** is **Newstuff** AND **object_type** is **Folder**)

© 2025 Siemens

```
"expression": {
    "$or": [{
            "$source": "ctx.selected",
            "$query": {
                "$source": "modelType.typeHierarchyArray",
                "$query": {
                    "$in": ["ItemRevision"]
                }
            }
        },
        {
            "$source": "ctx.selected",
            "$query": {
                "$and": [{
                        "$source": "modelType.typeHierarchyArray",
                        "$query": {
                            "$in": ["Folder"]
                        }
                    },
                    {
                        "$source": "props.object_name.dbValue",
                        "$query": {
                            "$eq": "Newstuff"
                        }
                    }
                ]
            }
        }
    ]
}
```

Example:

Use of **$adapt**: Enable this command handler when the target of the selected intermediary object is a **Cpd0DesignElement** object.

```
"expression": {
    "$source": {
        "$adapt": "ctx.selected"
    },
    "$query": {
        "$source": "modelType.typeHierarchyArray",
        "$query": {
            "$in": ["Cpd0DesignElement"]
        }
    }

}
```

Example:

Use of **$adapt** along with **$all**: Enable this command handler when all of the targets of the selected intermediary objects are **Cpd0DesignElement**

```
"expression": {
    "$source": {
        "$adapt": "ctx.mselected"
    },
    "$query": {
        "$all": {
            "$source": "modelType.typeHierarchyArray",
            "$query": {
                "$in": ["Cpd0DesignElement"]
            }
        }
    }

}
```

Example:

Use of **$lte**: Enable this command handler when the total workspace count is less than or equal to 1.

```
"expression": {
    "$source": "ctx",
    "$query": {
        "$source": "totalWorkspaceCount",
        "$query": {
            "$lte": 1
        }
    }
}
```

Example:

Reuse base condition: Enable this command handler when the base condition is true and type of adapted selected object is **Cpd0DesignElement**.

© 2025 Siemens

```
"expression": {
    "$and": [{
            "$source": "conditions.wiki_base",
            "$query": {
                "$eq": true
            }
        },
        {
            "$source": {
                "$adapt": "ctx.mselected"
            },
            "$query": {
                "$all": {
                    "$source": "modelType.typeHierarchyArray",
                    "$query": {
                        "$in": ["Cpd0DesignElement"]
                    }
                }
            }
        }
    ]
}
```

Reuse does not re-evaluate the base condition. It simply uses the evaluation result of the base condition as available from the conditions parameter on the evaluation context.

Declarative panel walk-through

Following is an example of a declarative panel in action where you create a saved search, but another one of that name already exists.

You perform a search for unassigned bolts owned by Ed.

1. You name the search *Ed unassigned bolts*, and activate the **Save** button.

   The **<aw-button>** in the view calls the **save** action when it is activated.

```
    <aw-panel-footer>
        <aw-button action="save" visible-when="conditions.isValidToSave">
            <aw-i18n2>i18n.Save</aw-i18n2>
        </aw-button>
    </aw-panel-footer>

</aw-command-panel>
```

   The **save** action in the view model calls a **TcSoaService** operation, **createFullTextSearch**.

© 2025 Siemens

```
"save":
{
    "actionType": "TcSoaService",
    "serviceName": "Internal-xxxx-FullTextSearch",
    "method": "createFullTextSearch",
    "inpudData":
    {
        "inputs":
        {
            "plmSearch": "{{function:getPinSearchValue}}",
            "savedSearchName": "{{data.searchName.dbValue}}",
            "searchString": "{{data.searchString.dbValue}}",
            "override": false,
            "recieveNotification": 0,
            "shareSavedSearch": 0,
            "searchFilterMap": "{{ctx.search.searchFilterMap}}"

        }
    }
},
```

2. The service fails because a saved search with that name already exists. The action uses a condition to handle the failure.

   The service fails with error code 141152, matching a condition. That condition raises the message **confirmOverwrite**.

```
"actionMessages":
{
    "failure":
    [
        {
            "condition": "errorCode.code==141152",
            "message": "confirmOverwrite"
        }
    ],

    "success":
    [
        {
            "condition": "data.pinToHome.dbValue",
            "message": "pinToHome"
        }
    ]
}
```

   confirmOverwrite presents the warning message along with two options, **Cancel** and **Overwrite**.

© 2025 Siemens

```
"messages":
{
    "confimOrverwrite":
    {
        "messageType": "WARNING",
        "messageText": "{{i18n.nameInUse}}",
        "messageTextParams":
        [
            "{{data.searchName.dbValue}}"
        ],
        "navigationOptions":
        [
            {
                "option": "Cancel",
                "text": "{{i18n.CancelText}}",
                "action": ""
            },
            {
                "option": "Overwrite",
                "text": "{{i18n.OverwriteText}}",
                "action": "overwrite"
            }
        ]
    }
}
```

3. You select **Overwrite**.

   The **Overwrite** option calls the **overwrite** action.

```
{
    "option": "Overwrite",
    "text": "{{i18n.OverwriteText}}",
    "action": "overwrite"
}
```

   The **overwrite** action calls the same service operation as before, but this time with **override** turned on.

© 2025 Siemens

```
"overwrite":
{
    "actionType": "TcSoaService",
    "serviceName": "Internal-xxxx-FullTextSearch",
    "method": "createFullTextSearch",
    "inpudData":
    {
        "inputs":
        {
            "plmSearch": "{{function:getPinSearchValue}}",
            "savedSearchName": "{{data.searchName.dbValue}}",
            "searchString": "{{data.searchString.dbValue}}",
            "override": true,
            "recieveNotification": 0,
            "shareSavedSearch": 0,
            "searchFilterMap": "{{ctx.search.searchFilterMap}}"

        }
    }
},
```

The successful SOA call returns a **success** event, which closes the panel.

```
"events":
{
    "success":
    [
        {
            "name": "layout.showToolInfo",
            "eventData":
            {
                "visible": false
            }
        }
    ]
},
```

The new saved search is created, overwriting the old one.

Note:

Siemens Digital Industries Software developers use the same building blocks to create panels as you. This example shows actual code snippets for an OOTB declarative panel, including a call to an *internal* service. Only use documented, published services in your code.

What are intermediary objects?

When performing certain functions, Active Workspace uses runtime intermediary objects to stand in place of the persistent business objects. These intermediary objects contain methods and properties needed for their function that don't need to be present on the persistent object they represent.

© 2025 Siemens

The two most common intermediary objects are:

Awp0XRTObjectSetRow

This object is used in objectSet tables. It works in conjunction with other **objectSet** runtime objects to represent their **awp0Target** object on a style sheet. They are part of the **aws2** feature template in the Business Modeler IDE.

Awb0Element

This object and its subtypes (**Awb0DesignElement**, **Awb0Connection**, and so on) are part of Active Content and used when Active Workspace displays objects in a structured manner, such as an assembly. They represent their **awb0UnderlyingObject** in a bill of materials, product structure, and so on. They are part of the **activeworkspacebom** feature template in the Business Modeler IDE.

What is really selected?

Use the context object to verify what type of object is really being selected. Following are some examples:

- **Single object**

  When the user is viewing a single object,

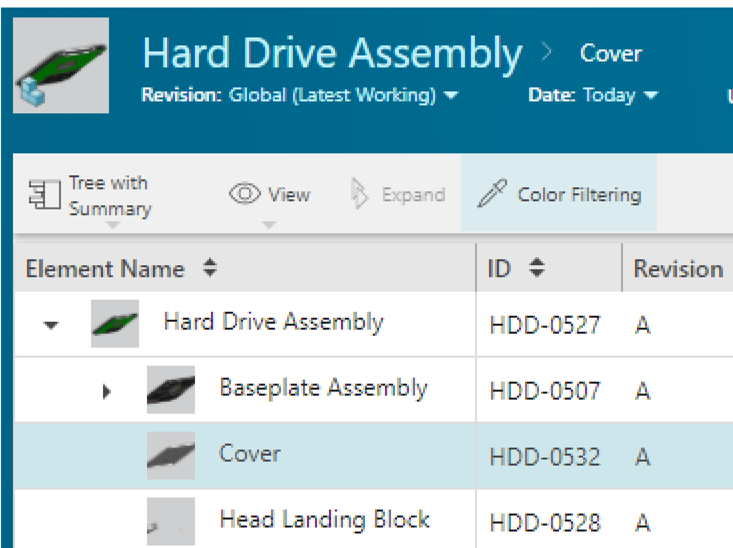
  then the object selected is the *persistent* business object shown, in this example the **ItemRevision** object.

© 2025 Siemens

```
▼selected: g
  ▶modelType: d {abstract
  ▶props: {owning_group:
   type: "ItemRevision"
```

- **Assembly**

  However, if you have **Active Content** installed and the user selects that same object in an assembly structure,



  then the object selected is the *runtime* business object **Awb0DesignElement**,

```
▼selected: g
   cellHeader1: "Cover"
  ▶modelType: d {abstract: fa
  ▶props: {awb0UnderlyingObje
   type: "Awb0DesignElement"
```

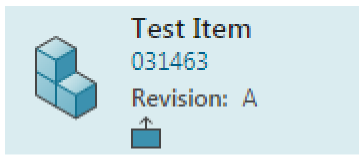
  which represents the underlying persistent business object by using its **awb0UnderlyingObject** relation property.

© 2025 Siemens

Using visual indicators to quickly recognize a property

What is a visual indicator?

A visual indicator is a small icon that appears when certain conditions are met; typically when a property exists, or contains a certain value. This is in addition to the regular icon that an object displays. An object can have several visual indicators registered to it. Active Workspace uses several visual indicators, such as **Checked Out**:



What do I need to create my own indicator?

You need to prepare the following:

- An **SVG** file that you will use for the indicator.
- Knowledge of the object types for which the indicator applies.
- Knowledge of the condition under which the indicator will appear.
- Knowledge of whether you need to pre-load any properties for your condition.

Use the configuration builder to register indicators.

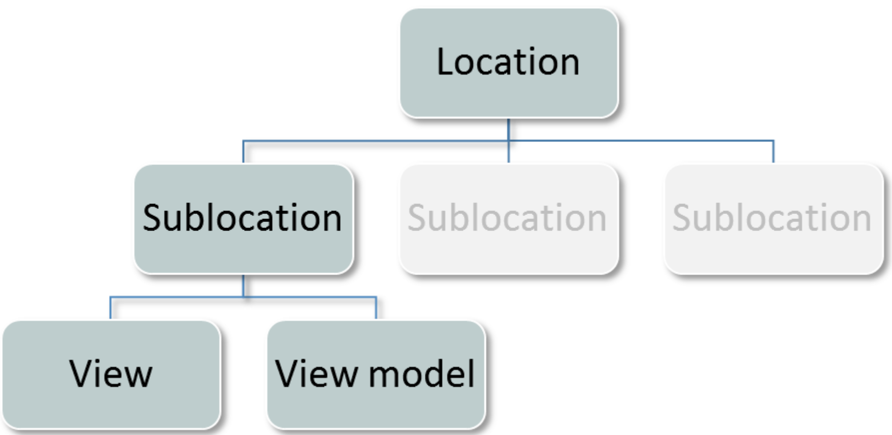
Using a sublocation to display a custom page

What is a sublocation?

A sublocation is a page in the UI that displays information and has a URL that points directly to it. Every sublocation must be assigned to a location, which is a grouping of related sublocations. Any number of sublocations may be assigned to a location. If only a single sublocation exists for a location, then the sublocation name is not shown.

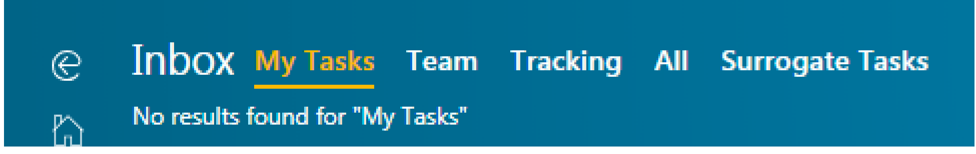
A sublocation is defined by *views* and *view models*.

© 2025 Siemens



Sublocation example

Examples of sublocations are: **My Tasks**, **Team**, **Tracking**, and so on. They are all part of the **Inbox** location.



The URL for the **My Tasks** sublocation is `http://host:port/#/myTasks`.

Custom sublocations

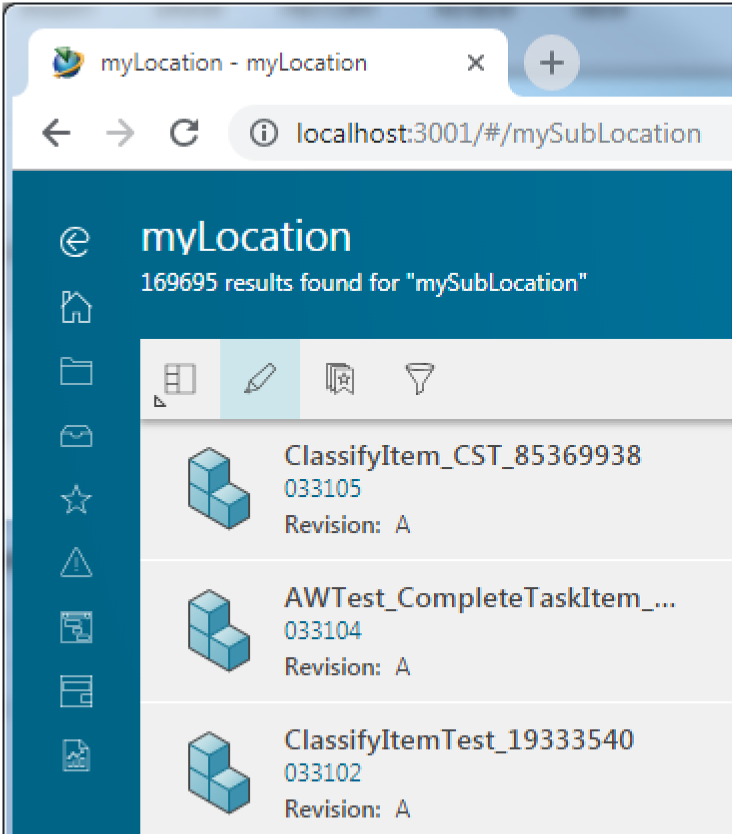
After creating your new sublocation, you can navigate directly to it by modifying your URL.

Example:

`http://host:port/#/`**mySubLocation**

This takes you to your new sublocation.

© 2025 Siemens



Declarative user interface

Declarative UI introduction

What is the declarative UI?

A capability provided by the Active Workspace framework that allows for a concise and codeless definition of UI view content, layout, routing, and behavior. Actions, messages, service calls and their inputs and outputs can be mapped and described using **HTML** and **JSON**. It provides an *abstracted* means of defining the client UI and its behaviors; the underlying implementation is hidden.

Why a declarative UI?

There are many reasons to use an abstracted, declarative user interface. It provides increased:

Performance	Reducing the number of lines of code decreases load times.
Efficiency	Enables faster iteration and reduces the need to develop and maintain code. A simple declarative view definition allows UX specialists to author the desired UI, while working with an application developer to wire up the view model based on the intent.

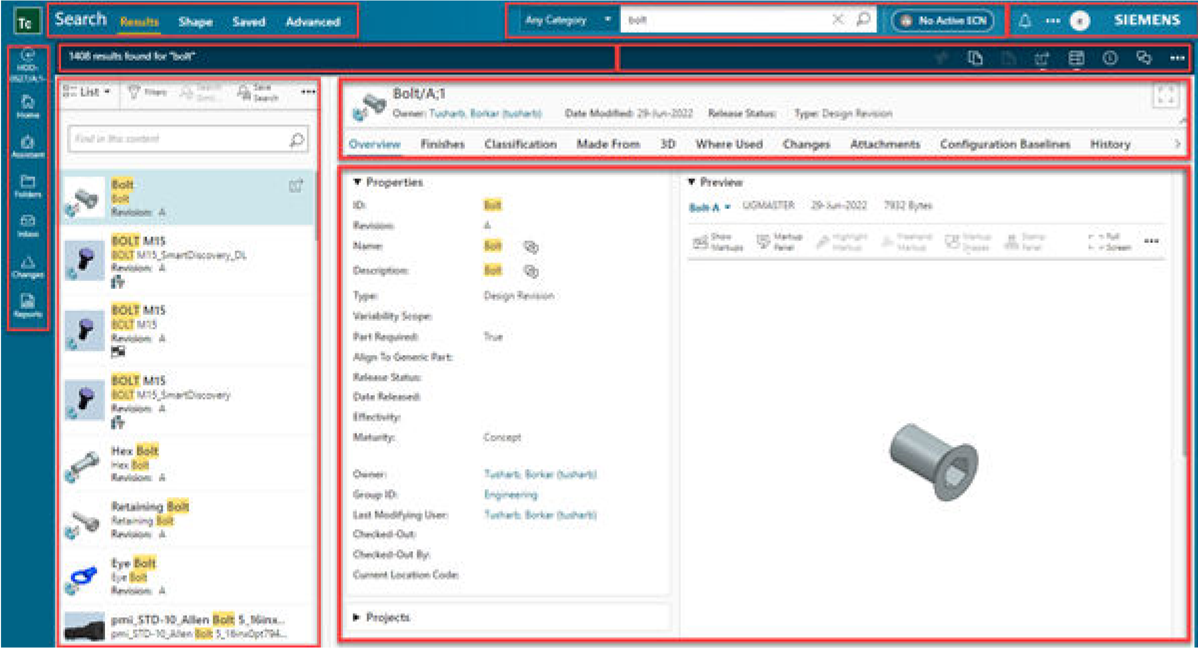
© 2025 Siemens

Sustainability Since the Active Workspace declarative elements are abstracted, the underlying web technology can evolve with minimal to no effect on existing layouts.

Consistency Each page and panel element is built from basic, modular UI elements instead of custom pieces.

How does it work?

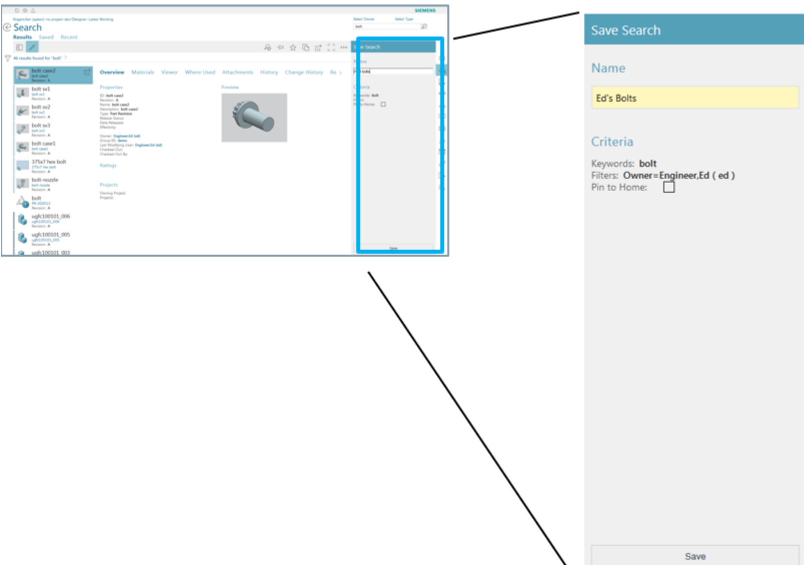
Declarative pages describe a single screen of the user experience and consist of several panels.



A declarative page is the entire presentation in the browser window and uses a URL.

- A declarative page is declaratively defined and shares the following characteristics:
  - Follows layout and content patterns.
  - Consists of several panels.
  - Creates a view that is described using UI elements.
  - Contains a view model that describes the page's data, i18n, and behaviors such as actions, conditions, messages, and routing.
- Declarative panels describe a region of the page.

© 2025 Siemens

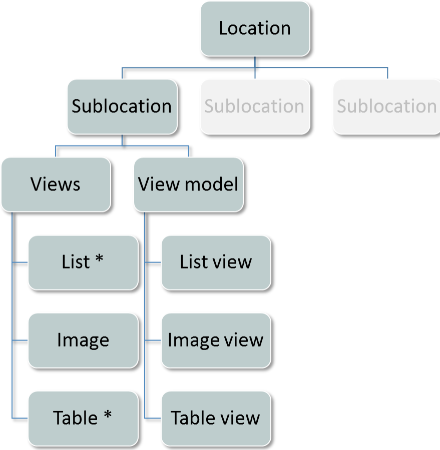


Panels are also declaratively defined and share the following characteristics.

- Include a view that described using UI elements.
- Contain a view model that describes the panel's data, i18n, and behaviors such as actions, conditions, messages, and routing.

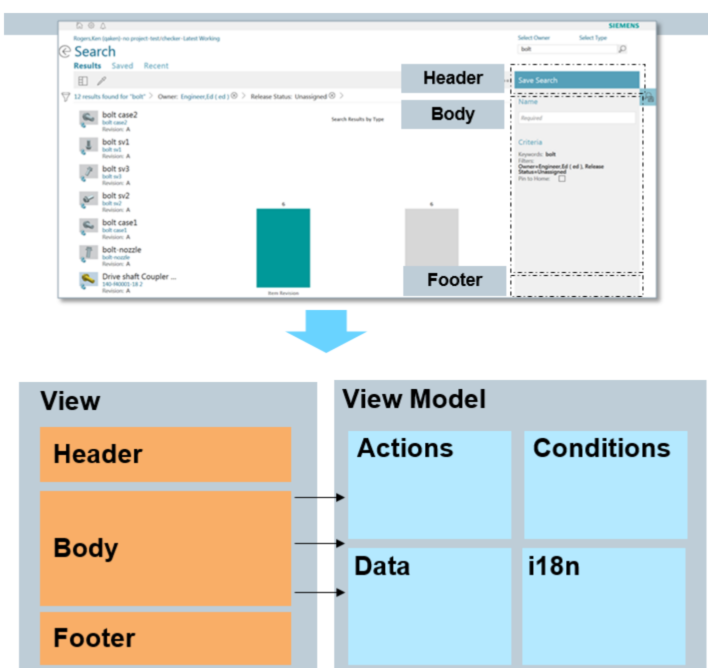
UI architecture

The Active Workspace UI consists mainly of sublocations grouped by locations. Each sublocation is defined by views, which rely on view models for their functionality.



The declarative definition of the UI has two main components, the view and the view model.

© 2025 Siemens



What files are involved?

The Active Workspace UI extends HTML with custom elements, which are part of **W3C**'s *Web Components*. Several custom elements work together to create content using the declarative UI:

Kit	A JSON text file that loads modules.
Module	A JSON text file that defines sublocations and commands.
View	A simple markup file (HTML) that controls a collection of related UI elements and their layout. These can be panels or pages, and may be displayed as the result of a command action or sublocation navigation. Views map to the view state which is defined in the view model.
View model	A JSON text file that is responsible for defining the view state, such as data, actions, i18n, and so on.
i18n	A JSON text file that provides localization capability for UI components.

How do I use it?

Use an Active Workspace developer environment command prompt to create a new native module, define your commands and custom elements, build and then publish it.

Visit the UI Pattern Library in the Digital Engineering Services documentation of Support Center.

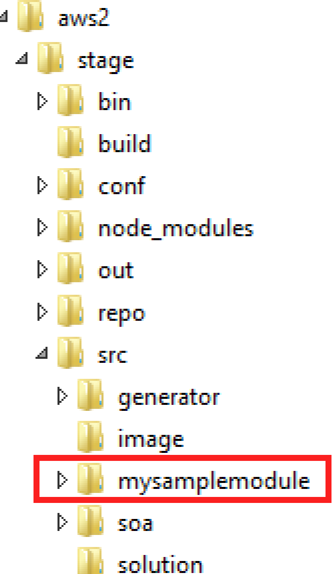
© 2025 Siemens

Declarative kit and module

The native module

Your custom declarative definitions reside in a module directory. You must create this directory in the **TC_ROOT\aws2\stage\src** directory. To help organize your configurations, you may maintain several modules, each in its own directory. You may create this directory manually, but using the **generateModule.cmd** script to create a module does this for you.

In this example, a **mysamplemodule** module directory was created.



kit.json

This singular file is provided OOTB, and is located in the **TC_ROOT\aws2\stage\src\solution** directory. It contains the definition for your installed configuration. You must register any modules you create in the **kit.json** file. You may edit this file manually, but using the **generateModule.cmd** script to create a module does this for you.

The **kit.json** file contains too much information to list here, but the opening section is the portion of interest.

© 2025 Siemens

kit.json

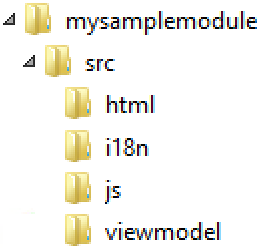
```
{
    "name": "tc-aw-solution",
    "modules": [
        "mysamplemodule",
        "tc-aw-solution"
    ],
    "version": "4.1.0",
    "bundles": {
        "js/base.bundle.js": {
            "deps": [
                "js/bootstrap"
            ],
            "include": [
                "js/aw-enter.directive",
                "js/iconService",
                "js/ngUtils",
                "js/NotyModule",
```

- **modules**

  This list of modules (defined by **modules.json**) are part of this kit.

module.json

A module may list other modules as dependents. In this file you define commands and their actions, conditions, and placement. Use the **generateModule** script to create this file initially. It creates the necessary directory structure and boilerplate files within the module directory.



The **module.json** file contains the following information:

module.json

```
{
    "name": "mysamplemodule",
    "description": "This is the mysamplemodule module",
    "type": [
        "native"
    ],
    "skipTest": true
}
```


© 2025 Siemens

- **name**

  This is the name of the module.

- **desc**

  This is the description of the module.

- **type**

  This is the type of module. The declarative UI uses the **native** type.

- **commandsViewModel**

  This is where you define your commands. You may instead use the **commandsViewModel.json** file to declare your command block. It must be a sibling of the **module.json** file.

  module.json

```
41    "commandsViewModel": {
42        "commands": {...
47        },
48        "commandHandlers": {...
59        },
60        "commandPlacements": {...
66        },
67        "actions": {...
76        },
```

- **states**

  This is where you define your locations and sublocations. These are ui-router states.

© 2025 Siemens

```
{ } module.json
 8⊖     "states": {
 9⊖         "MySubLocation": {…
11⊖             "data": {
12                  "priority": 0,
13⊕                 "label": {…
17                  "clientScopeURI": "",
18                  "nameToken": "MySubLocation",
19⊖                 "context": {…
26                  }
27              },
28⊕             "dependencies": […
31⊖             "params": {…
33              },
34              "parent": "MyLocation",
35              "reloadOnSearch": false,
36              "templateUrl": "/html/aw.native.sublocation.html",
37              "type": "subLocation",
38⊕             "url": "/MySubLocation"…
40      },
```

You can use the **visibleWhen statement** within your sublocation state definition to control its visibility based upon a condition.

- **dependencies**

  This is a list of other module names that are required.

Declarative control of sublocation visibility

The `visibleWhen` attribute used within a sublocation state definition can be used to control its visibility.

> Note:
>
> These snippets are provided for reference only, and are not designed to be production-ready code.

expression

The expression can be used to check the following:

ctx

The context object can be checked.

```
"expression":"ctx.selected.type=='aType'"
```

preferences

© 2025 Siemens

A preference can be checked.

```
"expression":"preferences.aMultiValuePreference.values.values[0]=='aVa
lue'"
```

parentState.data

The parent page's **data** section can be checked.

```
"expression":"data.aProp =='aValue'"
```

parentState.params

The parent page's **params** section can be checked.

```
"expression":"params.aParam =='aValue'"
```

A JavaScript function

The function should be asynchronous in nature and the *.js* file name should be provided in **deps**.

```
visibleWhen: {
     expression:"isSubPageEnabled(ctx.loadedObject)",
     deps:["js/subPageProvider"]
}
```

An Example

The following snippet from the OOTB state definition is an example of how the **Advanced Search** sublocation is controlled by the **AW_Advanced_Search_Visibility** preference.

© 2025 Siemens

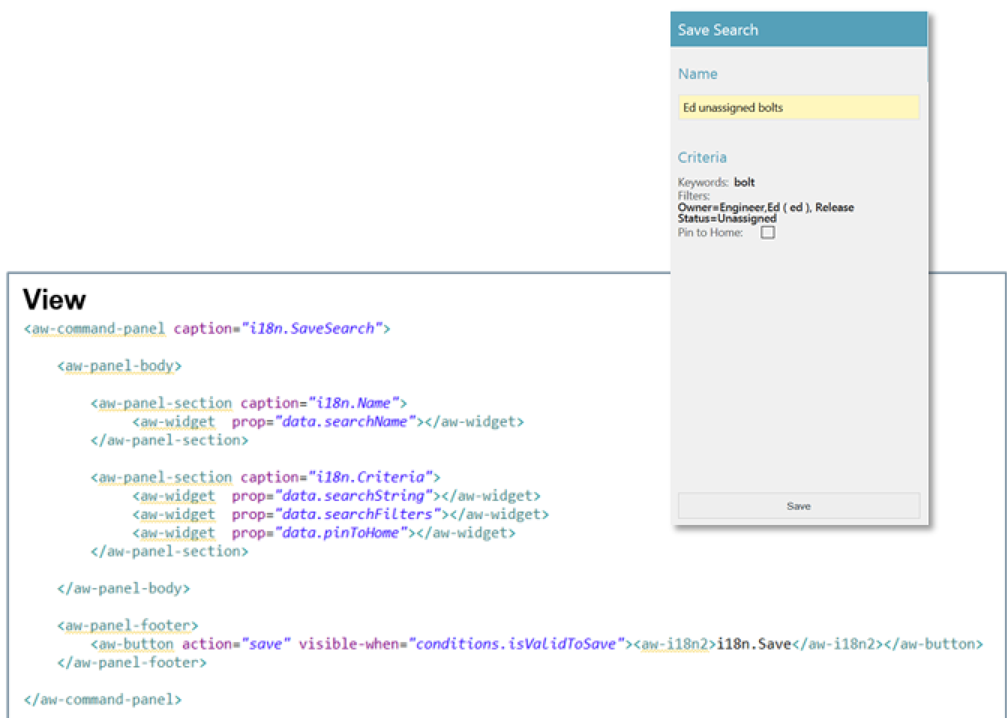
```
"teamcenter_search_advancedSearch": {
   "controller": "DefaultSubLocationCtrl",
   "data": {
      "priority": 400,
      "label": {
         "source": "/i18n/SearchMessages",
         "key": "advancedText"
      },
      "clientScopeURI": "Awp0AdvancedSearch",
      "nameToken": "teamcenter.search.advancedSearch"
   },
   "dependencies": ["js/aw.default.sublocation.controller"],
   "parent": "com_siemens_splm_client_search_SearchLocation",
   "presenter": "AdvancedSearchSubLocationPresenter",
   "type": "subLocation",
   "url": "/teamcenter.search.advancedSearch",
   "params": {
      "cmdId": null
   },
   "visibleWhen": {
      "expression": "preferences.AW_Advanced_Search_Visibility.values.values[0]==1
                  || preferences.AW_Advanced_Search_Visibility.values.values[0]=='true'"
   }
}
```

Declarative view

What is a view?

The view is an HTML markup file consisting of UI elements. The view is also responsible for:

- Defining the view hierarchy including sections and content.
- Mapping to data, actions, conditions, and i18n that are defined in the view model.
- Controlling the visibility of UI elements using **visibleWhen** clauses.



*View.html

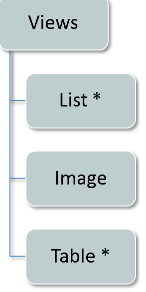
These files are located in the **module\src\html** directory.

Their file naming pattern is to prefix the page or panel name onto **View.html**. For example, create the **cmdQuickLinksView.html** file to represent the view for the **cmdQuickLinks** panel.

Place your UI elements in this file to define the view.

View types

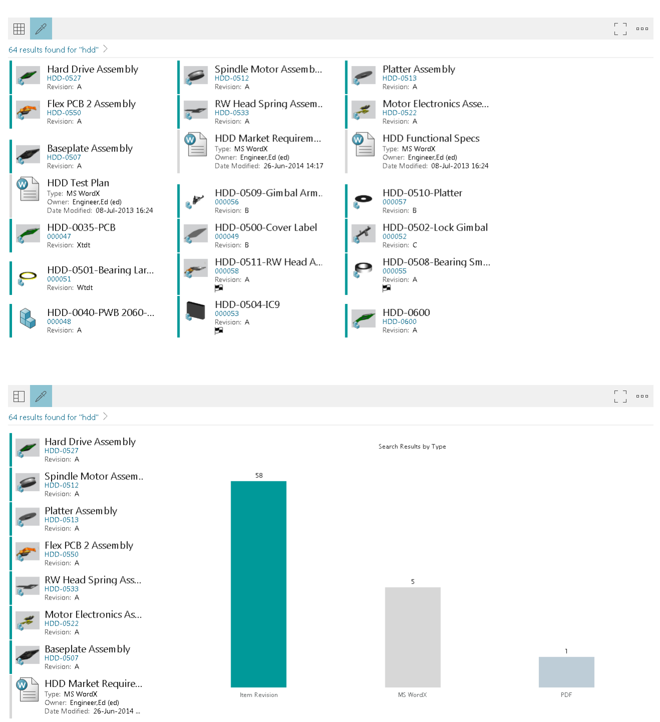
There are several types of predefined views available.



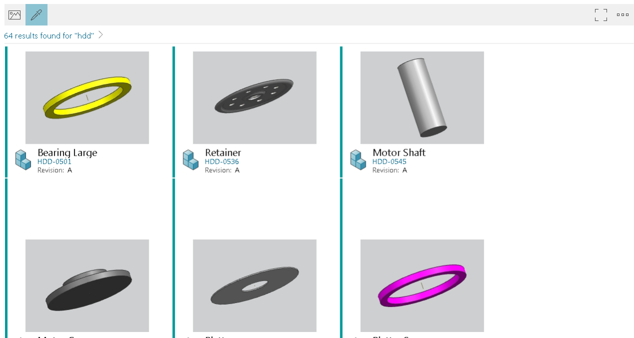
List

* With or without summary.

© 2025 Siemens

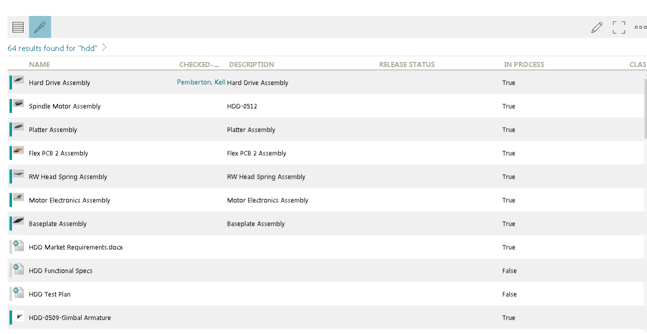


Image



Table

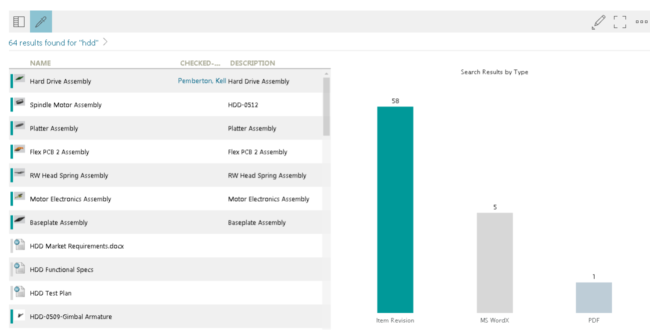
* With or without summary.



64 results found for "hdd"

NAME	CHECKED-...	DESCRIPTION	RELEASE STATUS	IN PROCESS	CLASSI
Hard Drive Assembly	Pemberton, Kell	Hard Drive Assembly		True	
Spindle Motor Assembly		HDD-0512		True	
Platter Assembly		Platter Assembly		True	
Flex PCB 2 Assembly		Flex PCB 2 Assembly		True	
RW Head Spring Assembly		RW Head Spring Assembly		True	
Motor Electronics Assembly		Motor Electronics Assembly		True	
Baseplate Assembly		Baseplate Assembly		True	
HDD Market Requirements.docx				True	
HDD Functional Specs				False	
HDD Test Plan				False	
HDD-0509-Gimbal Armature				True	

© 2025 Siemens



Declarative view model

What is a view model?

The view model is a JSON file. It is responsible for view state such as:

- Imports
- Actions
- Data
- Conditions
- i18n

*ViewModel.json

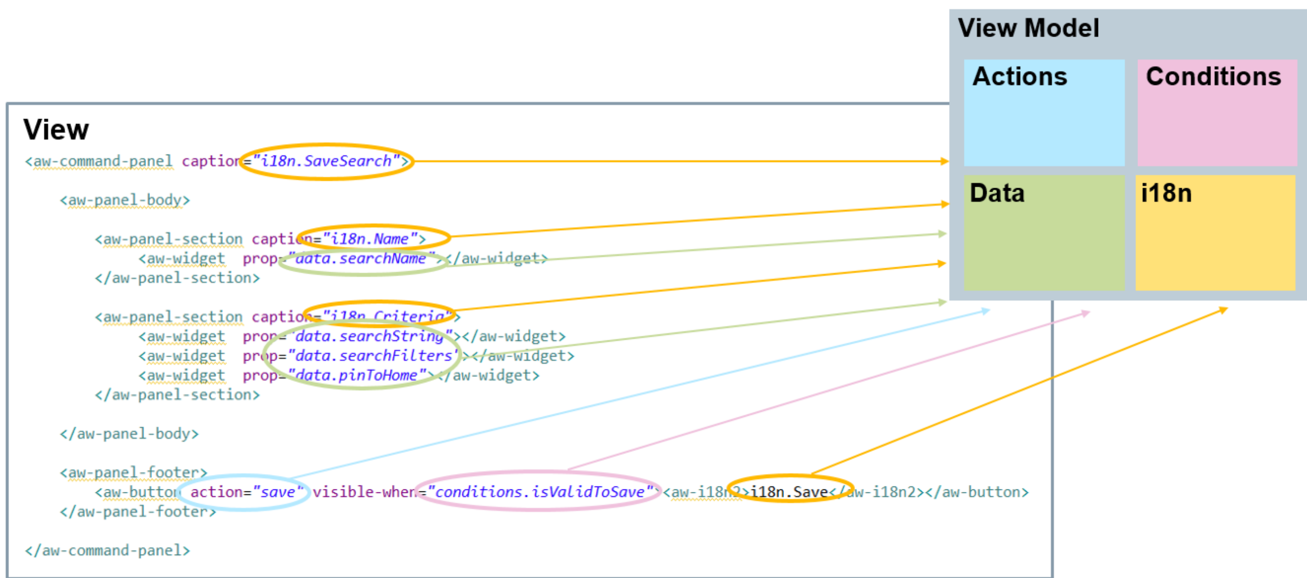
These files are located in the **module\src\viewmodel** directory.

Their file naming pattern is to prefix the page or panel name onto **ViewModel.json**. For example, create the **cmdQuickLinksViewModel.json** file to represent the view model for the **cmdQuickLinks** view.

Place your UI elements in this file to define the view, and import any necessary directives.

© 2025 Siemens

Mapping the view to the view model



Imports

Imports are used to indicate the custom elements we want to use in our view and view model:

<aw-command-panel>

<aw-section>

visible-when

<aw-button>

<aw-widget>

<aw-panel-body>

<aw-i18n2>

<aw-panel-footer>

View Model

```
{
    "imports": [ "js/aw-command-panel.directive",
                 "js/aw-panel-body.directive",
                 "js/aw-panel-section.directive",
                 "js/aw-panel-footer.directive",
                 "js/aw-button.directive",
                 "js/aw-widget.directive",
                 "js/aw-i18n2.directive",
                 "js/visible-when.directive"],
```

Actions

The **actions** JSON object consists of the following components.

actionType

Supported options: TcSoaService, JSFunction, and RESTService

inputData

JSON data for the action input

outputData

© 2025 Siemens

JSON data for the action output

events

Triggered in response to the action

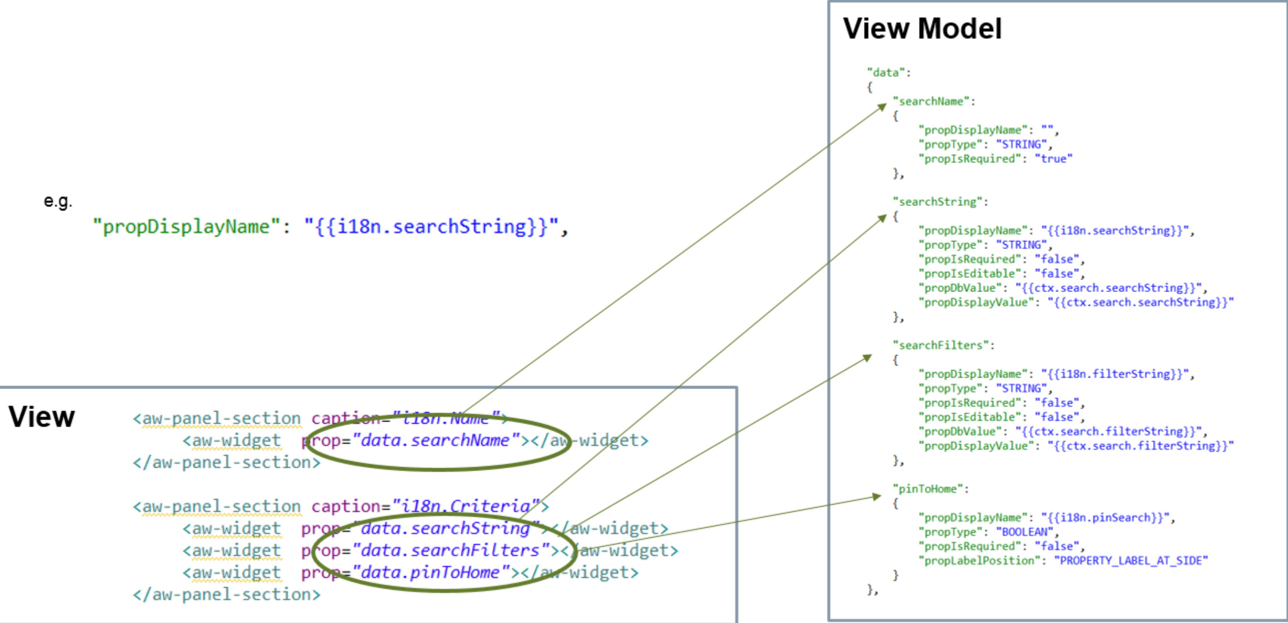
actionMessages

User messages and condition support

Data

The view can refer to data and view state in the view model data section.

Using {{ }} allows declarative binding to the live data.

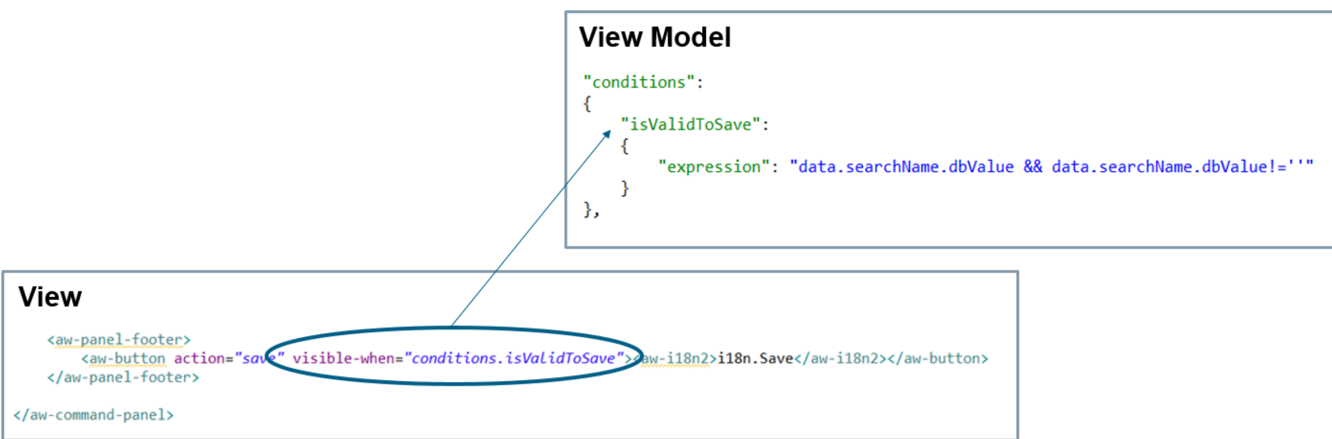


Conditions

Conditions evaluate to true or false, may use Boolean expressions (&&, ||, ==, !=), and may be used as follows:

- Use in **visibleWhen** clauses.
- Use for event handling.
- Refer to the data model state.
- Update the view model state.

© 2025 Siemens



i18n

User-facing text is localized. The view refers to localizations defined in the **i18n** section of the view model.

i18n text is provided as a JSON bundle.

View Model

```
"i18n":
{
    "Name":[ "SearchMessages","BaseMessage"],
    "SaveSearch": ["SearchMessages"],
    "Criteria": ["SearchMessages"],
    "Save":["SearchMessages"],
    "nameInUse": ["SearchMessages"],
    "CancelText": ["SearchMessages"],
    "OverwriteText": ["SearchMessages"],
    "pinToHome": ["SearchMessages"],
    "searchString": ["SearchMessages"],
    "filterString": ["SearchMessages"],
    "pinSearch": ["SearchMessages"]
}
```

View

```
<aw-panel-section caption="i18n.Name">
     <aw-widget  prop="data.searchName"></aw-widget>
</aw-panel-section>

<aw-panel-section caption="i18n.Criteria">
     <aw-widget  prop="data.searchString"></aw-widget>
     <aw-widget  prop="data.searchFilters"></aw-widget>
     <aw-widget  prop="data.pinToHome"></aw-widget>
</aw-panel-section>
```

Declarative messages

Messages are localized, are launched by actions, and cover information, warning, and error notifications. Messages may also:

- Present the user with options.
- Trigger actions.
- Leverage view model data and conditions.

© 2025 Siemens

Messages.json

This file is located in the **module\src\i18n** directory.

The file naming pattern is to prefix the module name onto **Messages.json**. For example, create the **quickLinksMessages.json** file to represent the localized text strings used within the **quickLinks** module.

Mapping messages and i18n

In the **...ViewModel.json** files, strings are localized using the **i18n** object.

For example, the **displayName** here is bound to **i18n.checkBoxName**.

```
"data": {
  "check": {
    "displayName": "{{i18n.checkBoxName}}",
    "type": "BOOLEAN",
    "isRequired": "true",
    "vertical": true,
    "dbValue": false...
  "object": {...
},
"dataProviders": {...
"i18n": {
  "cmdMyCommandTitle": ["MyModuleMessages"],
  "header": ["MyModuleMessages"],
  "body": ["MyModuleMessages"],
  "footer": ["MyModuleMessages"],
  "save": ["MyModuleMessages"],
  "checkBoxName": ["MyModuleMessages"]
}
```

In turn, the **i18n** object refers to the **...Messages.json** file to retrieve the actual string for display.

In this case, all of the **i18n** entries refer to a single file, **MyModuleMessages**. The system locates **src\i18n\MyModuleMessages.json** to look up the localized strings. For this example, **Enable the OK button in footer**, is displayed.

```
{
    "cmdMyCommandTitle": "My Command",
    "header": "Header",
    "body": "Body",
    "footer": "Footer",
    "checkBoxName": "Enable the OK button in footer",
    "save": "Ok",
    "MyLocationHeaderTitle": "MyLocation",
    "MyLocationBrowserTitle": "MyLocation",
    "MyLocationBrowserSubTitle": "MyLocation",
    "MySubLocationTitle": "MySubLocation Title",
    "MySubLocationChartTitle": "MySubLocation Chart Title"
}
```


© 2025 Siemens

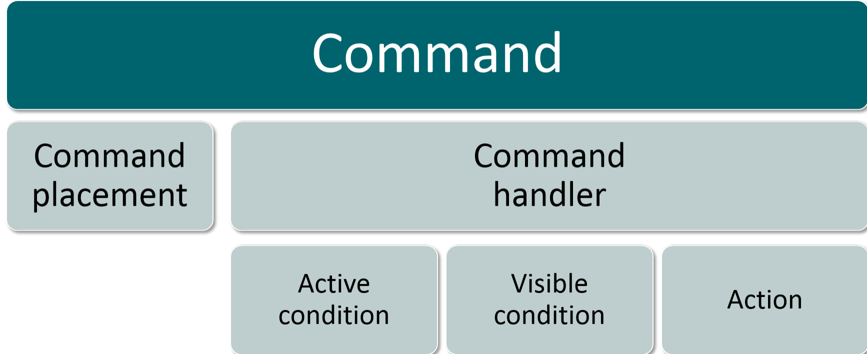
This extra layer of abstraction allows the **...Messages.json** file to be exchanged based upon locale without having to modify the **...ViewModel.json** files. You can maintain as many message files as you need.

Learn the declarative command architecture

Declarative command object - commands

Declarative command object hierarchy

A few basic objects define a declarative command.



commands JSON object

A command is the concept of a command. Its property name in the **commands** object is the *command Id*. Command contains the following properties:

- **iconId**: refers to the SVG icon stored in the image folder in the WAR location
- **isGroup**: true if a group command, false otherwise
- **title**: either an **i18n** key, or just a string

Below are some example **commands**:

© 2025 Siemens

```
"commands":
{
    "MyCheckout": {
        "iconId": "cmdWalk",
        "isGroup": false,
        "title": "{{i18n.MyCheckOutTitle}}"
    },
    "Awp0InboxFilters": {
        "iconId": "cmdfilter",
        "isGroup": false,
        "title": "Filters"
    },
    "Awp0AlertUser": {
        "iconId": "cmdAlerts",
        "isGroup": false,
        "title": "Alert"
    },
    "Awp0InContextReports": {
        "iconId": "cmdGenerateReport",
        "isGroup": false,
        "title": "Generate Report"
    }
},
```

Controlling command visibility

You can limit which commands your users see, based on various conditions. This helps reduce screen clutter and allows you to tailor your users' experience to your company's needs.

Control the visibility of commands in the Active Workspace interface using a combination of client-side conditions in the declarative command handlers, and server-side conditions in the Business Modeler IDE.

- Server-side conditions provide a base-level of visibility from which client-side conditions can further restrict.
- Client-side conditions are preferable, since they are much easier to modify and do not require a server call.
- Server-side conditions have a deeper data model access that might be required.

Some OOTB commands specify server-side conditions, many do not. Check the command list in the Business Modeler IDE to see if a command has a server-side condition listed. Remember load all appropriate templates. Even if there are no server-side conditions attached OOTB, you can still attach a condition if you have a need.

Client-only control

Siemens Digital Industries Software recommends using only declarative client-side conditions and expressions whenever possible. You use declarative conditions in the **activeWhen** and **visibileWhen**

© 2025 Siemens

clauses of your declarative command handler. Condition expressions are powerful and offer a lot of flexibility, including the capability to check a server-side command condition.

Example:

```
"cmdQuickAccessHandler": {
    "id": "cmdQuickAccess",
    "action": "quickAccessAction",
    "activeWhen": true,
    "visibleWhen": {
        "condition": "conditions.quickAccessVisibility"
    }
}
```

Client and server control

This allows you to do more complex decision-making, such as checking for project membership or the value of environment variables.

You can create server-based visibility for your custom command by registering it in the **Client UI Configuration** section of the Business Modeler IDE. Remember load all appropriate templates.

You must use the `ctx.visibleServerCommands` declarative expression on the client-side in order to implement server-based conditions.

Example:

You can specify a **visibleWhen** condition in the command handler,

```
"commandHandlers": {
    "c9CommandHandler": {
        "id": "c9Command",
        "action": "activatec9Command",
        "activeWhen": true,
        "visibleWhen": {
            "condition": "conditions.commandIsVisible"
        }
    }
}
```

and then use the following expression.

```
"conditions": {
    "commandIsVisible": {
        "expression": "ctx.visibleServerCommands.C9command"
    }
}
```


© 2025 Siemens

The Active Workspace client will ask the Teamcenter platform to evaluate the conditions attached to the Business Modeler IDE command definition named **C9command**.

Command : C9command

Details

Project: c9x

Name: C9command

Display Name: command

Description:

Selection Mode: Multiple

Tool Tip:

Icon:

User Input Required?

Condition: C9myCommandCond

COTS?

Template c9x

Where can I learn more about server conditions?

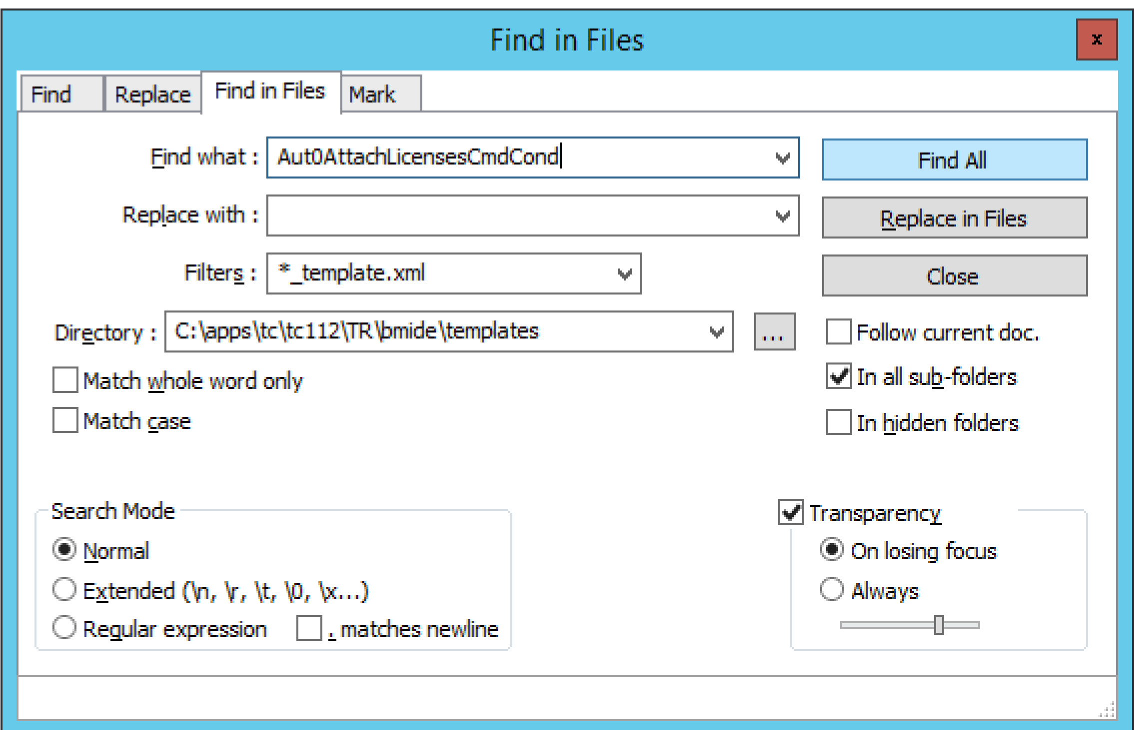
The Business Modeler IDE is where you can define these conditions. They are used for many purposes on the server side, including workflow creation, deep copy rules, list of values, and so on. They can check object property values, user session information, other conditions, and even things like preference values using the **Fnd0ConditionHelper** object. More complete information on server conditions are found in the Teamcenter administration documentation. If you intend to modify the visibility of a command that uses server conditions, be sure to examine the condition closely to learn how it works, and verify what other functionality might be using that condition.

How can I find which commands use a condition?

You must know which commands use a condition before you attempt to modify it. Use the Business Modeler IDE search files function. One way to find command condition attachments is to search all template files within your *TC_ROOT\bmide\templates* directory for the name of the command you are interested in. You must have all your Business Modeler IDE templates installed, but not necessarily used in the project, to get a complete result.

You can use any method you wish to search. In this example, **Aut0AttachLicensesCmdCond** is searched. You should search your own extension files as well.

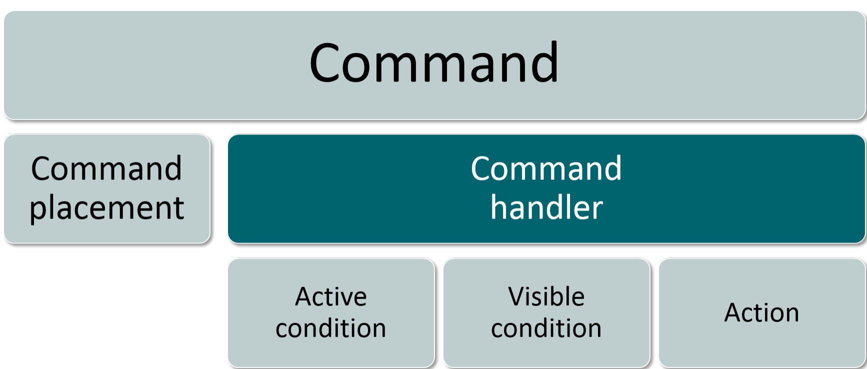
© 2025 Siemens



Declarative command object - commandHandlers

Declarative command object hierarchy

A few basic objects define a declarative command.



commandHandlers JSON object

A command can have many command handlers. However at any given time, there may be at most only one *active* **commandHandler** for a command. A declarative **activeWhen** condition that evaluates a Boolean expression controls whether a handler is currently active. If more than one handler for a given command evaluates to true, then the more *specific* condition, the condition with longer expression, is chosen to be active. The active command handler determines:

© 2025 Siemens

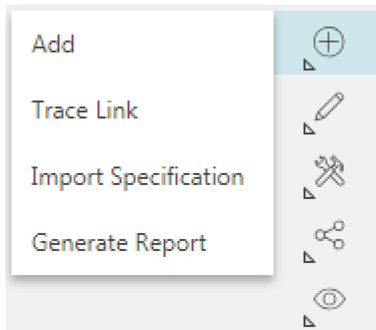
1. When a command is visible.
2. What a command will do.

A command handler has the following properties:

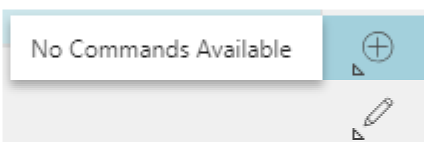
- **id**: The *command Id* for the associated command.
- **action**: The declarative action to be executed.
- **activeWhen**: Determines when this command handler is active for the associated command.
- **enableWhen**: Determines when the associated command is enabled. If a command is visible but not enabled, it will appear grayed out. Use this for commands you want to stay in their place, even when they're not active.
- **visibleWhen**: Determines when the associated command is visible. This condition is only evaluated if this command handler is active.

  Use **visibleWhen=true** on command groups to keep them visible at all times.

  Commands contained in command groups are evaluated when the group is opened.

  If there are no valid commands in a command group, the group displays the **No Commands Available** message when opened.

Below are some example **commandHandlers**:

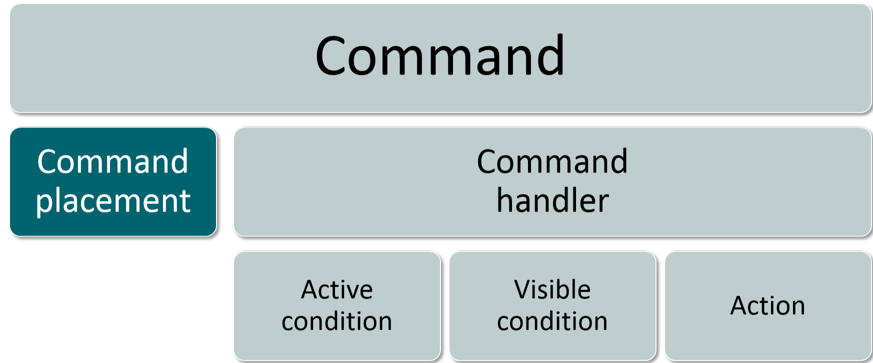
© 2025 Siemens

```
"commandHandlers":
{
    "myCheckoutCommandHandler": {
        "id": "MyCheckout",
        "action": "checkItOut",
        "activeWhen": true,
        "visibleWhen": true
    },
    "inboxFilterCommandHandler": {
        "id": "Awp0InboxFilters",
        "action": "activateFilterPanel",
        "activeWhen": true,
        "visibleWhen": true
    },
    "itemCopyCommandHandler": {
        "id": "Awp0AlertUser",
        "action": "alertItemRevision",
        "activeWhen": {
            "condition": "conditions.alertItemRevisionCommandActive"
        },
        "visibleWhen": {
            "condition": "conditions.alertItemRevisionCommandVisible"
        }
    },
    "itemCopyCommandHandlerQwerty": {
        "id": "Awp0AlertUser",
        "action": "alertItemRevisionSpecial",
        "activeWhen": {
            "condition": "conditions.alertItemRevisionSpecialCommandActive"
        },
        "visibleWhen": {
            "condition": "conditions.alertItemRevisionCommandVisible"
        }
    },
```

Declarative command object - commandPlacements

Declarative command object hierarchy

A few basic objects define a declarative command.



© 2025 Siemens

commandPlacements JSON object

A command can have many placements. A placement is the actual visual rendering of the display of the command. A command placement has the following properties:

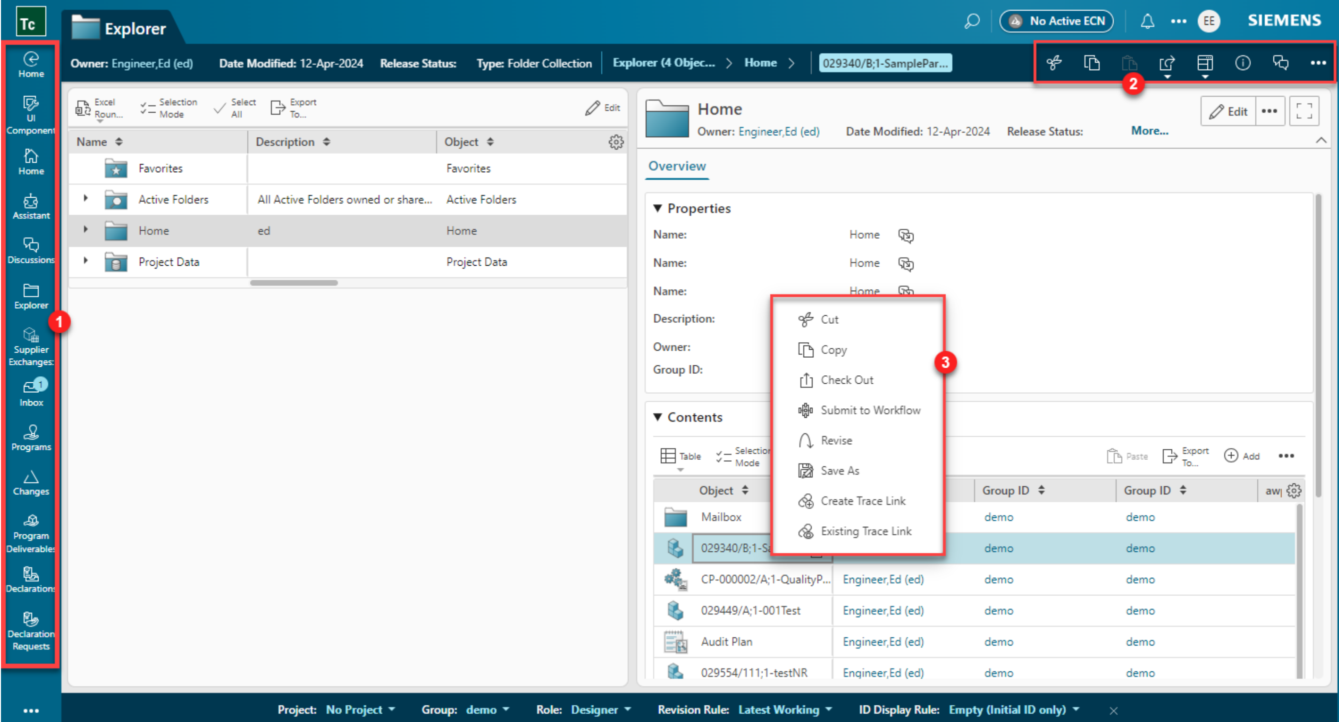
- **id**: The *command Id* for the associated command.
- **uiAnchor**: A well known name for an **<aw-command-bar>** element.
- **priority**: The relative priority order to render commands.
- **relativeTo**: (optional) The *command id* of this command will be placed relative to another command. The **priority** property will be applied relative to the specified command. In other words if multiple commands are placed 'relativeTo' the same command, they will be placed in ascending sorted priority order relative to the specified command. Negative priority means that this command will be inserted before the 'relativeTo' command. Positive priority means the command will be appended after the 'relativeTo' command.

Example uiAnchor names

Following are some common anchor points. There are far too many to list, and they can be discovered easily.

1 — aw_globalNavigationbar
2 — aw_workarea_toolbar
3 — aw_contextMenu2

This is the generic context (right-click) menu anchor point for Active Workspace tables. Declarative tables can also have their own anchor point for context menus, adding additional commands.

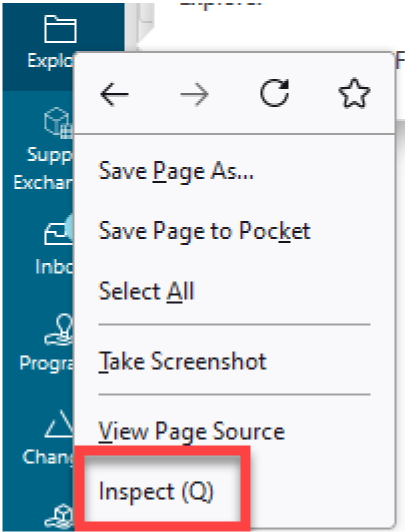


© 2025 Siemens

How can I find other uiAnchor points?

From the developer console in your browser, inspect the page's elements. You will find the anchor listed.

For example, to find out which anchor point the home folder command uses in this position, inspect it using your web browser.

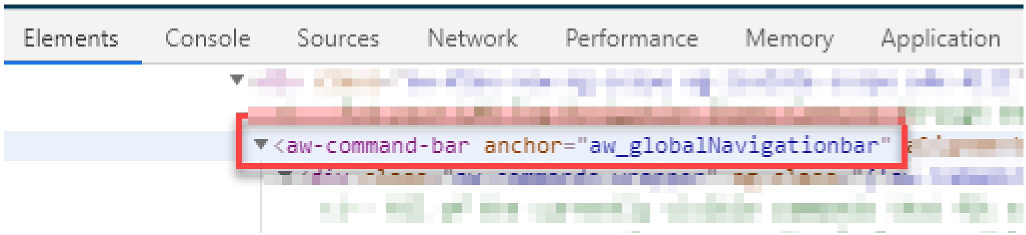


Tip:

This example shows Chrome. Firefox calls this feature **Inspect Element**. Others may vary.

Find the element containing the commands. It will have an attribute named **anchor**. In this case, **aw-command-bar**.

The element inspector shows that this command is placed on the bar using the **aw_globalNavigationbar** anchor point.



Example commandPlacements

Below are some example **commandPlacements**:

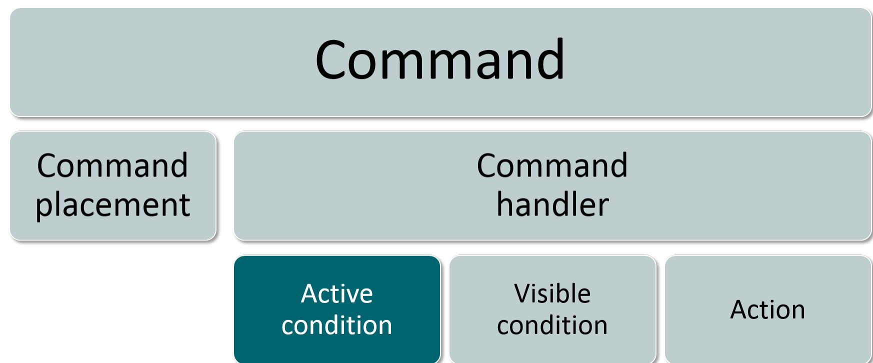
```
"commandPlacements": {
  "Awp0CopyToolsAndInfo": {
    "id": "Awp0Copy",
    "uiAnchor": "aw_viewerCommands",
    "priority": 200
  },
  "Awp0GoBackGlobalNavigationbar": {
    "id": "Awp0GoBack",
    "uiAnchor": "aw_globalNavigationbar",
    "priority": 1
  },
  "Awp0ChangeThemeSessionbar": {
    "id": "Awp0ChangeTheme",
    "uiAnchor": "aw_userSessionbar",
    "priority": 2,
    "showGroupSelected": false
  },
  "Awp0GoHomeGlobalToolbar": {
    "id": "Awp0GoHome",
    "uiAnchor": "aw_globalToolbar",
    "relativeTo": "Awp0ChangeTheme",
    "priority": -1
  },

...
},
```

Declarative command object - activeWhen

Declarative command object hierarchy

A few basic objects define a declarative command.



© 2025 Siemens

activeWhen JSON object

This determines when a command handler is active. A command handler must be both active and visible to display in the UI.

In the following **commandHandlers**, various **activeWhen** conditions are shown.

```
"commandHandlers":
{
    "myCheckoutCommandHandler": {
        "id": "MyCheckout",
        "action": "checkItOut",
        "activeWhen": true,
        "visibleWhen": true
    },
    "inboxFilterCommandHandler": {
        "id": "Awp0InboxFilters",
        "action": "activateFilterPanel",
        "activeWhen": true,
        "visibleWhen": true
    },
    "itemCopyCommandHandler": {
        "id": "Awp0AlertUser",
        "action": "alertItemRevision",
        "activeWhen": {
            "condition": "conditions.alertItemRevisionCommandActive"
        },
        "visibleWhen": {
            "condition": "conditions.alertItemRevisionCommandVisible"
        }
    },
    "itemCopyCommandHandlerQwerty": {
        "id": "Awp0AlertUser",
        "action": "alertItemRevisionSpecial",
        "activeWhen": {
            "condition": "conditions.alertItemRevisionSpecialCommandActive"
        },
        "visibleWhen": {
            "condition": "conditions.alertItemRevisionCommandVisible"
        }
    },
```

> Note:
>
> Declarative conditions can be defined with arbitrary expressions utilizing client side context. However, to accommodate server-side visibility logic the **IApplicationContextService** keeps track of the commands which the server evaluates to be visible. Therefore you can build expressions for your declarative condition that refer to the real-time server-side visibility.
>
> For example, if you want to have a condition expression that uses the server-side visibility of the **Awp0Checkout** command your condition expression would simply be

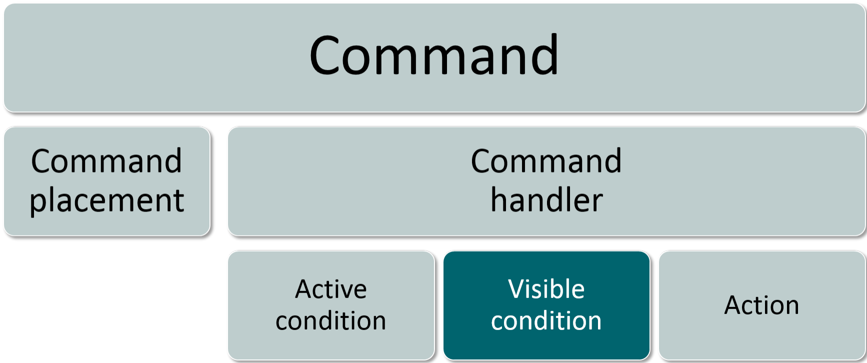
© 2025 Siemens

"`ctx.visibleServerCommands.Awp0Checkout`". This variable can be used by itself or it can be used with other client side expressions.

Declarative command object - visibleWhen

Declarative command object hierarchy

A few basic objects define a declarative command.



visibleWhen JSON object

This determines when a command handler is visible. A command handler must be both active and visible to be displayed in the UI.

In the following **commandHandlers**, various **visibleWhen** conditions are shown.

© 2025 Siemens

```
"commandHandlers":
{
    "myCheckoutCommandHandler": {
        "id": "MyCheckout",
        "action": "checkItOut",
        "activeWhen": true,
        "visibleWhen": true
    },
    "inboxFilterCommandHandler": {
        "id": "Awp0InboxFilters",
        "action": "activateFilterPanel",
        "activeWhen": true,
        "visibleWhen": true
    },
    "itemCopyCommandHandler": {
        "id": "Awp0AlertUser",
        "action": "alertItemRevision",
        "activeWhen": {
            "condition": "conditions.alertItemRevisionCommandActive"
        },
        "visibleWhen": {
            "condition": "conditions.alertItemRevisionCommandVisible"
        }
    },
    "itemCopyCommandHandlerQwerty": {
        "id": "Awp0AlertUser",
        "action": "alertItemRevisionSpecial",
        "activeWhen": {
            "condition": "conditions.alertItemRevisionSpecialCommandActive"
        },
        "visibleWhen": {
            "condition": "conditions.alertItemRevisionCommandVisible"
        }
    },
```

Note:

Declarative conditions can be defined with arbitrary expressions utilizing client side context. However, to accommodate server-side visibility logic the **IApplicationContextService** keeps track of the commands which the server evaluates to be visible. Therefore you can build expressions for your declarative condition that refer to the real-time server-side visibility.

For example, if you want to have a condition expression that uses the server-side visibility of the **Awp0Checkout** command your condition expression would simply be "`ctx.visibleServerCommands.Awp0Checkout`". This variable can be used by itself or it can be used with other client side expressions.

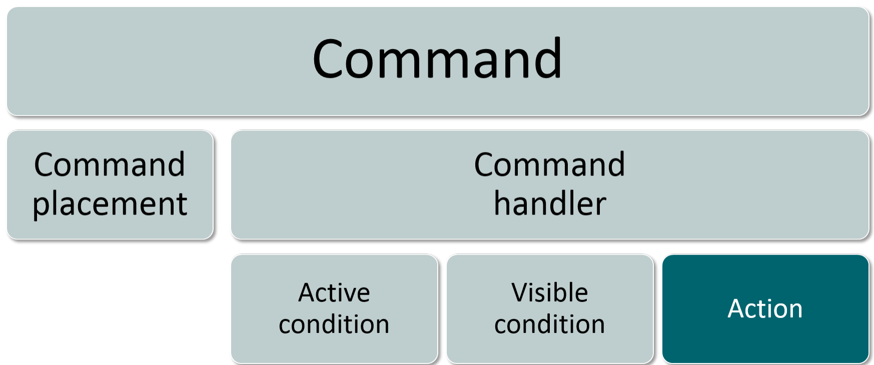
The Business Modeler IDE documentation for your Teamcenter platform contains information on how to create command visibility conditions.

© 2025 Siemens

Declarative command object - actions

Declarative command object hierarchy

A few basic objects define a declarative command.



actions JSON object

- **actionType** supported options:

  TcSoaService
  JSfunction
  JSfunctionAsync
  RESTService
  Navigate

- **inputData**: JSON data for the action input
- **outputData**: JSON data for the action output
- **events**: triggered in response to the action
- **actionMessages**: user messages, and condition support

Following is an example **action** that calls the **checkout** Teamcenter Services

© 2025 Siemens

```
"actions":
{
    "checkItOut":
    {
        "actionType": "TcSoaService",
        "serviceName": "Core-2006-03-Reservation",
        "method": "checkout",
        "inputData":
        {
              "objects" : [{"uid": "{{ctx.selected.uid}}", "type": "{{ctx.selected.type}}"}]
        }
    },
```

Error message processing

The Active Workspace client is responsible for displaying any error messages. If an error is thrown by the framework for example, the client must decide if it ignores it or displays it. The error will get logged to the console regardless. If you write custom actions, they must process any errors you wish to display to the user.

Siemens Web Framework

This functionality is part of the Siemens Web Framework upon which Active Workspace is built. Refer to *Messages* within the Digital Engineering Services product on Support Center.

Data providers

Learn about data providers

What is a data provider?

There are two types of data providers:

client	You can define a client-side data provider using declarative definitions. Use this *model view* mechanism to gather data from Teamcenter or other sources for presentation in the UI.
server	You can define a custom Teamcenter server-side data provider to gather data and send it to a client.

What are the benefits?

All server data providers use a single, common Teamcenter service operation: `Finder::performSearch`. Even custom data providers are covered by this operation, which means you do not have to write a custom service wrapper.

© 2025 Siemens

Client data providers are flexible enough to retrieve data from various sources without requiring Java code.

Use an existing server data provider

> Warning:
>
> The provided server-side data providers that ship with Teamcenter and Active Workspace are not published and are for Siemens Digital Industries Software internal use. They may be removed or changed without notice. The following information is provided for informational purposes only.

You can use the browser's developer tools to examine data providers in action.

1. Open developer tools on your browser to record network traffic.

2. Navigate to a page or perform a search.

3. Filter the network traffic to find **performsearch** calls.

4. Examine the **Request Payload** in the **Headers** tab.

From here you can examine the **searchInput** object.

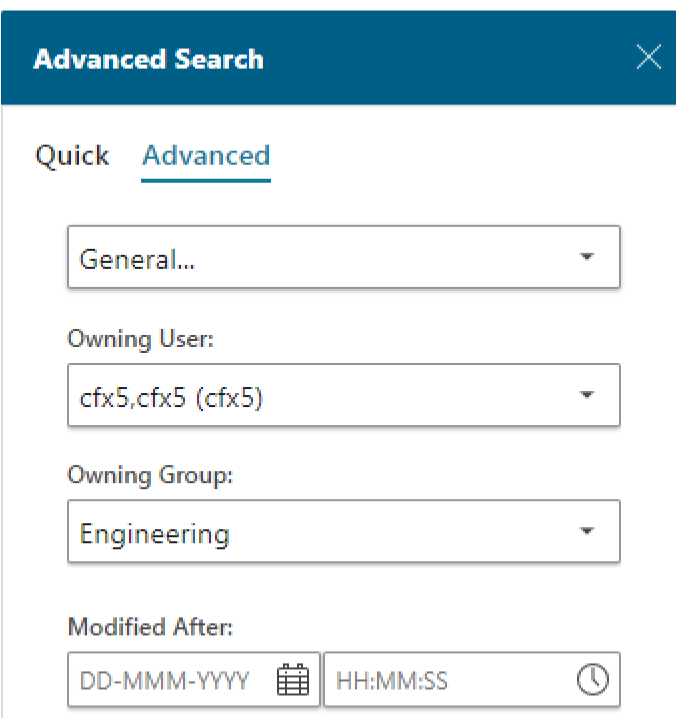
Awp0SavedQuerySearchProvider

Documentation for this provider is being provided on a temporary basis.

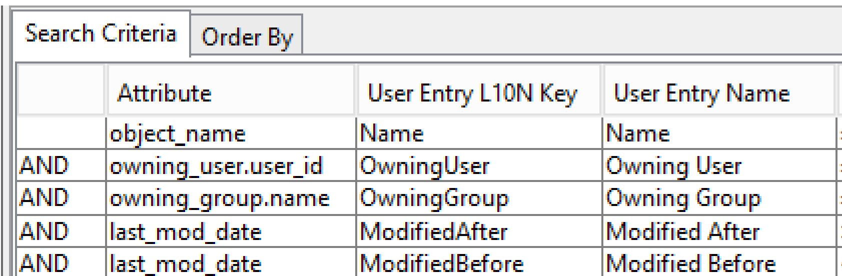
The purpose of this server-side data provider is to run a Teamcenter query and return the results to the client-side data provider.

In this example, the **General...** query is called using the **Advanced Search** capability of Active Workspace.

© 2025 Siemens



Use the Query Builder perspective in the rich client to examine the criteria for the **General...** query.

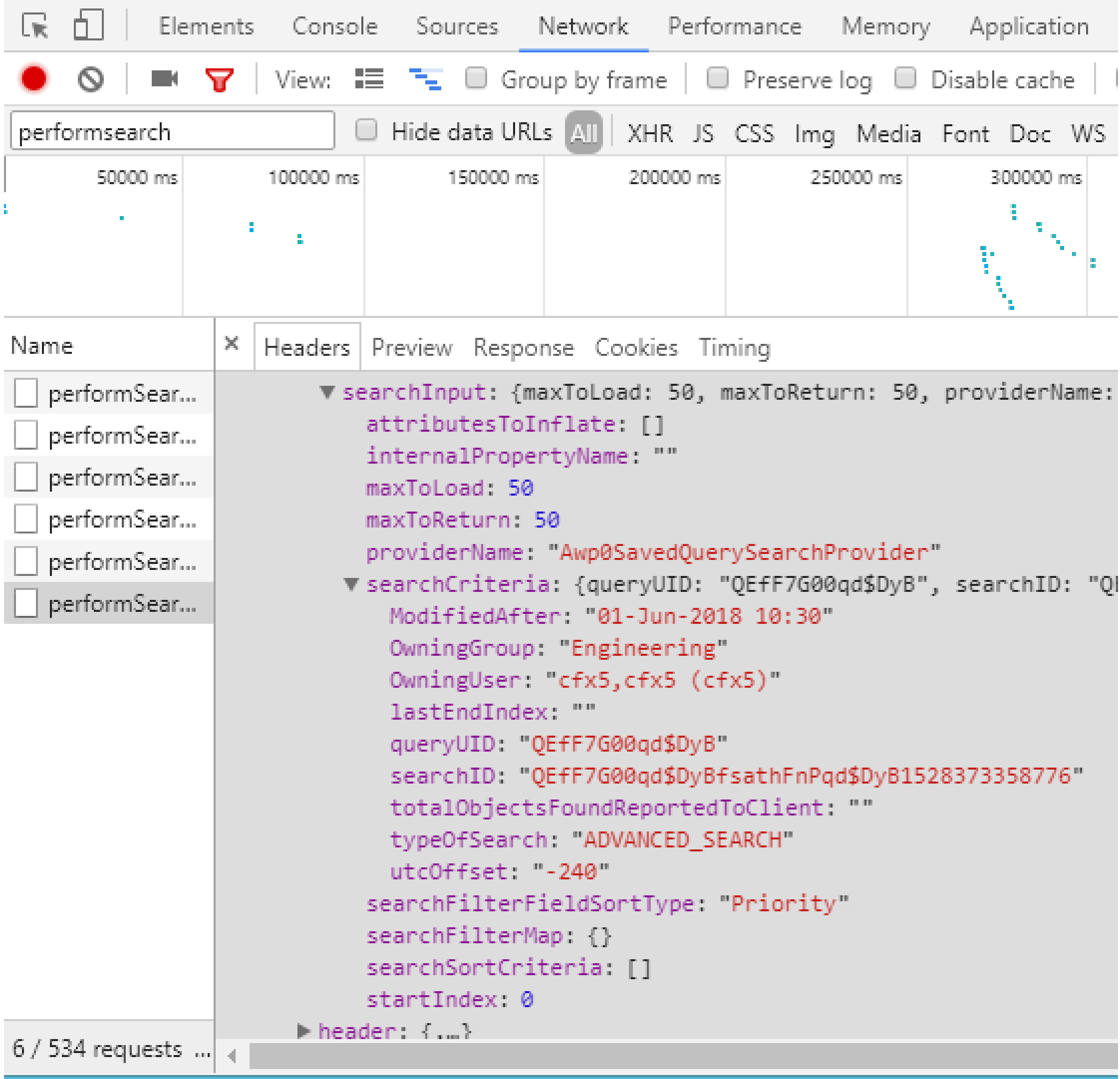


Search Criteria | Order By

	Attribute	User Entry L10N Key	User Entry Name
	object_name	Name	Name
AND	owning_user.user_id	OwningUser	Owning User
AND	owning_group.name	OwningGroup	Owning Group
AND	last_mod_date	ModifiedAfter	Modified After
AND	last_mod_date	ModifiedBefore	Modified Before

Following is the resulting client-side data provider call. Notice how the **searchCriteria** contains all of the information to run the query on the server.

© 2025 Siemens



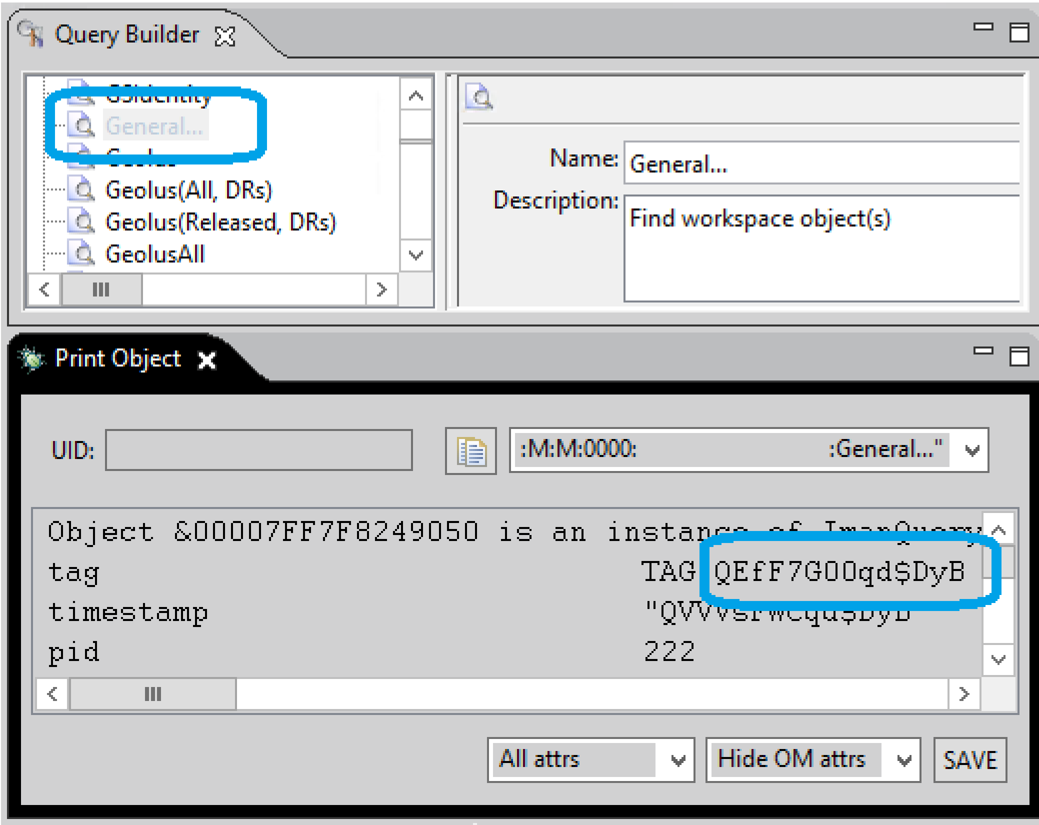
Two of these criteria are always required for this data provider.

searchID	An identifier that must be unique, but does not carry any other significance. It is how the client identifies this particular query call if it needs to.
typeOfSearch	This must provide the type of search being requested.

One of the following two criteria is required to identify which query is run.

queryUID (shown)	Using this criteria removes any question as to which query is run. The **queryUID** is the unique identifier of the query being requested, also known as the C tag or C++ BusinessObjectRef. In this example, **QEff7G00qd$DyB** represents the **General...** query in this Teamcenter database. One way to retrieve this value is to examine the saved query using the rich client and the **Print Object** view.

© 2025 Siemens



queryName Using this criteria is easier, but the possibility exists that the query you want might be deleted and a new query created with the same name.

This is the name of the query. In this example, it would be **General...**.

Example:

```
queryName="General..."
```

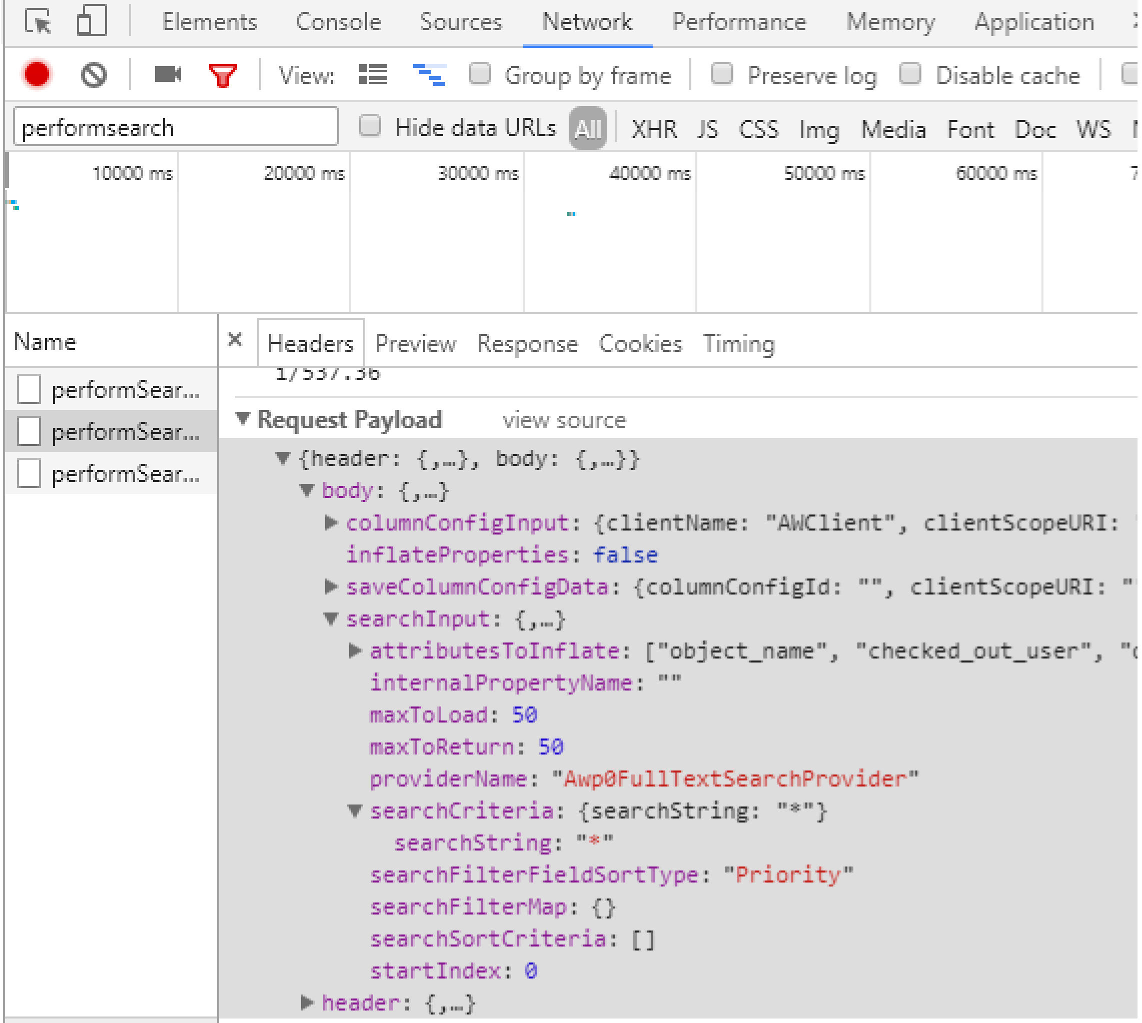
or

```
queryName="Item Revision..."
```

Awp0FullTextSearchProvider

This provider uses the full text indexed search engine, and is used commonly throughout Active Workspace, including the global search area.

© 2025 Siemens



The following criteria are of note:

searchString	The text that will be queried. In this example, the wildcard asterisk is used.
searchFilter Map	This shows any filtering criteria, like object type, owning user, and so on. In this example, there is no filter.

Creating a custom server data provider

You need to prepare several things in order to create your own server data provider.

Use the Business Modeler IDE to:

- Create a child business object of the `Fnd0BaseProvider`.
- Override the `fnd0performSearch` operation.

© 2025 Siemens

- Implement `fnd0performSearchBase` in your custom code.
- Build, package, and deploy your template.

You can now call the `Finder::performSearch` service operation from your client data provider.

Learn client-side data providers

Introduction to the client dataProvider

Use a data provider in your view model to retrieve information from virtually any source. The `dataProviders` object is an abstraction layer for components which fulfill the demand to load or paginate data and pass it to the underlying object.

Where can I use data providers?

You can use data providers in the following UI components.

- aw-list
- aw-tree

Configuring a simple data provider

Data provider objects are defined within a view model file using the following basic parameters.

```
dataProviders: {
}
```

The **UI Pattern Library** on Support Center maintains up-to-date schema definitions.

You can use the following components to define the most basic data provider.

action	Lists the actions available to the provider. These specify how the data is retrieved from the source. These are defined in the **actions** section of the view model file.
response	A data bound array of returned objects.
totalFound	A data bound value of the number of results.

```
dataProviders: {
   "MyDataProvider": {
      "action": "fetchData",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}"
```


© 2025 Siemens

```
        }
    }
```

Example data provider

```
    "dataProviders": {
        "imageDataProvider": {
            "action": "loadData",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}"
        }
    },
    "actions": {
        "loadData": {
            "actionType": "REST",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "startIndex": "{{data.dataProviders.imageDataProvider.startIndex}}",
                    "url": "sample_url"
                }
            },
            "outputData": {
                "totalFound": "{{results.fetcheddata.length}}",
                "searchResults": "{{results.fetcheddata}}"
            }
        }
    }
```

Using a static dataProvider with fixed lists

You can use a data provider to retrieve information from a fixed list.

Static data providers must define their **dataProviderType** to be **Static**.

```
    "dataProviderType": "Static"
```

The data object

The **UI Pattern Library** on Support Center maintains up-to-date schema definitions.

Use the **data** object to store the information you will retrieve.

Example

```
    "dataProviders": {
        "locationLink": {
            "dataProviderType": "Static",
            "response": [
                "{{data.Romulus}}",
```


© 2025 Siemens

```
                "{{data.Remus}}"
            ],
            "totalFound": 2
        }
    }

    "data": {
        "Romulus": {
            "displayName": "Romulus",
            "type": "STRING",
            "dbValue": "Romulus",
            "dispValue": "Romulus"
        },
        "Remus": {
            "displayName": "Remus",
            "type": "STRING",
            "dbValue": "Remus",
            "dispValue": "Remus"
        }

    }
```

Using an action dataProvider for most data

You can use a data provider in response to an action. Action data providers do not need to declare their **dataProviderType**, it is the default.

Additional configurable parameters

selectionModelMode	Indicate the selection mode in the data provider. Valid values are **single** and **multiple**. The default is **single**.
commands	Add commands inside the data provider.
commandsAnchor	Specify a command anchor bar.
preSelection	Indicate if the newly added object will be shown as selected. Valid values are **true** and **false**. The default is **true**.

Events triggered by a data provider

The following events are triggered when:

- {{dataProviderName}}.**selectionChangeEvent**: An object selection change happens in **aw-splm-table** or **aw-list**. This event is triggered with the latest selected object.
- {{dataProviderName}}.**modelObjectsUpdated**: The underlying view model collection is updated.

© 2025 Siemens

- {{dataProviderName}}.**selectAll**: All objects in the data provider are selected.
- {{dataProviderName}}.**selectNone**: All objects in the data provider are deselected.

Example: Configure a data provider as part of an action in the view model

```
"actions": {
    "reveal": {
        "actionType": "dataProvider",
        "method": "imageDataProvider"
    }
}
```

Example: Call multiple data providers as action in the view model

```
"actions": {
    "reveal": {
        "actionType": "dataProvider",
        "methods": ["getHistory", "getFavorites", "performSearch"]
    }
}
```

Example: Pass additional data as input to the data provider

```
"imageDataProvider": {
    "action": "loadData",
    "response": "{{data.searchResults}}",
    "totalFound": "{{data.totalFound}}",
    "inputData": {
        "someData": "{{ctx.abcd}}"
    }
}
```

Configuring pagination for your dataProvider

You can use pagination to help minimize the initial loading time, and also help reduce the load on the server. Pagination support for client data providers work in conjunction with the server. The server should support and return basic parameters which are required for pagination to work. The client data provider has two major fields to enable pagination:

response	An array of data received via a SOA or REST call.
totalFound	Indicates the total number of objects to be loaded in the UI element (list, table, etc.). This number is used by the data provider to calculate the end of pagination.

The data provider calculates the **startIndex** for the next SOA or REST call based on the length of the response data.

© 2025 Siemens

Start index

startIndex is required by the stateless server to send the next set of data upon scrolling.

```
"startIndex" : "{{data.dataProviders.sampledataProvider.startIndex}}"
```

This parameter is calculated and maintained by the data provider. This parameter should be passed as input to the REST or SOA call.

```
"loadData": {
    "actionType": "RESTService",
    "method": "GET",
    "inputData": {
        "request": {
            "method": "GET",
            "startIndex":
"{{data.dataProviders.sampledataProvider.startIndex}}",
            "url": "sample_url"
        }
    }
}
```

Learn the dataProvider selection model

The data provider comes with a default selection model. The primary responsibility of a selection model is to:

- Maintain a list of objects that are selected inside the data provider.
- Keep the internal state information of the selection. Multi-select state, selection mode, and selection status, for example.

Basic methods

setMode

Change selection the mode.

isMultiSelectionEnabled

Check if multi-select mode is active.

setMultiSelectionEnabled

Enable or disable multi-select mode.

isSelectionEnabled

Check if selection is enabled.

setSelectionEnabled

© 2025 Siemens

Enable or disable selection.

evaluateSelectionStatusSummary

Determine the selection state. None selected, some selected, or all selected.

getSelection

Get the current selection.

setSelection

Change the current selection.

addToSelection

Add an object to the current selection

removeFromSelection

Remove an object from the current selection

toggleSelection

Toggle an object's selection status.

selectNone

Clear the current selection. This is an alias for **setSelection**. It does not fire the data provider event that tables expect.

getCurrentSelectedCount

Get the number of objects selected.

isSelected

Check if an object is selected.

getSortedSelection

Get all selected objects and sort them by the order determined in the selection model.

Example: Basic selection model

You can specify multi-select capability using the **selectionModelMode** parameter.

```
"imageDataProvider": {
    "action": "loadData",
    "response": "{{data.searchResults}}",
    "totalFound": "{{data.totalFound}}",
    "selectionModelMode": "multiple"
}
```


© 2025 Siemens

Example: Customize the selection model

You can add a custom selection model based on your needs. Use the **inputData** parameter on the data provider.

```
"imageDataProvider": {
    "action": "loadData",
    "response": "{{data.searchResults}}",
    "totalFound": "{{data.totalFound}}",
    "inputData": {
        "selectionModel": "{{data.selectionModel}}"
    }
}
```

Learn about sorting and filtering with your dataProvider

You must perform all sorting and filtering on the server. The data provider does not have its own sorting or filtering capabilities, though it can assist server by sending the sort and filtering criteria from the **action** associated with the data provider.

```
"dataprovider": {
    "gridDataProvider": {
        "action": "loadData",
        "response": "{{data.searchResults}}",
        "totalFound": "{{data.totalFound}}",
        "inputData": {
            "selectionModel": "{{subPanelContext.selectionModel}}",
            "searchSortCriteria": "acending"
        }
    }
},
"action": {
    "loadData": {
        "actionType": "RESTService",
        "serviceName": "GET",
        "inputData": {
            "request": {
                "method": "GET",
                "withCredentials": false,
                "url": "https://some-url",
                "searchInput": {
                    "searchFilterMap": "{{ctx.activeFilterMap}}",
                    "searchSortCriteria":
"{{data.dataprovider.gridDataProvider.sortCriteria}}",
                    "startIndex": "{{data.dataProviders.gridDataProvider.startIndex}}"
                }
            }
        }
    }
}
```


© 2025 Siemens