# Siemens Web Framework

Software Version Siemens Web Framework

SIEMENS

Unpublished work.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 publication 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: **https://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: **https://support.sw.siemens.com**

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

# Table of Contents


Overview ..... 1-1
Packages ..... 1-2
Storybook ..... 1-7
Prerequisites ..... 1-14
Getting started ..... 1-17
Changelog ..... 1-47

Core concepts ..... 2-1
Why declarative UIs? ..... 2-3
Main declarative UI artifacts ..... 2-5
Component ..... 2-5
Properties vs. state ..... 2-8
Rendering components ..... 2-10
Mapping React hooks to declarative constructs ..... 2-12
View ..... 2-20
View model ..... 2-22
Data ..... 2-24
Actions ..... 2-31
Functions ..... 2-49
Data providers ..... 2-51
Data parsers ..... 2-60
Events ..... 2-65
Lifecycle hooks ..... 2-71
I18n and l10n ..... 2-76
Messages ..... 2-82
Conditions ..... 2-84
Context ..... 2-93
UI elements ..... 2-104
Command ..... 2-107
Group command ..... 2-121
Shuttle command ..... 2-127
Ribbon command ..... 2-129
Toggle command ..... 2-130
Cell command ..... 2-131
Commands in command panel and panel section ..... 2-134
Table commands ..... 2-136
Context menu command ..... 2-137
Command overflow ..... 2-138
Application and component-level State ..... 2-141
Application context ..... 2-143
Services ..... 2-148




Unpublished work. © 2025 Siemens

Tutorials ..... 3-1
Your first *Hello World* application ..... 3-2
Configuring the common frame ..... 3-6
URL routing and page layout ..... 3-13
Lists with data provider and selection ..... 3-18
Selection summary ..... 3-25
Working with commands ..... 3-31
Property widgets and validation ..... 3-44
Editing in a summary view ..... 3-52
Localization ..... 3-59
Working with tables ..... 3-72
Editing data in a table ..... 3-86
Selection summary with table ..... 3-98
Local and global selections ..... 3-102
Selection master-details pattern ..... 3-107

Advanced Tutorials ..... 4-1
Creating reusable components ..... 4-1
Creating a component using a declarative view ..... 4-2
Creating a component using a JSX render function - Example 1 ..... 4-8
Creating a component using a JSX render function - Example 2 ..... 4-12
Creating a component performing DOM manipulation ..... 4-15
Creating a component integrating a third-party React component ..... 4-18
Creating a component integrating a third-party vanilla JavaScript component ..... 4-22
Interaction Patterns between Components ..... 4-27
Example Components ..... 4-27
Component Interaction using Global Configuration ..... 4-31
Component Interaction using Subscribe/Publish of Events ..... 4-34
Component Interaction using Sync Strategy ..... 4-39
Component Interaction using Inverse Flow ..... 4-43
Comparison between various Component Interaction Patterns ..... 4-48
Building Custom Pages ..... 4-49
Setup mock server for Tracking App ..... 4-49
Building pages using Common Location ..... 4-53
Building sub-pages using Common Location ..... 4-56
Building lists using Standard sublocation ..... 4-60
Building tables using Standard sublocation ..... 4-65
Building reusable pages using Standard sublocation ..... 4-70
Building details pages using Standard sublocation ..... 4-74
Building an alternate primary workarea pages using standard sublocation ..... 4-82
Configuring Commands ..... 4-89
Configuring command placement, visibility, and action ..... 4-89
Configuring an extended tooltip for a command ..... 4-98
Configuring context menu commands ..... 4-100
Reusing the same command in different places ..... 4-105
Configuring cell commands ..... 4-107
Configuring different behaviors for the same command ..... 4-112


Unpublished work. © 2025 Siemens

**Managing Dialogs, Property Widgets, Forms and Validations** ..... **4-117**
Managing Dialogs ..... 4-118
Property Widgets, Forms and Validations ..... 4-121
Dynamic Widgets and Custom Validations ..... 4-155

## Creating your solution ..... 5-1

**Modules** ..... **5-6**
**Kits** ..... **5-8**
**Sites** ..... **5-10**
**Workspaces** ..... **5-11**

## Creating User Interfaces ..... 6-1

**URL Routing** ..... **6-1**
Subpage Visibility ..... 6-7
**Page-Level Components** ..... **6-9**
Common Frame ..... 6-13
Common Root ..... 6-14
Location ..... 6-16
Sublocation ..... 6-23
Location and Sublocation Configuration ..... 6-64
Location and Sublocation Examples ..... 6-65
**Layout Elements** ..... **6-86**
**Using custom stylesheets** ..... **6-113**
**Command Sub Panels** ..... **6-114**
**Table** ..... **6-118**
**Breadcrumb** ..... **6-119**
**Data Flow Patterns** ..... **6-121**
Uni-directional Data Flow ..... 6-123
Inverse Data Flow ..... 6-124
Complex Inverse Data Flow ..... 6-125
**Data Communication** ..... **6-126**
subPanelContext ..... 6-127
Atomic data ..... 6-129
**Selection Management** ..... **6-142**
selectionData ..... 6-144
selectionModel ..... 6-151
**Form Validation** ..... **6-153**
Validation Criteria ..... 6-161
**Accessing the DOM** ..... **6-171**
**Responsiveness** ..... **6-172**
**Reusability** ..... **6-180**

## Configuration and Extensibility ..... 7-1

**Codeful extensions** ..... **7-1**
**Context configuration** ..... **7-2**
**Clipboard service** ..... **7-4**


Unpublished work. © 2025 Siemens

**Bootstrap** ..... **7-6**
**Authentication** ..... **7-9**
**Solution configuration** ..... **7-17**
**Custom locale configuration** ..... **7-25**
**Adapters** ..... **7-27**
**Extended tooltips** ..... **7-31**
**Layout slots** ..... **7-42**
**Sync strategy** ..... **7-44**
**Drag and drop** ..... **7-48**
**Navigation URL Token Configuration *(Teamcenter-specific)*** ..... **7-56**
**Tab** ..... **7-57**

**Tools** ..... **8-1**
**Default NPM scripts** ..... **8-1**
**Environment variables** ..... **8-4**
**Configure custom production bundles** ..... **8-6**
**Configure custom production bundles using bundles.json** ..... **8-13**
**Using the Command Line Interface** ..... **8-15**
Using code generators ..... 8-18
**Implementing unit tests** ..... **8-24**
**Running audit checks** ..... **8-29**
**Configuring JSON IntelliSense and validation** ..... **8-32**
**Enabling runtime debug logs** ..... **8-36**
**Siemens Web Framework DevTools** ..... **8-39**
**React Development Tools extension for Google Chrome** ..... **8-46**
**Redux Development Tools extension for Google Chrome** ..... **8-48**
**Recommended Visual Studio Code extensions** ..... **8-50**

**Upgrading** ..... **9-1**
**Upgrading to version 4.x** ..... **9-1**
Webpack Build ..... 9-2
ES6 Conversion ..... 9-6
Unit Test Conversion ..... 9-9
Service Conversion ..... 9-11
Manual Service Conversion ..... 9-12
Upgrading to Version 4.1 ..... 9-14
Upgrading to Version 4.2 ..... 9-15
Upgrading to Version 4.3 ..... 9-22
**Upgrading to version 5.x and 6.x** ..... **9-23**
Beyond Angular Rationale ..... 9-23
What's new? ..... 9-26
Functional Limitations ..... 9-28
Evaluating the impact of the upgrade ..... 9-28
Upgrading your project configuration ..... 9-32
Changes to Declarative Contracts ..... 9-35
Removing AngularJS-specific code ..... 9-43


Unpublished work. © 2025 Siemens

Converting AngularJS directives to components ... 9-44
Changes to Data Management ... 9-51
Changes to routing and layout ... 9-57
Changes to specific UI elements ... 9-58
Loading SVG images ... 9-70
Migrating unit tests to Jest ... 9-76
Upgrading to version 6.1 ... 9-80
**Upgrading to version 7.x ... 9-81**
Updating your package.json file ... 9-81
Migrating from aw-sidenav to aw-dialog ... 9-82
Changes to user session controls ... 9-93
Change in right wall command placement ... 9-94
Upgrading to version 7.1 ... 9-94

**Best Practices ... 10-1**
**Implementation best practices ... 10-1**
**Troubleshooting best practices ... 10-2**
**Security best practices ... 10-4**
**Performance best practices ... 10-8**

**Reference ... 11-1**
**Components ... 11-1**
aw-advanced-bar ... 11-1
aw-avatar ... 11-2
aw-back-button ... 11-3
aw-balloon-popup-panel ... 11-4
aw-break ... 11-4
aw-button ... 11-4
aw-cell-header ... 11-6
aw-chart ... 11-6
aw-cheat-sheet ... 11-7
aw-checkbox ... 11-7
aw-checkbox-list ... 11-8
aw-check-list ... 11-8
aw-chip ... 11-9
aw-chip-list ... 11-9
aw-column ... 11-10
aw-column-chart ... 11-11
aw-command-bar ... 11-11
aw-command-panel ... 11-13
aw-command-panel-section ... 11-14
aw-command-sub-panel ... 11-15
aw-compare2 ... 11-16
aw-compile ... 11-16
aw-date ... 11-17
aw-datetime ... 11-18
aw-default-cell ... 11-19
aw-default-group-cell ... 11-19


Unpublished work. © 2025 Siemens

aw-default-sublocation....11-20
aw-dialog....11-21
aw-echart....11-22
aw-echart-advanced....11-23
aw-empty-workarea....11-24
aw-expandable-row....11-25
aw-i18n....11-25
aw-file-upload....11-26
aw-flex-column....11-28
aw-flex-row....11-29
aw-frame....11-30
aw-full-screen....11-30
aw-guidance-message....11-32
aw-header-context....11-33
aw-header-props....11-33
aw-icon-button....11-34
aw-image....11-35
aw-image-cell....11-35
aw-include....11-36
aw-label....11-36
aw-layout-container....11-37
aw-layout-slot....11-37
aw-link....11-38
aw-link-with-popup....11-38
aw-link-with-popup-and-icon....11-39
aw-link-with-popup-menu....11-39
aw-list....11-40
aw-listbox....11-42
aw-list-filter....11-42
aw-navigate-breadcrumb....11-43
aw-navigate-panel....11-44
aw-notification-cell....11-44
aw-numeric....11-45
aw-panel....11-45
aw-panel-section....11-46
aw-password....11-47
aw-pic....11-47
aw-popup....11-48
aw-popup-close-button....11-49
aw-popup-draggable....11-49
aw-popup-panel....11-50
aw-popup-panel2....11-50
aw-popup-primary-button....11-50
aw-popup2....11-51
aw-progress-indicator....11-51
aw-property....11-52
aw-property-boolean....11-52
aw-property-image....11-53
aw-property-numeric....11-53


Unpublished work. © 2025 Siemens

aw-property-string....11-53
aw-property-string-val....11-54
aw-property-val....11-54
aw-radiobutton....11-54
aw-row....11-55
aw-scrollpanel....11-56
aw-selection-summary....11-56
aw-search-box....11-58
aw-separator....11-59
aw-session-controls....11-59
aw-sidenav....11-60
aw-source-editor....11-61
aw-slider....11-63
aw-splitter....11-63
aw-splm-table....11-64
aw-summary-header....11-67
aw-step-progress....11-68
aw-tab-bar....11-69
aw-tab-set....11-69
aw-textarea....11-70
aw-textbox....11-71
aw-tile....11-72
aw-tile-canvas....11-72
aw-tile-group....11-73
aw-time....11-73
aw-togglebutton....11-74
aw-toolbar....11-74
aw-tree....11-76
aw-warning-label....11-77
aw-widget....11-77
**Attributes....11-78**
aw-class....11-78
aw-click....11-78
aw-enter....11-79
aw-repeat....11-79
enable-when....11-80
exist-when....11-80
extended-tooltip....11-80
show-when....11-85
visible-when....11-86
**Types....11-86**
ActiveCommandConfig....11-87
AdvancedBarOrientationEnum....11-87
AdvancedBarTypeEnum....11-87
AlignContentEnum....11-87
AlignmentTypeEnum....11-88
ArrowOptions....11-88
AvatarSizeEnum....11-88
AwWidgetOptionsConfiguration....11-89

awHttpInstance..... 11-89
awParseServiceInstance..... 11-91
AwStateInstance..... 11-92
BannerContext..... 11-92
BarOrientationEnum..... 11-93
ButtonSizeEnum..... 11-93
ButtonTypeEnum..... 11-93
ButtonWidthEnum..... 11-94
CellProperty..... 11-94
ChartPointsConfig..... 11-94
ChartProvidersConfiguration..... 11-95
ChartsConfigObject..... 11-95
ChartTypeEnum..... 11-96
ChartYaxisValuesEnum..... 11-96
CheckListConfiguration..... 11-96
CheckListValue..... 11-97
ChipConfiguration..... 11-97
ChipTypeEnum..... 11-98
Column..... 11-98
ColumnChartConfiguration..... 11-100
ColumnConfiguration..... 11-100
ColumnProviderConfiguration..... 11-102
ColumnValue..... 11-102
CrumbsConfiguration..... 11-103
CrumbsObject..... 11-103
CurrentLocationData..... 11-103
CurrentStateData..... 11-104
CurrentSublocationData..... 11-104
DataProvider..... 11-104
DataProviderConfiguration..... 11-106
DataProviderTypeEnum..... 11-107
DeclarativeViewModel..... 11-108
DialogHeight..... 11-108
DialogOptions..... 11-108
DialogPlacement..... 11-109
DialogWidth..... 11-109
DirectionTypeEnum..... 11-110
DisplayedViewModes..... 11-110
EditorConfigOptions..... 11-110
EditorLanguageEnum..... 11-111
EditorThemeEnum..... 11-112
EchartProvidersConfiguration..... 11-112
EventBusEventCallback..... 11-113
ExtendedTooltipFlipBehavior..... 11-113
ExtendedTooltipOptions..... 11-113
ExtendedTooltipPlacement..... 11-114
FlipBehaviorOptions..... 11-114
Grid..... 11-114
GridOptions..... 11-115


Unpublished work. © 2025 Siemens

HeaderContextAlignmentTypeEnum..... 11-117
HeaderPropertiesConfiguration..... 11-117
IconButtonConfiguration.....11-117
IconOptionsEnum.....11-118
Indicator.....11-118
JustifyContentEnum..... 11-118
LabelPositionEnum.....11-119
Layout.....11-119
LayoutContext.....11-120
LayoutGridSize..... 11-120
LinkIconOptionsConfiguration.....11-120
LinkOptionsConfiguration..... 11-121
LinkPopupResponsePropertyValue.....11-121
LocalizedLabel..... 11-121
LocalStorageEventCallback..... 11-122
LocationContext..... 11-122
LocationTitles..... 11-122
MessageConfigurationObject..... 11-123
MessageDefn.....11-123
MessageNavigationOption..... 11-124
MessageTextLink..... 11-124
MessageTypeEnum.....11-124
NavigateAction.....11-124
NotificationCellProperty..... 11-125
NotificationLevelTypeEnum.....11-125
PanelCollapseEnum..... 11-126
PicClassEnum..... 11-126
PlacementOptions..... 11-126
PointCoordinates..... 11-127
PopupAlignmentEnum.....11-127
PopupOptions.....11-127
PopupHooks..... 11-133
PropertyLoadInput.....11-133
PropertyTypeEnum..... 11-133
PropertyValue.....11-134
RenderingHintEnum..... 11-135
RenderingStyleEnum..... 11-135
ResponsiveLayoutGridSizes..... 11-136
ScrollingParentBehaviors..... 11-136
SelectionModel..... 11-136
SelectionModelModeEnum.....11-138
SeriesMarker..... 11-139
SeriesTooltip.....11-139
SeriesTypeEnum..... 11-139
SideNavConfigObject..... 11-140
SideNavHeightEnum.....11-140
SideNavWidthEnum.....11-140
SidenavModeEnum.....11-141
SliderLabelMap.....11-141

SliderOption..... 11-141
SliderOrientation..... 11-142
SliderValue..... 11-142
SlideTypeEnum..... 11-143
SortCriteriaConfiguration..... 11-143
SortDirectionEnum..... 11-143
SourceEditorConfig..... 11-144
StepConfiguration..... 11-144
SupportedViewModes..... 11-144
TabConfiguration..... 11-145
TableContext..... 11-145
TableLoadResult..... 11-146
TableRowHeightEnum..... 11-146
TableTreeNode..... 11-147
TextPositionEnum..... 11-147
ThemeIndexEnum..... 11-147
TileActionObject..... 11-148
TileActionTypeEnum..... 11-148
TileConfiguration..... 11-148
TileContentObject..... 11-149
TileGroupConfiguration..... 11-149
TileIconsObject..... 11-150
TileSizeEnum..... 11-150
TransistionHooks..... 11-150
TreeData..... 11-151
TreeLoadInput..... 11-151
UploadModel..... 11-152
UploadModel_Object..... 11-153
UploadedFileMetadata..... 11-153
UploadedFiles..... 11-154
UploadedFilesMetadata..... 11-154
ViewModeContext..... 11-154
ViewModelCollection..... 11-155
ViewModelProperty..... 11-156
ViewModelPropertyAttributeHolder..... 11-163
ViewModelObject..... 11-164
ViewModelTreeNode..... 11-164
ViewModelTreeNodeArray..... 11-165
ViewTuple..... 11-166
WrapStyleEnum..... 11-166
**AppCtx..... 11-166**
**Configuration points..... 11-169**
Action Schema..... 11-169
Adapters Schema..... 11-215
AliasRegistry Schema..... 11-220
BatchJobViewModel Schema..... 11-221
Build Schema..... 11-230
CommandsViewModel Schema..... 11-238
Condition Schema..... 11-265


Unpublished work. © 2025 Siemens

Decorators Schema....11-269
Delta Schema....11-276
DragAndDrop Schema....11-283
Indicators Schema....11-291
IndicatorObject Schema....11-295
Kit Schema....11-297
Module Schema....11-314
NavigationURLToken Schema....11-321
Paste Schema....11-325
PropertyRendererTemplates Schema....11-340
ReusableSchemaDefs Schema....11-343
SecondaryWorkareaTabs Schema....11-372
Site Schema....11-382
SPLMTableViewModel Schema....11-384
State Schema....11-392
States Schema....11-400
SyncStrategy Schema....11-400
TabsViewModel Schema....11-415
TypeIconsRegistry Schema....11-433
TypeIconsRegistryObject Schema....11-434
ViewModel Schema....11-438
Workspace Schema....11-483
WorkspaceContribution Schema....11-489
**Services....11-494**
js/appCtxService....11-494
js/awHttpService....11-496
js/awParseService....11-496
js/awStateService....11-496
js/bannerService....11-497
js/browserUtils....11-497
js/clipboardService....11-499
js/conditionService....11-500
js/configurationService....11-501
js/dateTimeService....11-501
js/dialogService....11-507
js/eventBus....11-508
js/iconService....11-509
js/localStorage....11-510
js/localeService....11-511
js/logger....11-513
js/messagingService....11-514
js/modelPropertyService....11-516
js/navigationService....11-517
js/pageLayoutService....11-518
js/pasteService....11-519
js/popupService....11-520
js/splmTablePublishedService....11-521
js/splmTablePublishedTreeService....11-525


Unpublished work. © 2025 Siemens

Glossary..... 12-1



Unpublished work. © 2025 Siemens

# 1. Overview

Siemens Web Framework is a configuration-first framework that lets you rapidly develop web applications with significantly less code to write, test, maintain and download at run time. It follows top trends in the front-end development like component-oriented design, declarative coding, consolidation of state management, SPA and client-side routing, and it allows extensions using JavaScript code to manage application complexity.

At Siemens, we use Siemens Web Framework to deliver several web applications that leverage Teamcenter as well as next-generation apps for the cloud.

## Main Features

Siemens Web Framework provides:

- A complete framework to realize your *end-to-end solution* for your frontend applications, allowing you to wire-in your own backend through REST APIs, Websockets, etc.
- A highly *modular* architecture based on *modules* and *kits* to facilitate maintainability, code ownership, and reuse.
- A *declarative* approach to develop user interfaces in a technology-independent way, without being tied to any JavaScript framework.
- A rich set of ready-made *UI Elements* that can be used to build web applications that are consistent with our common user experience.
- The ability to define your own *theme* for your web application.
- Built-in web *analytics* to collect page visits and usage data throughout your application.
- Support for unit testing (via Jest and Testing Library) and BDD *testing* via a dedicated automation framework leveraging Cucumber syntax.
- Comprehensive built-in *tooling* for building your web application, auditing your code, and debugging.
- A public *storybook* web site with a lot of working examples of all the available UI elements.
- A comprehensive set of useful *tutorials* to get started.

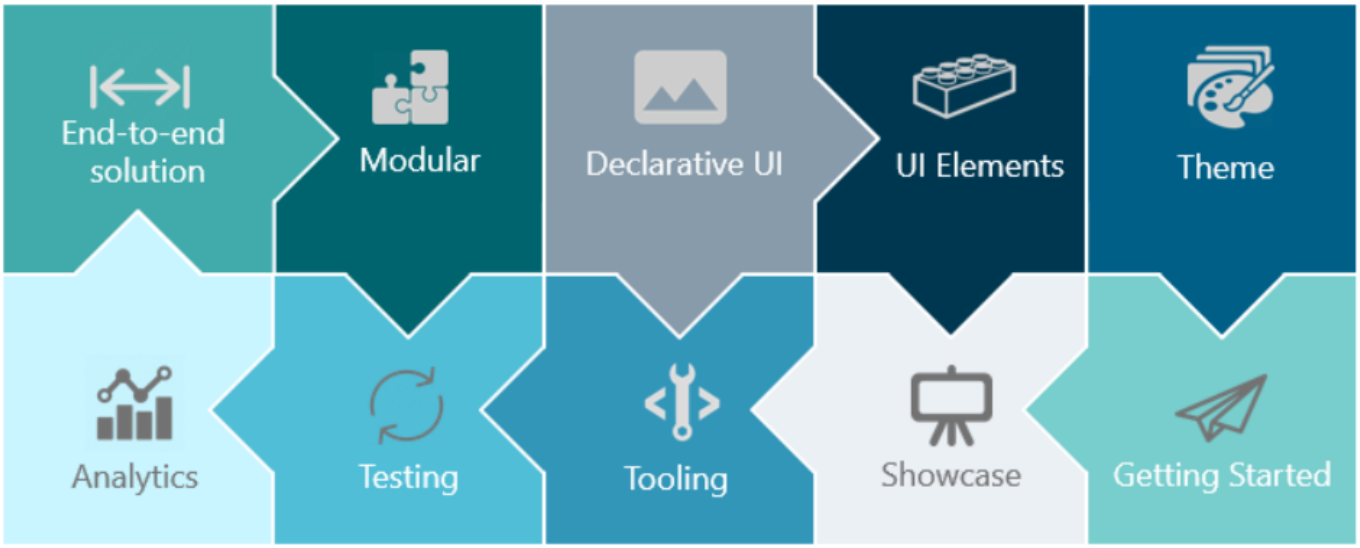
## Getting Started

To get started with Siemens Web Framework:

1. Setup your **environment** to start building your web applications.
2. Follow the **first tutorial** to build a simple *Hello World* application.
3. Follow the rest of the **tutorials** to learn the basics of the framework.
4. Dive into the **declarative concepts** to understand the overall philosophy around the framework.


Unpublished work. © 2025 Siemens

5. Explore the comprehensive **Siemens Web Framework Storybook** to see what UI elements are already provided by the framework.
6. Follow the **advanced tutorials** to learn more advanced concepts, once you are comfortable with the basics.
7. Go through the **best practices** and the rest of the documentation to learn the best way to develop web applications.




# Packages

Siemens Web Framework is comprised of different NPM packages, these can be:

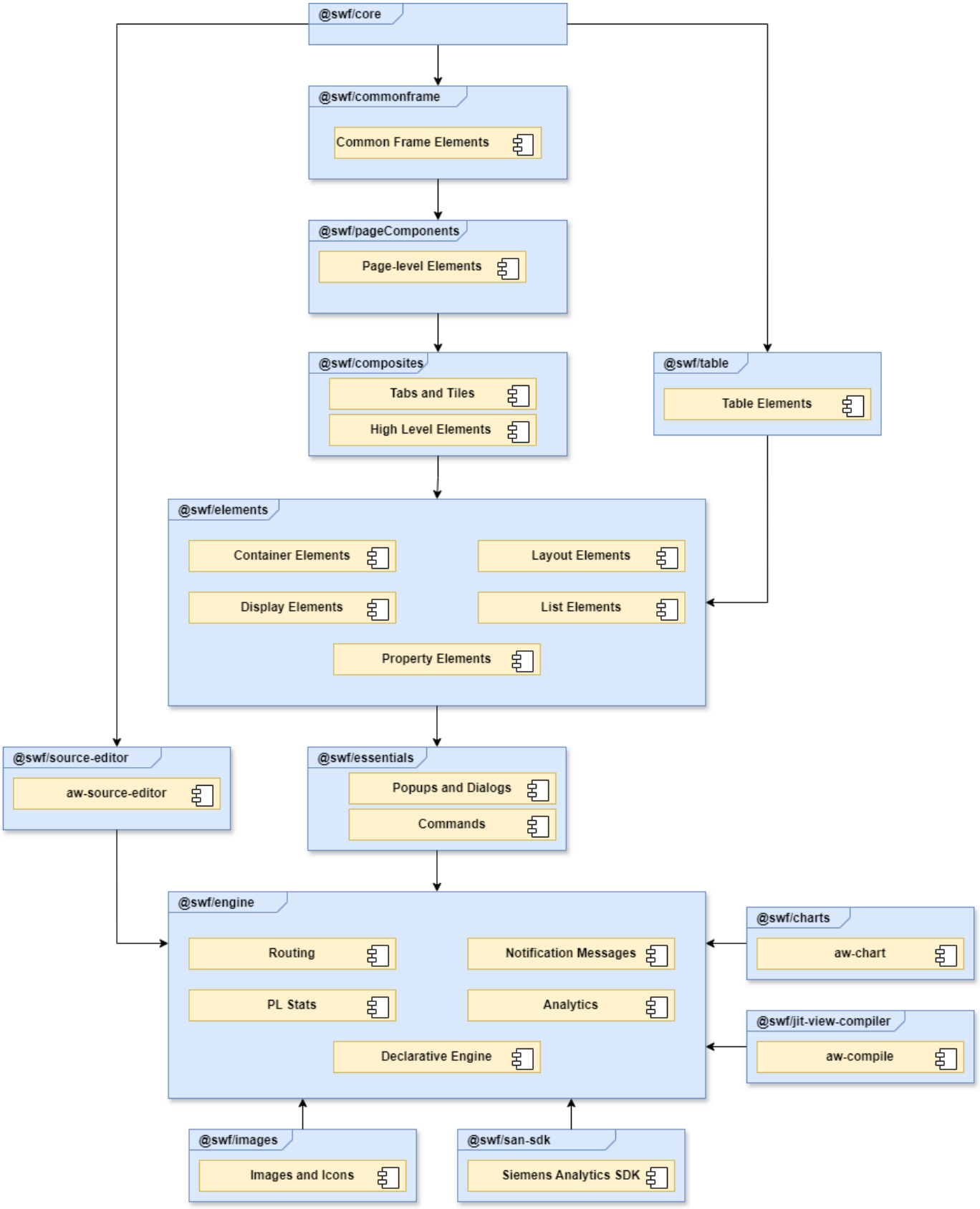
- runtime packages, typically distributed to your consumers and specified as *dependencies* of your SWF applications
- development packages, typically not distributed to your consumers and specified as *devDependencies* of your SWF applications

## Runtime packages

As of version 7.1, Siemens Web Framework has a more modular architecture and its runtime is delivered through many different NPM packages, each delivering a specific set of functionalities.

The following diagram shows the dependencies between each of these packages:


Siemens Web Framework
Unpublished work. © 2025 Siemens





Unpublished work. © 2025 Siemens

If you are creating a complete application, typically specifying `@swf/core` will automatically include all packages. For more advanced use cases in which you may not need every feature and every component provided by SWF such as when creating standalone reusable components you can choose to use lower-level packages such as `@swf/table` or `@swf/engine` directly.

**Warning**

SWF modularization is still underway! While you are welcome to use any of the following packages in your application, be aware that their contents may change in future minor or major versions of Siemens Web Framework.

## @swf/core

This is the main package to use at run time in SWF application, as it will automatically pull all the other runtime packages as dependencies.

## @swf/source-editor

This is an optional package that depends on `@swf/engine` and provides the source editor component which relies on the Monaco library to deliver a comprehensive rich text editor.

## @swf/charts

This is an optional package that depends on `@swf/engine` and provides the **aw-chart** component which relies on the **HighCharts** library to deliver a comprehensive charting solution.

**Note**

A license of HighCharts must be purchased in order to be able to use this package.

## @swf/jit-view-compiler

This is an optional package that depends on `@swf/engine` and provides the **aw-compile** component which can be used to render views and view models at run time. This can be useful in certain use cases when you need to implement highly-dynamic user interfaces that would rely on views and view models generated or updated at run time as opposed to development time.

## @swf/commonframe

**Warning**

This package is *experimental*: it is fully supported and you can use it in production, but be aware that its contents may change in future major or minor versions of Siemens Web Framework.

This package includes frame-level widgets like common root, global banner, global avatar etc


Siemens Web Framework
Unpublished work. © 2025 Siemens

## @swf/pagecomponents

**Warning**

This package is *experimental*: it is fully supported and you can use it in production, but be aware that its contents may change in future major or minor versions of Siemens Web Framework.

This package includes page-level widgets like:

- Location elements, such as default location, sublocations, layout manager etc

## @swf/composites

**Warning**

This package is *experimental*: it is fully supported and you can use it in production, but be aware that its contents may change in future major or minor versions of Siemens Web Framework.

This package includes high-level widgets like tabs, tiles, cards etc

## @swf/table

This package distributes the **table** component, an advanced table component able to render data in tabular or hierarchical form.

## @swf/elements

**Warning**

This package is *experimental*: it is fully supported and you can use it in production, but be aware that its contents may change in future major or minor versions of Siemens Web Framework.

This package includes common mid-level widgets like:

- container elements, such as panels and sections
- list elements, such as lists and cells
- layout elements, such as rows and columns
- display elements, such as buttons and images
- property elements, such textboxes and other form controls.


Unpublished work. © 2025 Siemens

## @swf/essentials

**Warning**

This package is *experimental*: it is fully supported and you can use it in production, but be aware that its contents may change in future major or minor versions of Siemens Web Framework.

This package includes support for commands, popups, and dialogs.

## @swf/engine

This package contains the minimal engine for an SWF application, comprising the declarative engine which provides the abstraction layer on top of React, routing, notification messages, PLStats and bindings for the Siemens Software Analytics SDK.

## @swf/images

This package contains images (mostly SVG icons) icons that can be used with Siemens Web Framework.

## @swf/san-sdk

This package provides the integration with the Siemens Analytics SDK, providing built-in support for gathering data about user interaction with certain graphical elements like commands, tabs, and similar.

## Development packages

The following SWF packages are only used at development time, and are typically not distributed to consumers.

## @swf/cli

This package delivers the **SWF CLI**, a powerful command-line interface that can be used to create and manage SWF projects in a more productive way.

**Note**

This package must be **installed as a global package**, and it is meant to support all versions of Siemens Web Framework from version 6.0 onwards.

## @swf/tooling

This package is necessary to be able to build any SWF application, and it also contains some useful scripts and general tooling.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Note**

This package must be specified as in the `devDependencies` section of the `package.json` file of your SWF project.

### @swf/eslint-config-swf

This optional package is needed to support **declarative audit checks**.

**Note**

This package must be specified as in the `devDependencies` section of the `package.json` file of your SWF project.

## Storybook

A catalog of all UI elements provided by Siemens Web Framework is available via the Siemens Web Framework Storybook, which provides:

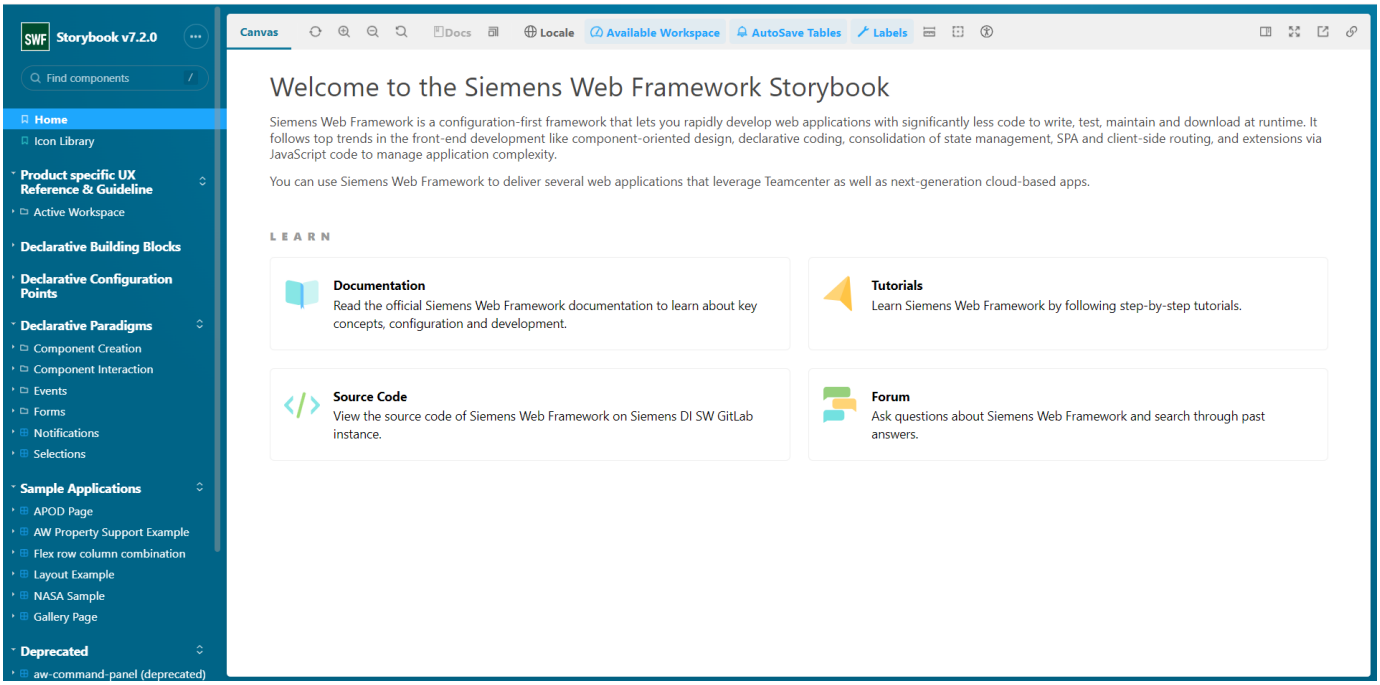
- An extensive collection of rich UI elements that are guaranteed to be always aligned with the latest UX policies.
- Examples and usage of all the public UI elements through interactive examples, user experience guidelines, code snippets and links to API documentation.

Prior to version 7.0 of Siemens Web Framework, a *Showcase* application was used for similar purposes.

A different version of Storybook is released for each version of Siemens Web Framework, and prior to version 7.0, the same was true for the Showcase applications. The most recent Showcase and Storybook versions are available on Support Center at the following links:

- **v6.3**
- **v6.2**
- **v6.1**
- **v6.0**
- **v4.6**
- **v4.5**
- **v4.4**
- **v4.3**
- **v4.1**
- **v3.3**

- v3.2

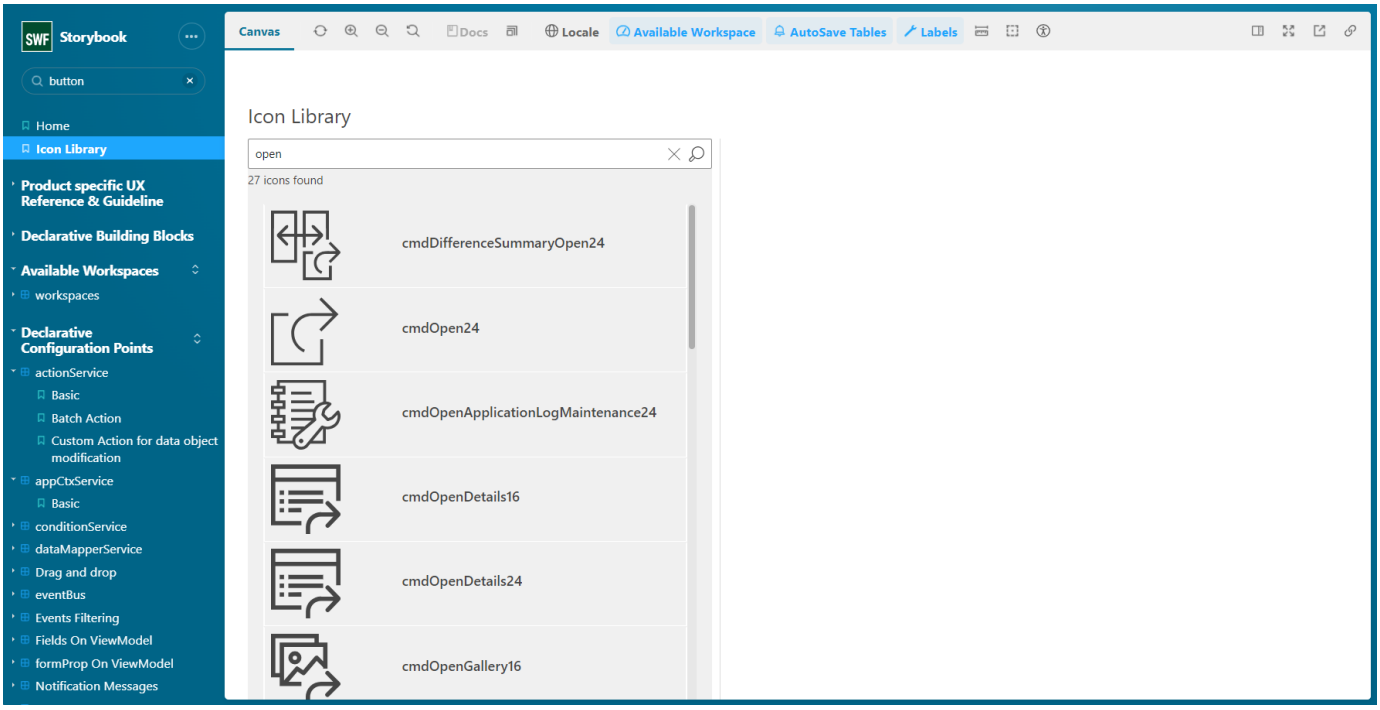



Storybook helps you to create a consistent user interface guaranteeing alignment with latest user experience principles. The following sections give a quick outline of the main functionalities that it provides.

## Icon Library

The icon library page provides a simple but useful interface to search through all icons that are included in Siemens Web Framework.

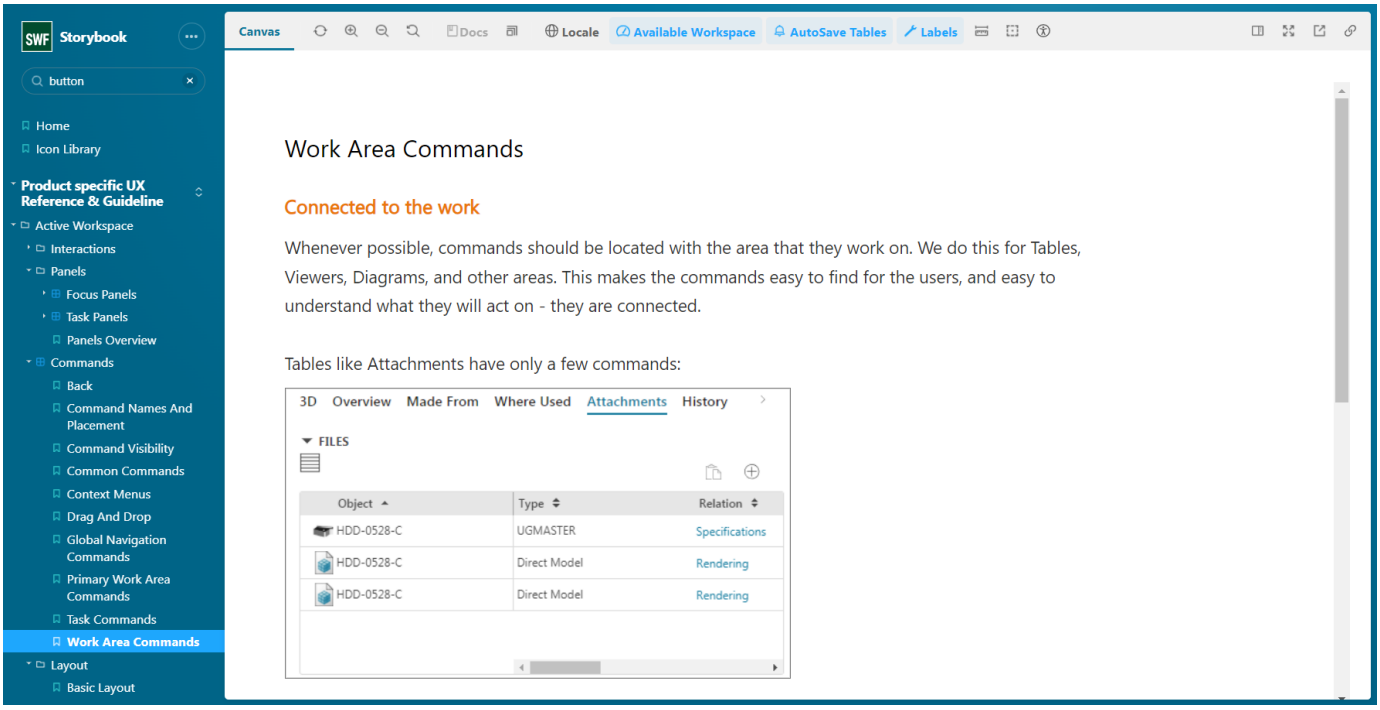

Siemens Web Framework
Unpublished work. © 2025 Siemens




## User Experience Reference Guidelines

The UX guidelines allow you to:

- familiarize yourself with the design principles to be followed while building applications using Siemens Web Framework.
- learn about the recommended layouts and styling, color palette, typography, iconography, etc.

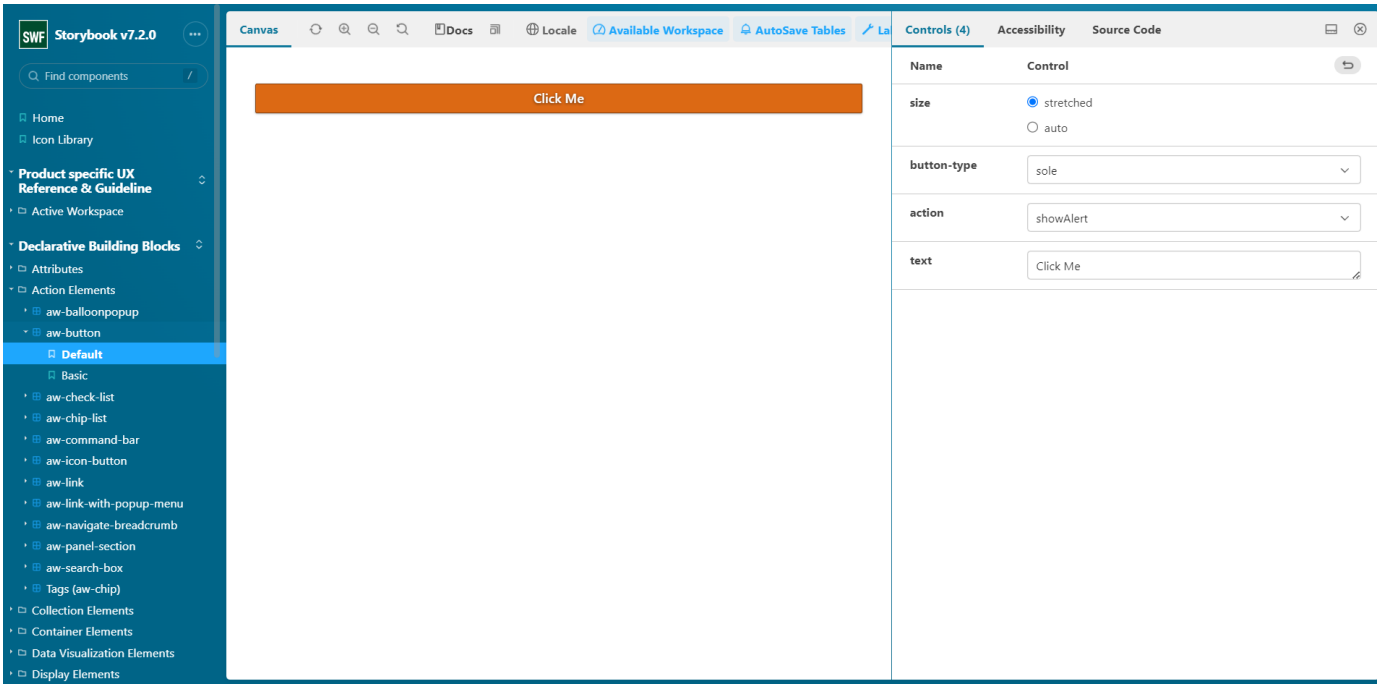



## Declarative Building Blocks

The declarative building blocks section includes:

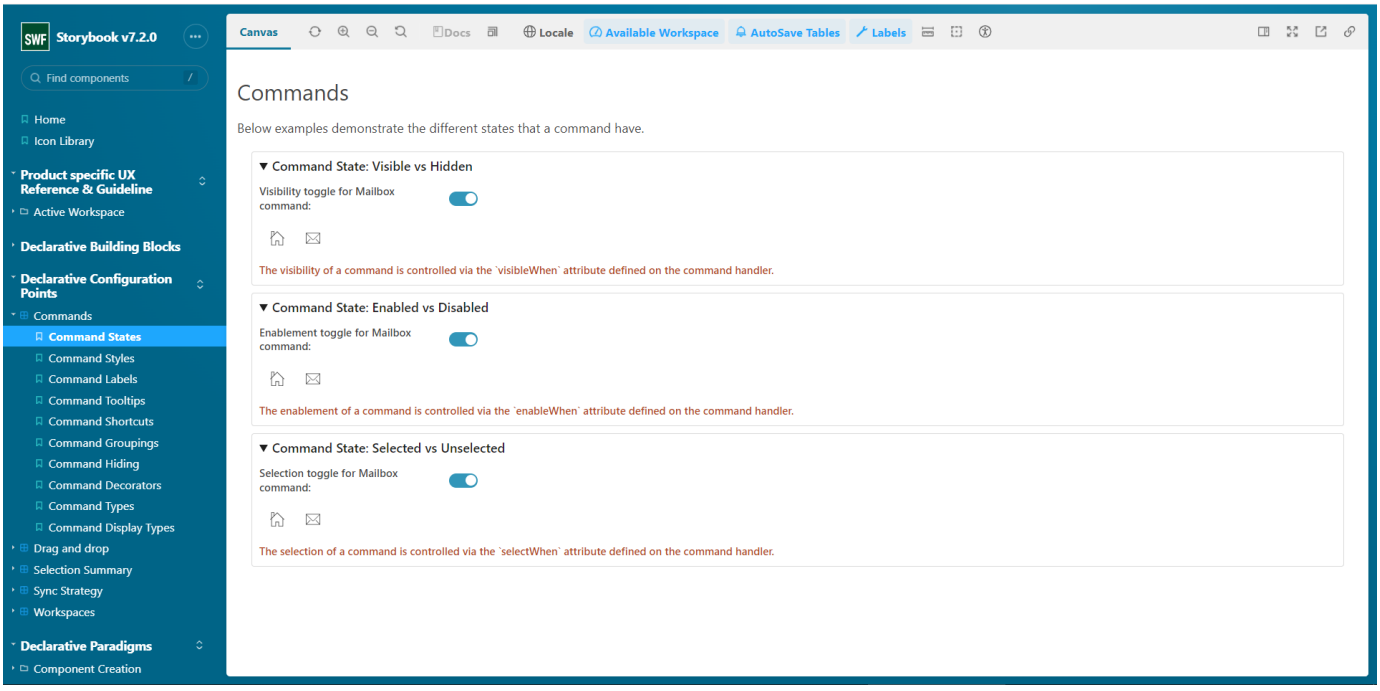
- the complete set of UI elements provided by the framework
- component examples customizable with controls
- interactive examples demonstrating usage of all UI elements
- working code snippets to help developers build applications faster
- some sample applications demonstrating framework capabilities


Siemens Web Framework
Unpublished work. © 2025 Siemens



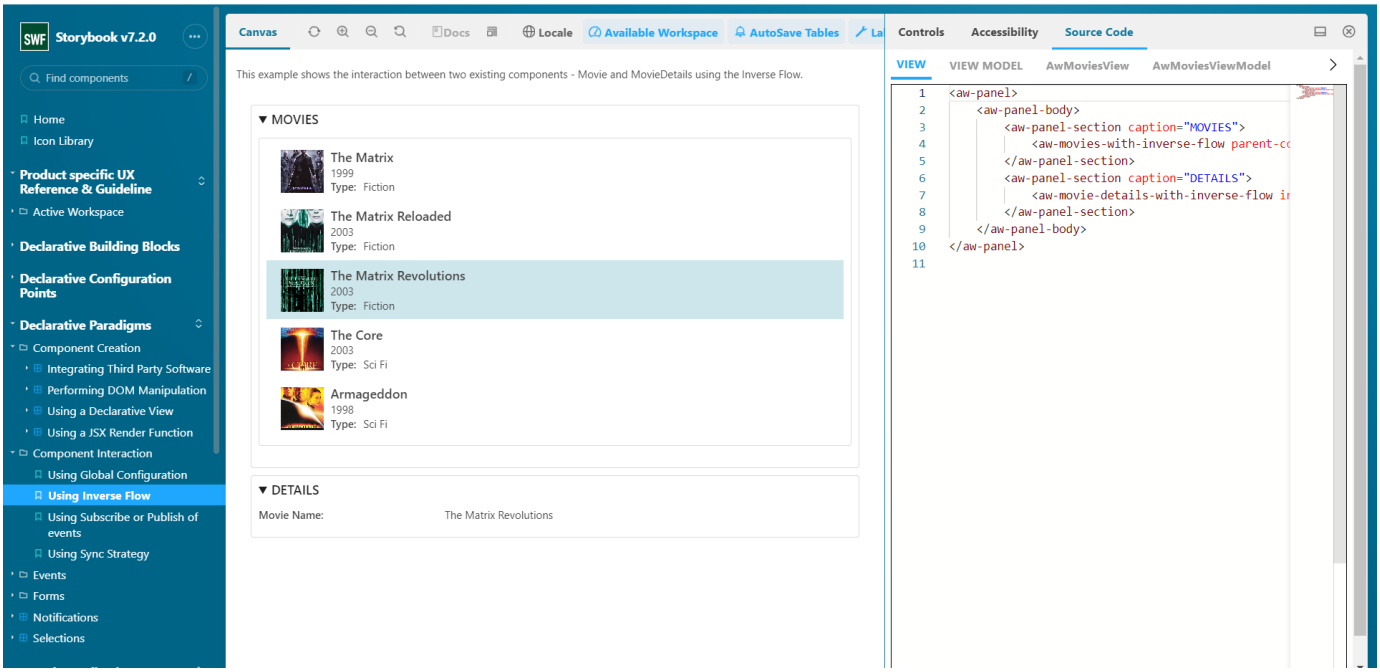

## Declarative Configuration Points

The declarative configuration points section includes dedicated examples for the configuration points provided by Siemens Web Framework.



## Declarative Paradigms

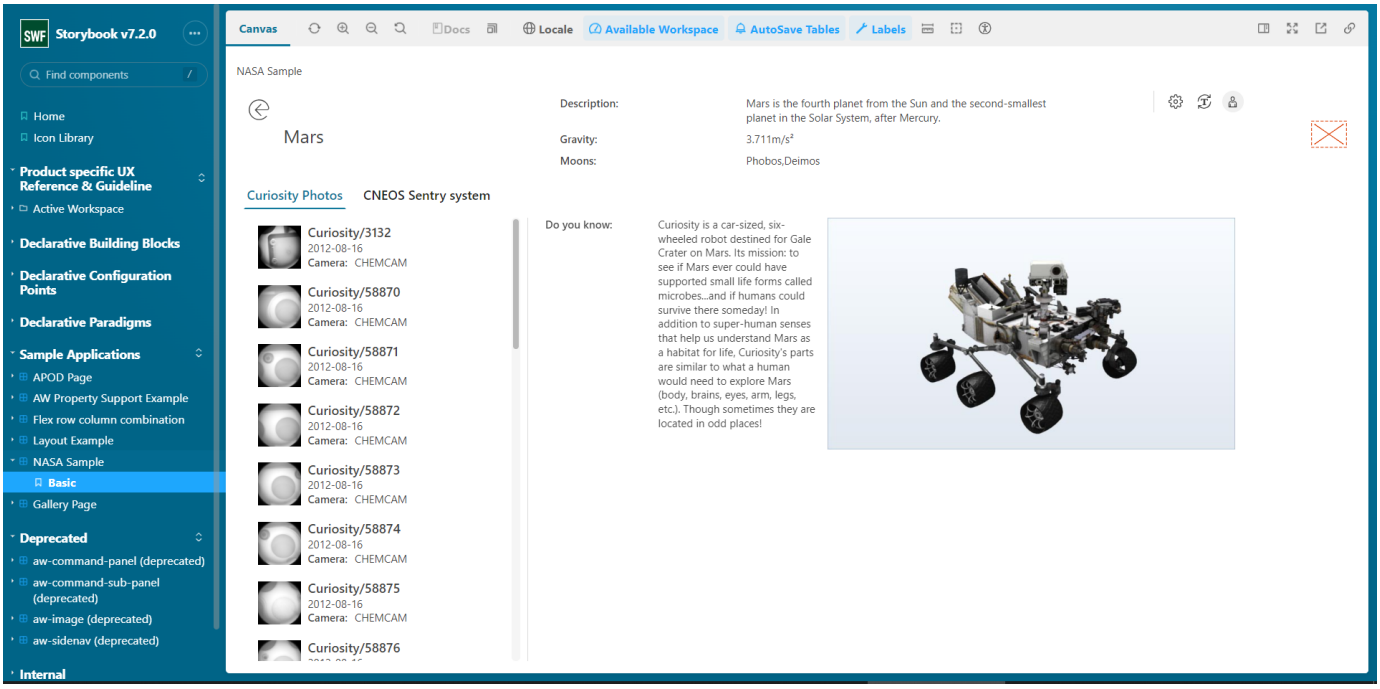
The declarative paradigms section includes dedicated examples to explain the key concepts of declarative framework provided by Siemens Web Framework.




## Sample Applications

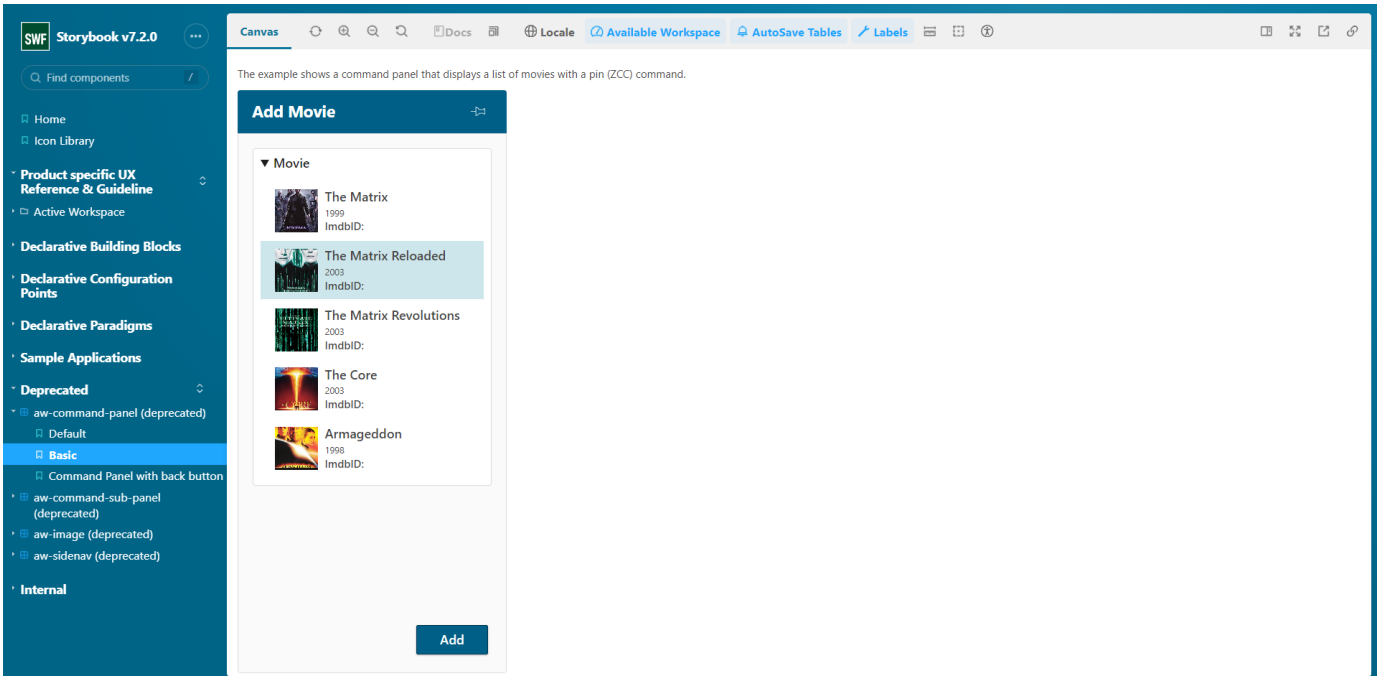
The sample applications section includes dedicated examples to explain the usage of few of the concepts or small applications built using Siemens Web Framework.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Deprecated

The deprecated section includes the documentation of the components which have been deprecated in Siemens Web Framework.



# Prerequisites

Siemens Web Framework is a declarative and configuration-driven framework to build modern, client-side single-page applications.

Although it does not require you to be proficient in web development or in any particular JavaScript framework, the following software knowledge will be beneficial to get the most out of the framework:

- Moderate knowledge of HTML5, CSS3, and JavaScript.
- Basic understanding of Model-View-View Model concepts (especially for versions up to 4.x).
- Basic understanding of Unidirectional Data Flow (for version from 5.x onwards).
- Basic understanding of the Document Object Model (DOM).

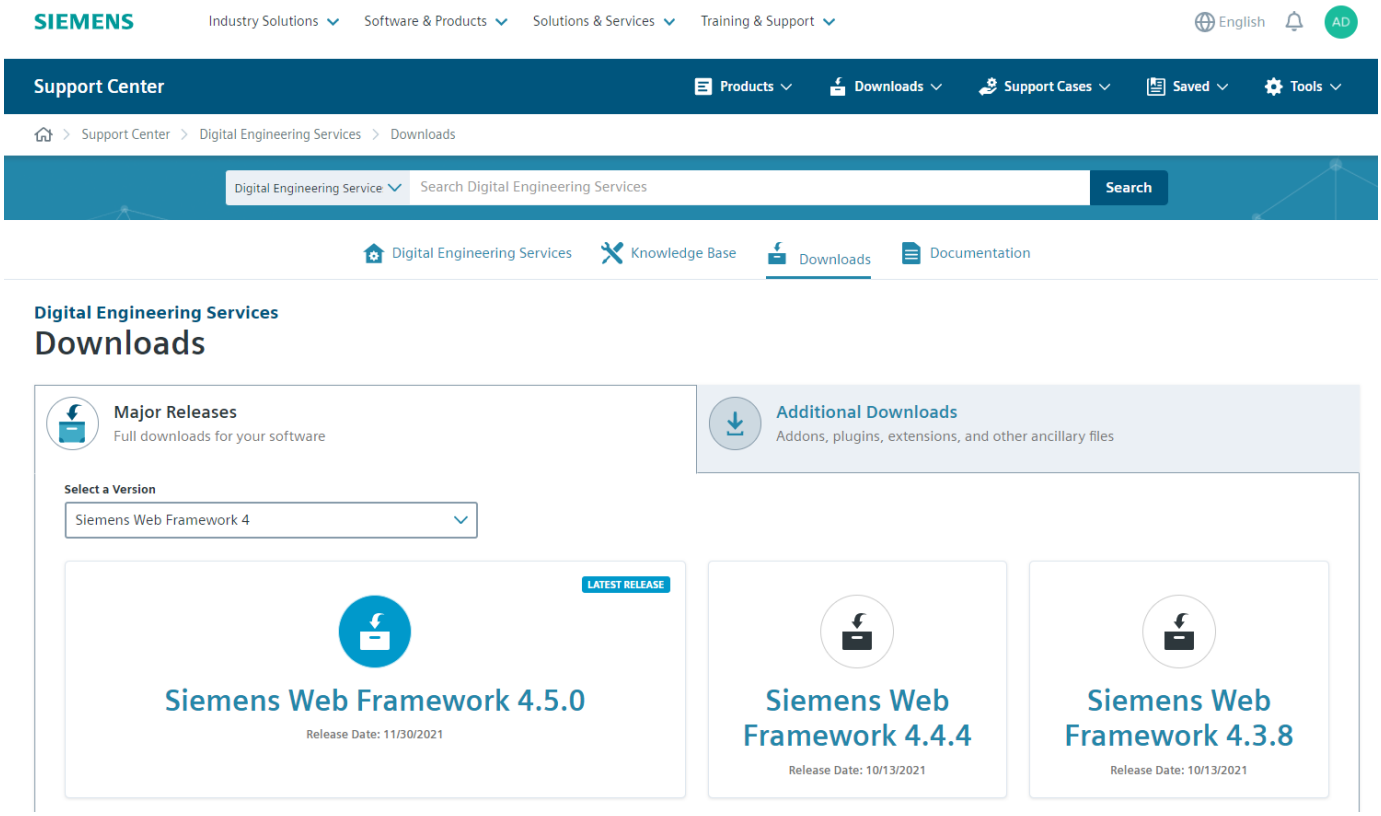
## Software prerequisites

- The current LTS version of **NodeJS**, required to build Siemens Web Framework projects and execute scripts.
- The latest versions of one of the following browsers:
  - **Google Chrome** (recommended)
  - **Microsoft Edge**
  - **Mozilla Firefox**
- A programmer editor or IDE suitable web development. Although choosing your editor largely depends on personal preference, we recommend **Visual Studio Code** for the high level of support for web development languages and for the large ecosystem of dedicated **extensions** that can greatly enhance your development productivity.

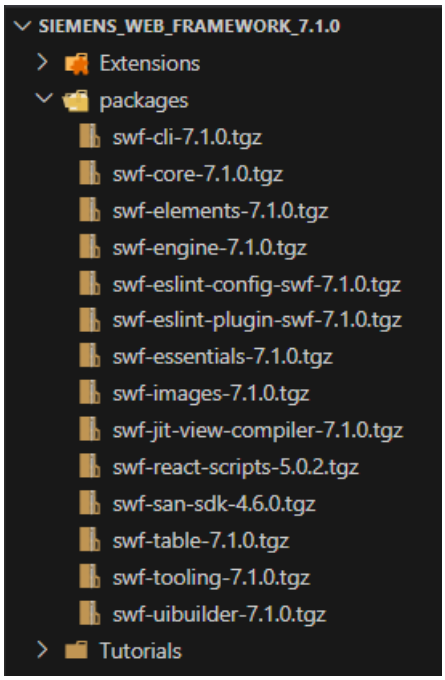
## Downloading and installing Siemens Web Framework

You can download a .zip file containing the source code, tutorials and some tools related to Siemens Web Framework from the **Teamcenter Downloads** section of **Support Center**. Select the desired version from the **Major Releases** section and download the desired release:


Siemens Web Framework
Unpublished work. © 2025 Siemens




After downloading the ZIP file, unzip it locally on your system. The folder structure should look like the following:



Note that the **packages** folder contains the .tgz files corresponding to the various Siemens Web Framework NPM packages.

## Installing the Siemens Web Framework command line interface

First, you should install the the **SWF Command Line Interface** (SWF CLI) to help you manage your Siemens Web Framework projects.

To do so, run the following command:

```
npm i -g ./Siemens_Web_Framework_7.2.0/packages/swf-cli-7.2.0.tgz
```

Note that you have to specify the path to the `swf-cli-<version>.tgz` package within the folder you downloaded and unzipped from Support Center. In this case, the command above assumes that you downloaded version *7.2.0* of Siemens Web Framework.

Then, run a quick check of successful installation using `aw --version` or `swf --version`. Ensure that it returns you version successfully.

**Note**

If you get an error such as *the term is not recognized*, this is probably because you either do not have the package installed globally, @swf/cli is not correctly installed, or its path has not been added to your PATH environment variable. To add a global package link to your PATH environment variable: **In Windows** Ensure that `aw.cmd` or `swf.cmd` exists at `%USERPROFILE%\AppData\Roaming\npm`. Then add this directory to the PATH environment variable and restart the terminal
**In Unix-like operating systems** NPM should handle global packages to be accessibile automatically. If the problem persists, try reinstalling the @swf/cli package.

## Creating a Siemens Web Framework project

Once you installed the SWF CLI, you can create a new Siemens Web Framework project by running one of the following commands in an empty directory on your system:

- `aw quickstart` This will initialize a new project, download all required and optional packages, add a sample page and start the development server immediately. You can read more about this command in the **getting started guide**.
- `aw init` This will just initialize an empty project, without creating any sample page or starting the development server.

In both cases, you will be asked whether you want to install Siemens Web Framework packages from a local folder or not. Answer **Yes** and specify the path to your **packages** folder containing all Siemens Web Framework packages:


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
$ aw quickstart
? Do you want to install SWF packages from a local folder? Yes
? Specify the path to the folder containing .tgz SWF packages ./Siemens_Web_Framework_7.2.0/packages
```

### Next steps

Once you completed the installation of prerequisite software and setup your project to use Siemens Web Framework, follow the **tutorials** and then the **advanced tutorials** to get started.

## Getting started

This page provides a comprehensive overview of all the key concepts of Siemens Web Framework and a walkthrough on how to create a simple but fully-functional example application.

### Purpose and philosophy

Siemens Web Framework (SWF) is a client-side, configuration-driven framework for building web-based Single Page Applications in JavaScript. If you have used a JavaScript framework like React, Angular, or Vue before it will look somewhat similar in some aspects, but very different in others.

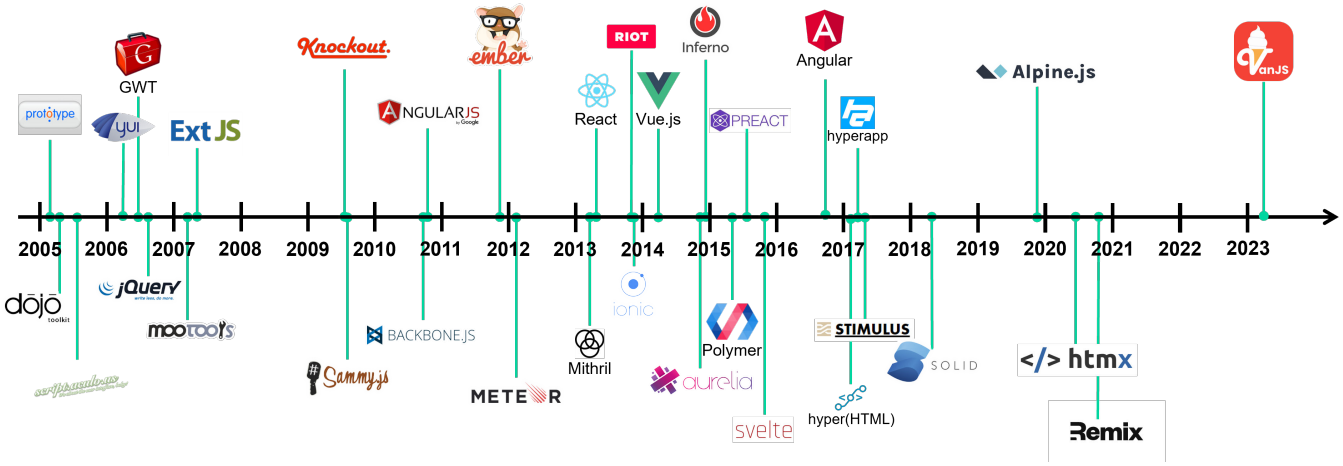
The mission statement of Siemens Web Framework is *to provide a scalable and modular enterprise framework for creating future-proof user interfaces*.

Let's digest this statement a little bit.

Siemens Web Framework aims at being *scalable* so that it can be used to build pretty much any user interface, from the smallest component to the most complicated enterprise application. It also aims to be *modular*, so that you can choose how much of its many functionalities you want to leverage for your project. Additionally, some enterprise applications like Active Workspace also support custom extensions, and Siemens Web Framework truly excels at providing a stable, declarative API to extend virtually any part of the user interface.

Finally, the main focus is on creating *future-proof* user interfaces. *Future-proof* in the sense that the codebase of a user interface should not become obsolete after a few years because of new technologies, instead, it should withstand the passage of time. Writing future-proof code in client-side JavaScript these days is very hard: over the past twenty years, a plethora of client-side JavaScript framework was released, and although the trend is slowing down in recent years, if you write your application using one of the many frameworks available out there, you have no guarantee that your code will not become obsolete because of breaking changes introduced, or because a newer, shinier technology was released.


Unpublished work. © 2025 Siemens




Siemens Web Framework is different: it has no expiration date, and its core API and contracts has remained the same since when it was first released. Sure, there have been some breaking changes and many new features have been introduced over the years, but the fundamental concepts and way to do things have remained the same.

## Why is Siemens Web Framework different from traditional JavaScript frameworks?

Siemens Web Framework can be seen as an abstraction layer on top of a traditional JavaScript framework. When it was first introduced, it leveraged AngularJS internally to do all the heavy lifting. Sadly though, as it happens, **AngularJS** reached its end of life at the end of 2021, and so Siemens Web Framework transitioned to use **React** instead.

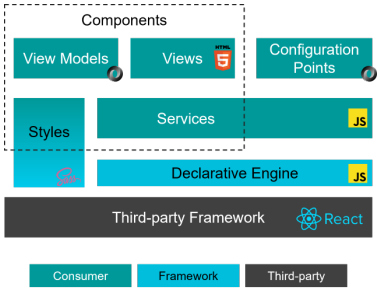
Think about this for a minute. If you ever used AngularJS and you are familiar with React, think about migrating a very big codebase from using AngularJS to React... what would it take? It would probably be simpler to rewrite your application from scratch.

Active Workspace, the web client of Teamcenter, went through this transition. But once Siemens Web Framework transitioned internally from AngularJS to React, it was significantly easier to adapt the codebase, and the impact of such transition was contained.

## High-level architecture

The following diagram shows the high-level architecture of Siemens Web Framework:


Siemens Web Framework
Unpublished work. © 2025 Siemens




At its core, Siemens Web Framework relies on a mainstream *third-party framework*. In the past it used to be AngularJS, and now since version 5.0 it is React. Right above the third-party framework, a *declarative engine* is implemented to decouple the public Siemens Web Framework APIs from the third-party framework. This is the only part of Siemens Web Framework that includes React-specific code, and it is very small.

For what concerns styling, styles are managed using **Sass** Siemens Web Framework provides a rich set of classes that fully implements the Siemens DI SW UX Design System. Consumers can also implement their own styles and even develop their own themes using Sass.

Above the *declarative engine* Siemens Web Framework provides a rich set of JavaScript *services* some of which are public and can be used by consumers, and nearly 100 public *component*, that implement most of the UI elements mentioned by DI SW UX policies and a few extra ones. In Siemens Web Framework, a *component* is typically made up by:

- a *view model* file, describing its public contract, behavior, and internal state
- a *view* file, essentially an HTML-based template able to access view model data (note that as an alternative, the component template can be implemented in a more codeful way using **JSX** syntax)
- a *service* file containing a JavaScript module with any JavaScript function needed by the component

All Siemens Web Framework components are implemented in this way, and you can (and wil!) also create your own components in this way, without ever accessing React-specific APIs.

## Creating your first project

Let's get started! The best way to get a taste of Siemens Web Framework is to install its powerful **Command Line Interface** (SWF CLI).

1. Download and install the current LTS version of **NodeJS**.
2. Download a .zip file containing the source code and tutorials of Siemens Web Framework from the **Teamcenter Downloads** section of **Support Center**. Select the desired version from the **Major Releases** section and download the desired release.
3. Unzip the downloaded Siemens Web Framework .zip file and unzip it to a local folder. The newly-created folder will contain different subfolders, one of which is **packages**, which contains all the Siemens Web Framework NPM packages.

4. Install the SWF command line interface by running the following command (make sure you point to the proper NPM package):

```
npm i -g ./Siemens_Web_Framework_7.2.0/packages/swf-cli-7.2.0.tgz
```

You will now be able to use the `aw` command to work on your Siemens Web Framework projects.

One the CLI is installed, create an empty directory called **starter**, open a command prompt and execute the following command:

```
aw quickstart
```

When asked the following question:

```
Do you want to install SWF packages from a local folder?
```

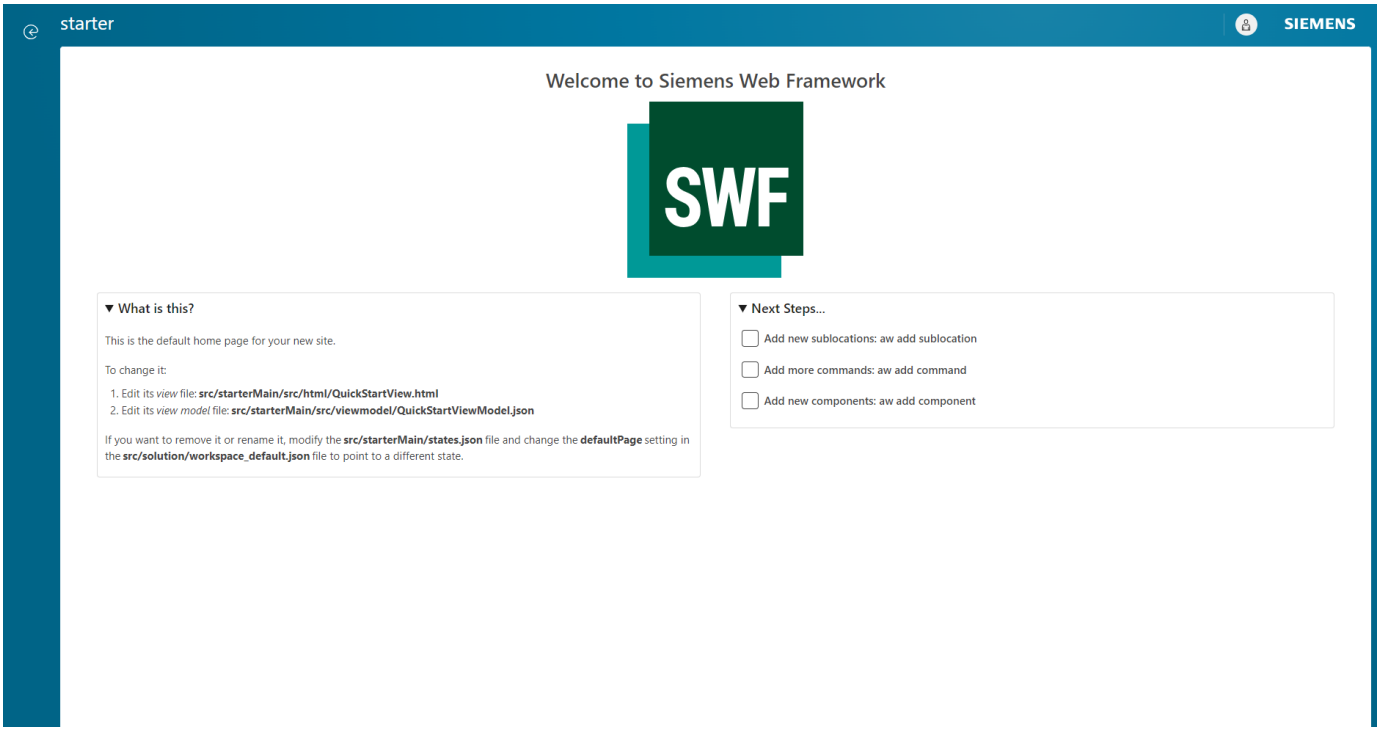
Select **Yes** and specify the folder containing the SWF .tgz packages you downloaded from Support Center.

This command will:

1. create the standard project structure for a new Siemens Web Framework project
2. install all NPM packages needed
3. add the necessary configuration to create a sample start page
4. run the project in development mode locally, on **http://localhost:3000**
5. open your default browser on **http://localhost:3000**

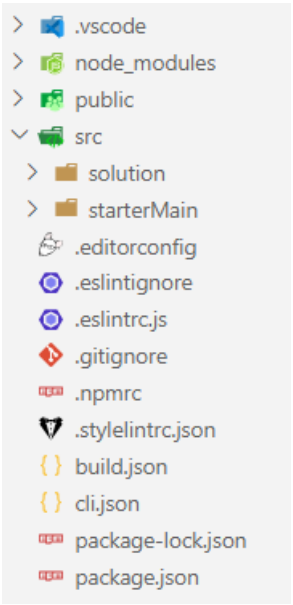
Once the quickstart command completes, you should see a welcome page in your browser similar to the following:


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Project structure

Let's have a look at the folders and files that were generated automatically by the `aw quickstart` command:




This is how a Siemens Web Framework project looks like. The top level includes some files and folder that would look fairly familiar to other JavaScript-based project folders, such as the following:

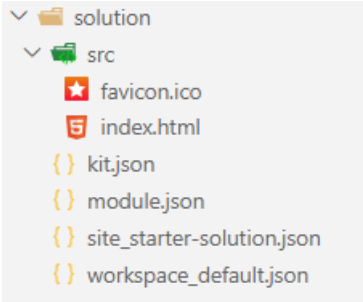
- The `.vscode` folder contains Visual Studio Code settings specific to Siemens Web Framework. Although it is not required that you use Visual Studio Code for Siemens Web Framework applications, you probably should.
- The `node_modules` folder contains all NPM package dependencies.
- The `src` folder contains the source files of your application.

Also, a few fairly standard configuration files are generated automatically to manage integration with popular tools like **ESLint**. The following two files are specific to Siemens Web Framework:

- `build.json` tells Siemens Web Framework build tools where to look for source files.
- `cli.json` is used internally by the CLI to store local project preferences like the default kit, module, or view model (more on this later on).

## The solution folder

The `src/solution` folder contains files related to your Siemens Web Framework **solution**, which is basically the configuration of your web application.




Don't worry too much about these files for now, just remember that:

- A solution is defined by a `kit.json` file, and typically you'll only use one kit per project.
- A **kit** can have multiple **modules**, each in a separate folder and identified by a `module.json` file.
- A solution can contain multiple **sites**, but typically you'll only use one.
- A solution can have multiple workspaces. Again, at first, you'll probably need just one.

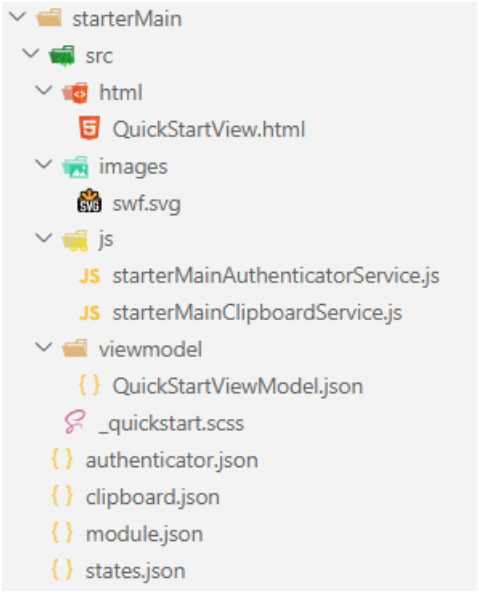
Another handy thing to remember is that if you want to change the home page for your site, you can do so in the workspace definition file (`workspace_default.json`).

## The main module

Your solution can have multiple modules, each in a specific folder. The `aw quickstart` command creates one module automatically for you, which contains all the files that are neede to render the quick start example page that is displayed when your application starts up.


Siemens Web Framework
Unpublished work. © 2025 Siemens

The folder structure of a module is very important, and it is always the same for all modules:




At top level, the following files are present:

- `authenticator.json` is used to configure authentication for your site. A default authenticator is required for every site, even if you don't plan to use any authentication. Luckily, it is generated automatically for you by the `aw quickstart` command.
- `clipboard.json` is used to configure clipboard support. This is not mandatory for a site to run, but again, a default clipboard configuration is generated by the `aw quickstart` command. You only need one clipboard configuration per site.
- `module.json` contains the definition of your module.
- `states.json` contains the definition of your **states**, or routes, which define all the pages of your application.
- `_quickstart.scss` contains some custom styles used for the quick start page. If you want to add custom styles, just create `.scss` files at top level within the `src` folder of your module, and make sure they start with an underscore. Siemens Web Framework uses **Sass** for managing styling.

The `src` folder contains the actual source files used, in this case, by the quick start page, and it is organized in the following sub folders:

- `html` this folder contains the **view** files, that are used to define the look and feel of portions of the user interface (components).
- `images` this folder contains any image that may be needed by your application. The quick start page contains a big Siemens Web Framework logo, and that's where it is stored.
- `js` this folder contains JavaScript **services** used to provide codeful extensions to your application. The `aw quickstart` command generates two files that are the default services used for the authenticator and clipboard.

- `viewmodel` this folder contains **view model** files, that are used to define the contract, internal state, and behavior of portions of the user interface (components).

## How the quick start page is rendered

*That's all very well but... how can I create a new page?* you ask, and that's a fair question.

Pages, or better *states* or *routes* as they are typically referred to in Single-Page Applications, are defined in `states.json` files. If you have multiple modules in your project, each module can have its own `states.json` file containing its own state definitions. At run time, all the state definitions from all modules are merged together (you'll see that this pattern applies to a lot of the Siemens Web Framework configuration files).

The `aw quickstart` command generated a `states.json` containing the following code:

```
{
  "quickstart": {
    "url": "/quickstart",
    "view": "QuickStart",
    "data": {
      "priority": 1,
      "label": "Quick Start",
      "nameToken": "starter:quickstart",
      "clientScopeURI": ""
    },
    "parent": "home"
  },
  "home": {
    "abstract": true,
    "data": {
      "headerTitle": "starter",
      "browserSubTitle": "Home"
    },
    "view": "commonLocation",
    "parent": "root"
  }
}
```

Here, there are *two* state definitions. Why two, if there's only one page? Well, the `home` state is flagged as `abstract`, which basically means it cannot be accessed directly, only its child states can. This is needed, for now, only to define the title displayed on the top of the page (starter) and the title to display in the browser tab.

The `home` state loads a view called `commonLocation`. This view isn't in your project, but it is defined within Siemens Web Framework and it's one of the standard **location** components that you can use.

The `quickstart` state is more interesting, as it is used to load the actual quick start page by specifying the `QuickStart` view, which loads the `QuickStart` component...


Siemens Web Framework
Unpublished work. © 2025 Siemens

## What is a component?

...but what is a *component*, exactly? We mentioned components casually so far, but now it's time to introduce them properly, as they are one of the core building blocks of Siemens Web Framework.

A component identifies a reusable piece of your user interface. That's pretty much as generic as it gets, as far as definition goes. But that's what it is, really. You want to create a page? You create a component. You want split the page in reusable chunks? You create components. You want to add a button on your page? You reference the `aw-button` component.

(Almost) everything is a component!

More concretely, in Siemens Web Framework a component is made up of:

- a **view model** file, that is mandatory to define its metadata, its contract, behavior and internal state.
- a **view** file, which is essentially an HTML file with some special placeholders wrapped in double curly braces (if you used AngularJS, Angular, or Vue, you know what it looks like already). Views are used to define how a component looks like, but you can also use a **render function** and JSX syntax instead, if you are so inclined (but we do recommend using view files as much as possible).
- optionally, a JavaScript **service** file, if you want or need to implement some custom behavior in a codeful way.

Back to our app, the `aw quickstart` created the `QuickStart` component for us, which is implemented by the following files:

- `src/html/QuickStartView.html` its view file.
- `src/viewmodel/QuickStartViewModel.json` its view model file.

They are fairly simple to understand, you should examine them to get a feel of how everything works.

Remember that naming conventions are very important in Siemens Web Framework, and in particular:

- View files must be placed in the `src/html` folder of a module, their name must be in PascalCase, be made up of two or more words, and end with `View.html`.
- View model files must be placed in the `src/viewmodel` folder of a module, their name must be in PascalCase, be made up of two or more words, and end with `ViewModel.json`.

To pair a view with its view model, their name must be the same, so that's why to define a *QuickStart* component there's a *QuickStartView.html* and a *QuickStartViewModel.json* file.

## Adding a new sub location

The QuickStart component helps giving you an overview of the basics of Siemens Web Framework, but it doesn't do much: it's basically just a static page.


Unpublished work. © 2025 Siemens

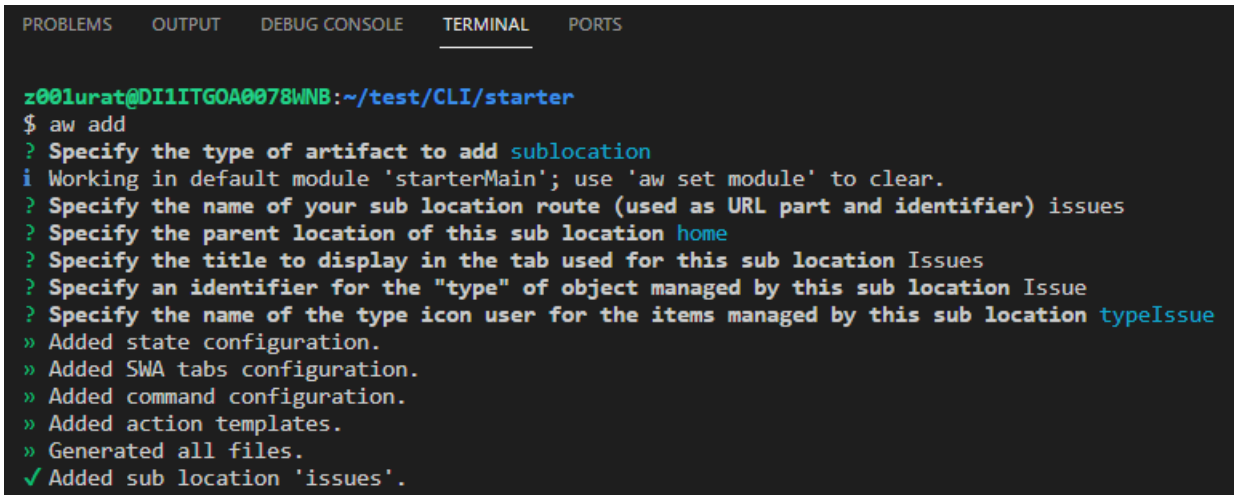
Let's create something more interesting. In your project folder, execute the following command:

```
aw add
```

Most of aw's top-level command require additional arguments or flags, but if you don't specify any of the required ones, you'll be prompted to provide them instead.

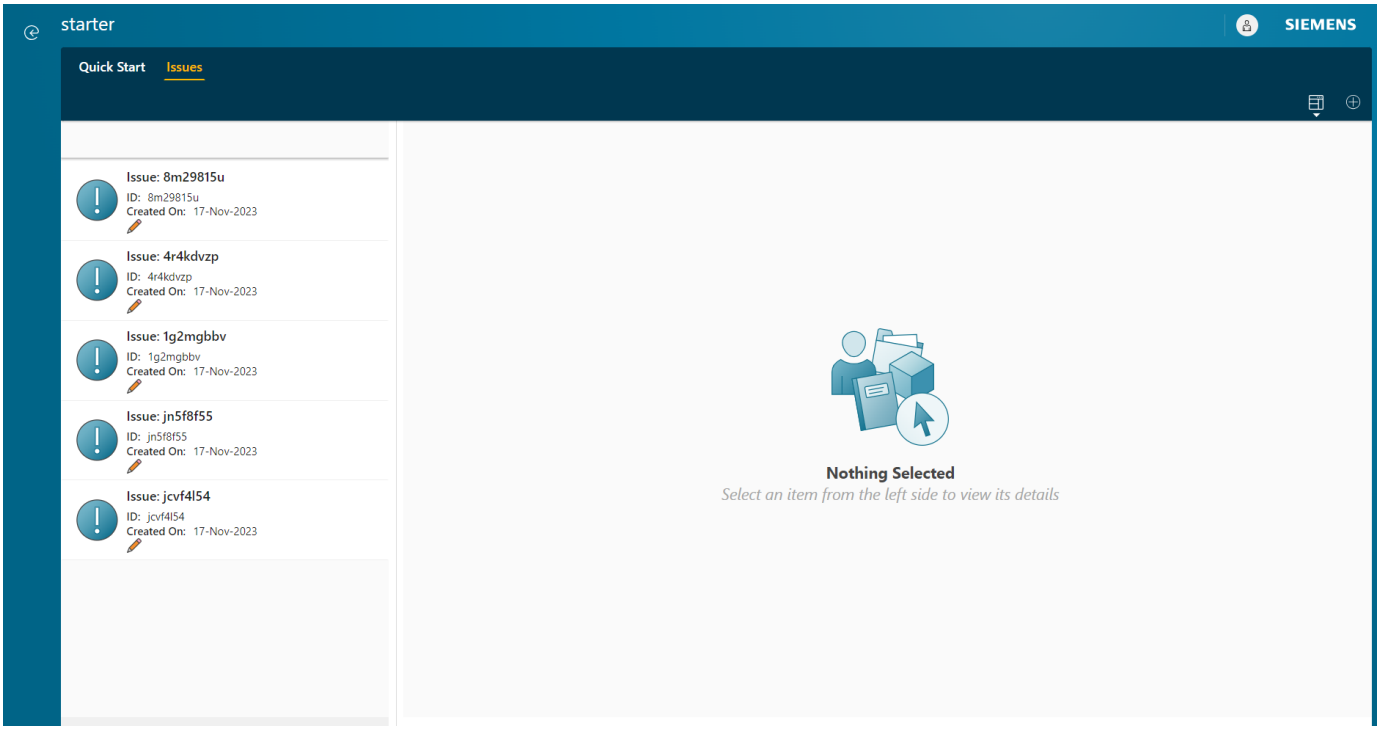
`aw add` by itself is not really enough to create anything, and the first thing you are prompted for is *what* declarative artifact you want to create.

Specify **sublocation** and continue following the prompts, as follows:




This will create a new sub location called **Issues**, that will be accessible via its corresponding sub location tab:


Siemens Web Framework
Unpublished work. © 2025 Siemens




Well, that definitely did *something*! We now have what looks like a simple but fully-functional master-detail page to manage... issues. Feel free to play around: you'll see that you can view issue details, add new issues, edit them, pin/unpin them, even delete them! Obviously it's all randomly-generated sample data (and it's all stored in localStorage), but it will be enough to get a *feel* of how everything works.

## Understanding view modes

How is data loaded and displayed in the new Issues sub location? To understand that, let's start from the state definition of the sub location itself.

The URL (or better, the URL *fragment*, after the #) determines the application route (state) to load. For the Issues sub location the fragment is `/issues`. If you open the `states.json` file in the `src` directory of the main module, you'll see that it indeed defines a sub location called Issues with a `url` property set to `/issues`:

```
{
    "issues": {
    "url": "/issues",
    "data": {
      "label": "Issues",
      "type": "Issue",
      "nameToken": "starter:issuesSubLocation",
      "clientScopeURI": "",
      "viewBase": "Issues",
      "viewModes": {
        "SummaryView": {
          "primaryWorkArea": "list",
```

```
          "secondaryWorkArea": true
        },
        "ListView": {
          "primaryWorkArea": "list",
          "secondaryWorkArea": false
        }
      }
    },
    "params": {
      "s_uid": null
    },
    "parent": "home",
    "view": "AwStandardSublocation"
  }
}
```

If you compare this state definition to the QuickStart one discussed earlier on, you'll notice that there's no `view` property. Instead, the `viewBase` and `viewModes` properties are used to determine which components to load in the master (primary work area, or simply PWA) and details (secondary work area, or simply SWA) portions of the page.

This sub location supports two view modes:

- `SummaryView`, which is the default one showing both the primary and secondary work area.
- `ListView`, which only shows the list in the primary work area but no secondary work area.

In the UI, you can switch view modes using the command button on the top-left of the primary work area.

The component to load in each portion is determined by the value of the `viewBase` property plus a suffix that is one of the following:

- for the PWA, the capitalized value of the `primaryWorkArea` property of the view mode definition (`List` in this case)
- `Details` for the SWA.

In this case, the view and view models that are loaded are the following:

- PWA: `IssuesListView.html` and `IssueListViewModel.json`.
- SWA: `IssuesDetailsView.html` and `IssueDetailsViewModel.json`.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Loading data in the primary work area

Let's have a closer look at the `src/html/IssuesListView.html` file. It only contains the following code:

```
<aw-panel>
  <aw-list dataprovider="data.dataProviders.items">
    <aw-default-cell vmo="item"></aw-default-cell>
  </aw-list>
</aw-panel>
```

This is enough to specify that this view contains a panel (**aw-panel**) with a list (**aw-list**) displaying cells (**aw-default-cell**).

How is the data loaded into the list then? Well, that's the **data provider**'s job.

This view is referencing the `items` data provider, which is defined in the `data.dataProviders` section of the view model associated to this view. If you look in `src/viewmodel/IssuesListViewModel.json`, that definitions looks like this:

```
{
    "dataProviders": {
        "items": {
            "action": "getItems",
            "response": "{{data.items}}",
            "totalFound": "{{data.items.length}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            }
        }
    }
}
```

This is a typical data provider definition. It has an action that is used to fetch the data, and it specifies where the response data and the number of items found will be stored in the view model.

Let's have a look at the `getItems` action:

```
{
    "getItems": {
        "actionType": "SampleDataGet",
        "inputData": {
            "params": {
                "generate": 5,
                "type": "Issue",
                "icon": "typeIssue"
            }
```

```
        },
        "dataParsers": [
            {
                "id": "itemsParser",
                "responseObjs": "data"
            }
        ],
        "outputData": {
            "items": "data"
        }
    }
}
```

**Actions** are the fundamental way to *do stuff* in Siemens Web Framework. This action is of a custom type called `SampleDataGet`, it takes some input parameters, and it stores data in the `data.items` section of the view model, specified via `outputData`.

There are different types of actions available out-of-the-box in Siemens Web Framework, or you can define your own for enhanced code reusability, like in this case. The definition of this action is stored in the `actionTemplateDefs.json` file. If you inspect it, you will notice that ultimately it references some JavaScript code to execute (the method `get` exported by the `js/sampleData` service, which has been added to the `src/js` folder of your module).

The `dataParsers` property references an `itemsParser` **data parser** which is defined in the same view model. Data parsers are used to basically map an array of objects (like the one returned by executing this action) to something else that Siemens Web Framework widgets understand. In this case, an array of simple objects representing issue data is transformed in an array of **ViewModelObjects**, each of which is displayed by a cell of the list.

The last piece to make sure data is displayed is the following hook definition in the `IssuesListViewModel`:

```
{
    "lifecycleHooks": {
        "onMount": "initItemsDataProvider"
    }
}
```

This simple configuration says that when the `IssuesList` component is first loaded on the page (mounted) the `initItemsDataProvider` action must be executed. This is another action defined in the view model:

```
{
    "initItemsDataProvider": {
        "actionType": "dataProvider",
        "method": "items"
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

This is a `dataProvider` action that is used to initialize the `items` data provider we discussed earlier. Once it is initialize, this data provider will call the `getItems` action to fetch the issues, which will then be displayed by the list.

This may feel a little bit complicated, but if you understand this, you will be able to create similar PWA views, as they all work in the same way really.

## Displaying selected data in the secondary work area

But how is data displayed in the SWA? Basically, if you try selecting and deselecting issues in the PWA, the corresponding details will be displayed in the SWA, like a proper master/detail page. If nothing is selected, a standard empty page will be displayed.

This mechanism is so common that it can be configured easily using a specific configuration file called `secondaryWorkareaTabs.json`. If you check in your module, it contains the following code:

```
[
  {
    "name": "Overview",
    "priority": 50,
    "view": "IssuesDetails",
    "visibleWhen": "selection[0].props.type.dbValue === 'Issue' &&
selection.length === 1"
  },
  {
    "name": "Overview",
    "priority": 100,
    "view": "Empty",
    "visibleWhen": "selection.length === 0"
  }
]
```

Try to read this code and compare it with the behavior you observed in the interface. This file contains an array of objects that determine which component will be loaded in the SWA, depending on whether the condition specified in the `visibleWhen` condition is satisfied or not.

We can now understand that if there's nothing selected, the `Empty` component will be loaded, otherwise if there's one item selected (and that item is an issue) then the `IssuesDetails` component will be loaded.

The `src/html/EmptyView.html` and `src/viewmodel/EmptyViewModel.json` are very simple and basically contain some static data to generate the standardized empty page displayed when nothing is selected.

The `src/html/IssuesDetailsView.html` file is more interesting:

```
<aw-panel>
  <aw-panel-section>
    <aw-widget prop="fields.item.title"></aw-widget>
```


Unpublished work. © 2025 Siemens

```
    <aw-widget prop="fields.item.created"></aw-widget>
    <aw-widget prop="fields.item.updated"></aw-widget>
    <aw-widget prop="fields.item.pinned"></aw-widget>
    <aw-widget prop="fields.item.description"></aw-widget>
  </aw-panel-section>
</aw-panel>
```

This is basically a simple form containing the standard properties of an issue, rendered through the **aw-widget** component.

In this case, to populate the data, there's no need for a data provider. Instead, it is sufficient to define an action to retrieve the data related to the selected item from the backend:

```
{
  "getSelectedItem": {
      "actionType": "SampleDataGet",
      "inputData": {
        "params": {
          "id": "{{subPanelContext.selection[0].identifier}}",
          "type": "Issue"
        }
      },
      "dataParsers": [
        {
          "id": "itemParser",
          "responseObjs": "data"
        }
      ],
      "outputData": {
        "item": "data"
      }
    }
}
```

...define a data parser to transform each item property in a **ViewModelProperty** object:

```
{
  "itemParser": {
    "identifier": "{{response.id}}",
    "props": {
      "id": {
        "name": "id",
        "displayName": "{{i18n.id}}",
        "value": "{{response.id}}",
        "displayValue": "{{response.id}}",
        "isEditable": false,
        "type": "STRING"
      },
      "title": {
        "name": "title",
        "displayName": "{{i18n.title}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "value": "{{response.title}}",
        "displayValue": "{{response.title}}",
        "isEditable": "{{ctx.editInProgress}}",
        "renderingHint": "textbox",
        "isRequired": true,
        "type": "STRING"
      },
      "description": {
        "name": "description",
        "displayName": "{{i18n.description}}",
        "labelPosition": "PROPERTY_LABEL_AT_TOP",
        "value": "{{response.description}}",
        "displayValue": "{{response.description}}",
        "isEditable": "{{ctx.editInProgress}}",
        "isRequired": true,
        "type": "STRING"
      },
      "pinned": {
        "name": "pinned",
        "displayName": "{{i18n.pinned}}",
        "value": "{{response.pinned}}",
        "displayValue": "{{response.pinned}}",
        "isEditable": false,
        "isRequired": true,
        "type": "BOOLEAN"
      },
      "created": {
        "name": "created",
        "displayName": "{{i18n.createdOn}}",
        "value": "{{response.created}}",
        "displayValue": "{{response.created}}",
        "isEditable": false,
        "isRequired": true,
        "type": "DATE"
      },
      "updated": {
        "name": "updated",
        "displayName": "{{i18n.updatedOn}}",
        "value": "{{response.updated}}",
        "displayValue": "{{response.updated}}",
        "isEditable": false,
        "isRequired": true,
        "type": "DATE"
      }
    }
  }
}
```

...and make sure this action gets executed when the selection changes and when the currently-selected item is being edited (more on editing later), by using an `onUpdate` lifecycle hook:

```
{
  "onUpdate": [
      {
```

```
        "action": "getSelectedItem",
        "observers": [
          "viewModel.subPanelContext.selection",
          "viewModel.ctx.editInProgress"
        ]
      }
  ]
}
```

## Adding new items

The `aw add sublocation` command automatically creates an *Add* command that can be used to create new items using a form displayed in a side panel. **Commands** are one of the most important artifacts of Siemens Web Framework, as they provide a way to define different types of **actions** to perform in your application.

A command is typically typically rendered as a clickable icon and may display a label next to it. Commands are placed in command bars that can be vertical or horizontal.

## Defining the *Add* command

Commands are defined in `commandsViewModel.json` files placed within a module folder. If you examine the structure of this file, it is divided into the following main sections:

- `commands`, containing the basic command metadata like title and icon to display.
- `commandHandlers`, containing information on which action to execute and when the command is visible, active, etc.
- `commandPlacements`, containing information on where the command is displayed within the UI.
- `conditions`, expressions typically used to determine when a command is visible, active, enabled etc., referenced in command handlers.
- `actions`, the actions that are executed by commands to perform different tasks.
- `onEvent`, event subscriptions that may trigger the execution of actions.
- `messages`, alerts, notifications and confirmation messages to display when a command is executed.
- `i18n`, references to localized strings, stored in `.json` files within the `src/i18n` folder of a module.

**Tip**

The following sections explain how a command is defined in a `commandsViewModel.json` file. Although you can create new commands manually by modifying this file, you can also use the CLI to generate most of the needed JSON code by running `aw add command` and following the prompts. Note that if you use the CLI you will also be able to search and select icons and placements from all the available ones.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Specifying the command title and icon

Specifying the title and icon of our *Add* command is straightforward enough. If you check the `commandsViewModel.json` file within our module, you will find the following code that specifies which icon to display and the command title:

```
{
  "commands": {
    "addIssueCmd": {
      "iconId": "cmdAdd",
      "title": "{{i18n.add}}"
    }
  }
}
```

The important thing to note here is that the title is configured using an expression pointing to the `add` localization string within the `commandsViewModel.json` file:

```
{
  "i18n": {
    "add": [
      "sampleDataMessages"
    ]
  }
}
```

This localization entry basically says that the text to display in the `add` string is stored in the `src/i18n/sampleDataMessages.json` file. It is a best practice to move all text to localization files to manage translations more efficiently, and also decouple strings displayed in the UI from the rest of the code.

## Configuring command placement

The placement of the command is specified in the `commandPlacements` section, like this:

```
{
  "commandPlacements": {
    "addIssueCmdPlacementWorkarea": {
      "id": "addIssueCmd",
      "uiAnchor": "aw_workarea_toolbar",
      "priority": 10
    }
  }
}
```

In this case, the command will be placed in a command bar called `aw_workarea_toolbar` (the horizontal command bar at the top of our sub location), and it will have a priority of 10. The priority is used to determine the ordering of commands within a particular command bar.

## Configuring an action to be executed via a handler

To be able to do something, a command needs a handler specifying when the command is active and visible, and the action to execute:

```
{
  "commandHandlers": {
    "addIssueCmdHandler": {
      "id": "addIssueCmd",
      "action": "addIssue",
      "activeWhen": {
        "condition": "conditions.inIssuesSubLocation"
      },
      "visibleWhen": {
        "condition": "conditions.inIssuesSubLocation"
      }
    }
  }
}
```

Note how the conditions used are the same, because the command is active whenever it is visible. The `inIssuesSublocation` condition (defined in the `conditions` section of the `commandsViewModel.json`) is used to check that we are currently in the Issues sub location.

The `addIssue` action is defined in the `actions` section, as follows:

```
{
"actions": {
    "addIssue": {
      "actionType": "dialog",
      "inputData": {
        "options": {
          "view": "AddIssueCmd",
          "caption": "{{i18n.add}}",
          "parent": ".aw-layout-workareaMain",
          "placement": "right",
          "width": "MEDIUM",
          "height": "FULL",
          "subPanelContext": "{{commandContext}}"
        }
      }
    }
  }
}
```

This is a `dialog` action containing a few options, and it is used to render a particular component (`AddIssueCmd`) within a right side panel.


Siemens Web Framework
Unpublished work. © 2025 Siemens

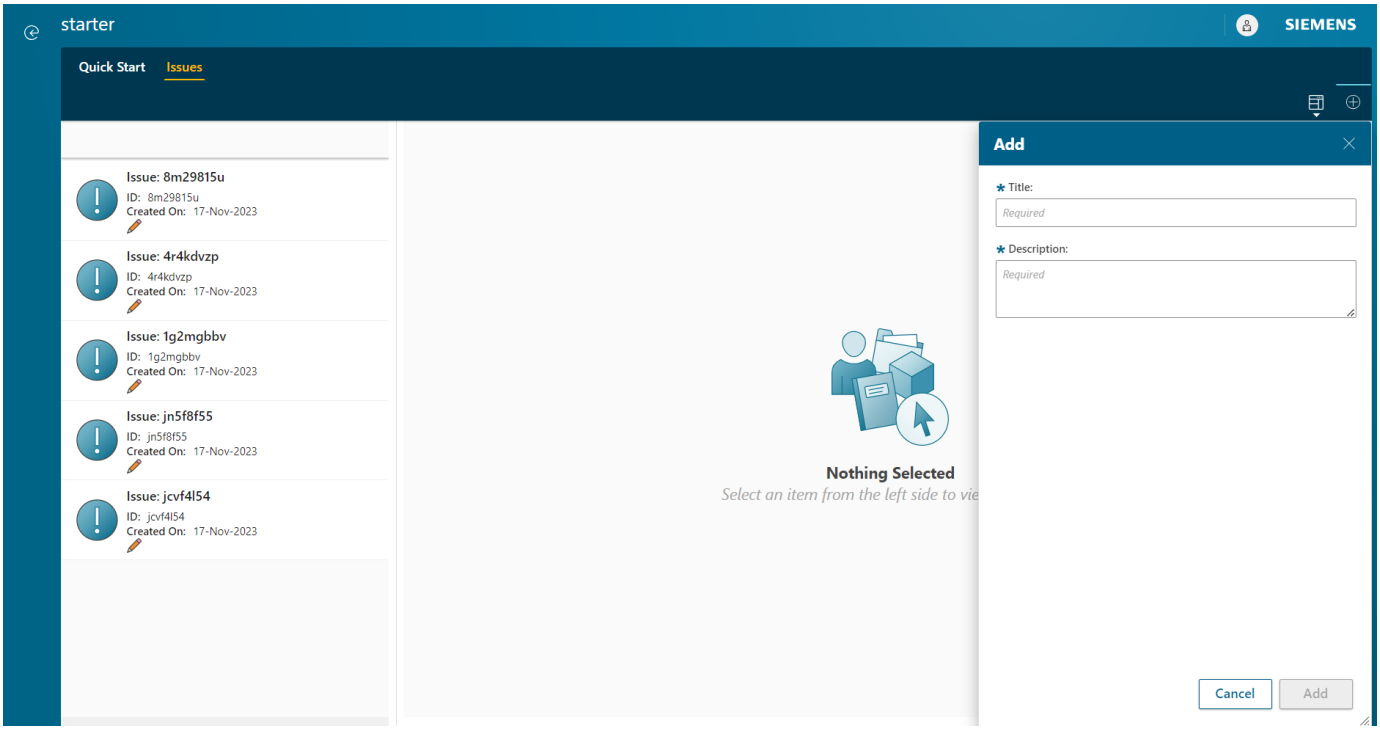
## Creating a component to be displayed in a dialog

The `addIssue` dialog described in the previous section references a component to be rendered in a side panel.

You can find the view model of this component in the `src/viewmodel/AddIssueCmdViewModel.json` file and the corresponding view in the `src/html/AddIssueCmdView.html`.

This is a simple component that displays a form to add a new issue, by specifying its title and description.

You can try it out by clicking the **Add** command on the right the workarea toolbar:




## Adding a form

The `src/html/AddIssueCmdView.html` view is fairly straightforward:

```
<aw-panel>
  <aw-panel-body>
    <aw-widget prop="data.title"></aw-widget>
    <aw-widget prop="data.description"></aw-widget>
  </aw-panel-body>
  <aw-panel-footer>
    <aw-button size="auto" action="closeDialog" button-type="base">
      <aw-i18n>i18n.cancel</aw-i18n>
    </aw-button>
```

```
    <aw-button size="auto" action="add" enable-
when="formProp.attributes.valid">
      <aw-i18n>i18n.add</aw-i18n>
    </aw-button>
  </aw-panel-footer>
</aw-panel>
```

Basically we are defining a panel displaying two fields corresponding to the title and description of the issue that we want to create.

The panel footer contains two buttons, one for performing the add operation and the other to cancel it by closing the dialog. Note in this case how we can use the `aw-i18n` component to render localized strings.

The title and description properties are defined in the `data` section of the `src/viewmodel/AddIssueCmdViewModel.json` viewmodel file, as follows:

```
{
  "data": {
    "title": {
      "displayName": "{{i18n.title}}",
      "type": "STRING",
      "isRequired": true,
      "isEditable": true,
      "dbValue": "",
      "dispValue": "",
      "renderingHint": "textbox"
    },
    "description": {
      "displayName": "{{i18n.description}}",
      "type": "STRING",
      "isRequired": true,
      "isEditable": true,
      "dbValue": "",
      "dispValue": ""
    }
  }
}
```

These are basic **ViewModelProperty** (*VMP* for short) definitions. VMPs are the standard way to model properties and fields in Siemens Web Framework.

**Tip**

You can quickly add a new property to an existing component by running `aw add property` and following the prompts.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Adding actions to save and cancel

You may have noticed that the two buttons in the form reference two actions. Like for commands, actions can be defined also in view models, as follows:

```
{
  "actions": {
    "closeDialog": {
      "actionType": "JSFunction",
      "method": "closeDialog",
      "deps": "js/dialogService",
      "inputData": {
        "editContext": null,
        "popupId": "{{subPanelContext.popupOptions.popupId}}"
      }
    },
    "add": {
      "actionType": "SampleDataPost",
      "inputData": {
        "params": {
          "icon": "typeIssue",
          "type": "Issue",
          "title": "{{data.title.dbValue}}",
          "pinned": false,
          "description": "{{data.description.dbValue}}"
        }
      },
      "events": {
        "success": [
          {
            "name": "primaryWorkarea.reset"
          },
          {
            "name": "closeDialog"
          }
        ]
      }
    }
  }
}
```

Note that, in this case:

- The `closeDialog` action calls the `closeDialog` method defined in **js/dialogService**, a public JavaScript service exposed by Siemens Web Framework.
- The `add` action calls an action of type `SampleDataPost`, which is a **custom action** defined in the `actionTemplateDefs.json` configuration file directly under the module folder.

Can you understand what the `add` action is doing and how? Try to locate the piece of JavaScript code that is executed by the `SampleDataPost` action.

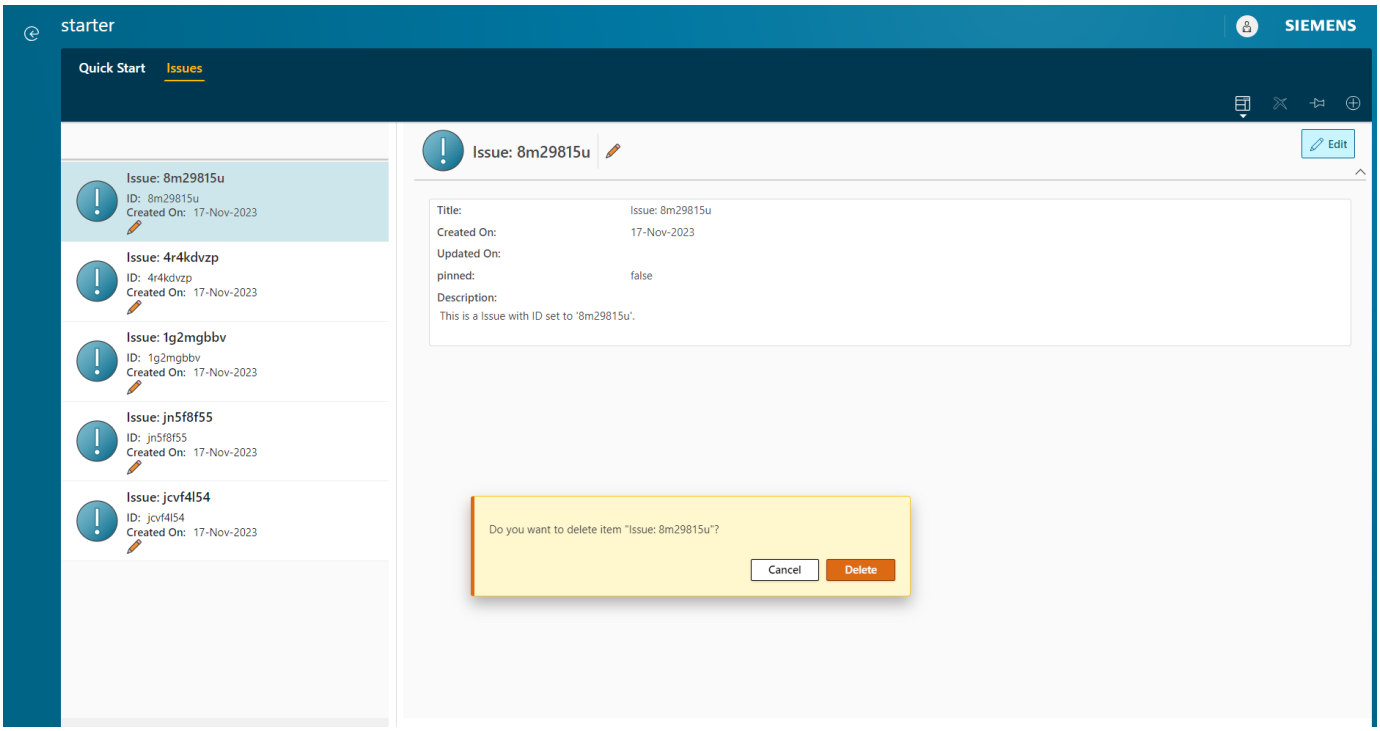

Unpublished work. © 2025 Siemens

Also, notice how the `add` action contains an `events` section. These events can be used to trigger other actions, and in this case the `primaryWorkarea.reset` is a standard Siemens Web Framework event that is used to refresh the primary work area of the page that displays the issues, and the `closeDialog` is a custom event that is used to close the dialog by calling the `closeDialog` action. How? By specifying the following subscription in the `onEvent` section of the view model:

```
{
  "onEvent": [
    {
      "eventId": "closeDialog",
      "action": "closeDialog"
    }
  ]
}
```

## Deleting an item

The **Delete** command can be used to delete an item. The interesting thing about this command is that it displays a confirmation message before deleting an item:




You can achieve this by:

1. Configuring an action to display the confirmation message.
2. Configuring another action to actually delete the item, and execute it only when the user clicks the **Delete** button in the confirmation message.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Configuring the confirmation message

Before diving into the actions needed to implement this functionality, let's have a look at how you can define messages in Siemens Web Framework.

Messages are first-class citizens in the commandsViewModel.json configuration file, and they are defined in their own specific section, like this:

```
{
  "messages": {
    "deleteItemConfirmPopup": {
      "messageType": "WARNING",
      "messageText": "{{i18n.deleteItemConfirmation}}",
      "navigationOptions": [
        {
          "option": "cancel",
          "text": "Cancel",
          "action": ""
        },
        {
          "option": "delete",
          "text": "Delete",
          "action": "deleteItem"
        }
      ],
      "messageTextParams": [
        "{{commandContext.selectionData.selected[0].cellHeader1}}"
      ]
    }
  }
}
```

Essentially, we are configuring a warning message (which is modal by default) with two buttons, specified as `navigationOptions`. Note that the `messageTextParams` array contains one item, which is set to an expression that identifies the currently-selected item retrieved by accessing the **selection data** object.

If you check the `deleteItemConfirmation` string in the `src/i18n/sampleDataMessages.json` file, it is set to `"Do you want to delete item \"{0}\"? "`. Essentially, you can specify placeholders referring to message text parameters positionally, and at run time they will be substituted with the result of the evaluated expression.

## Displaying the confirmation message

The easiest way to display the configuration message is to define a simple `Event` action, like this:

```
{
  "deleteItemConfirmation": {
      "actionType": "Event",
      "method": "Event",
```

```
      "inputData": {
        "events": [
          {
            "name": "confirmDeleteEvent"
          }
        ]
      }
    }
}
```

Here, we are firing a custom event called `confirmDeleteEvent` with no payload, and then we can subscribe to it and display the message, by adding the following subscription in the `onEvent` section of the `commandsViewModel.json` file:

```
{
  "onEvent": [
    {
      "eventId": "confirmDeleteEvent",
      "message": "deleteItemConfirmPopup"
    }
  ]
}
```

## Editing an item

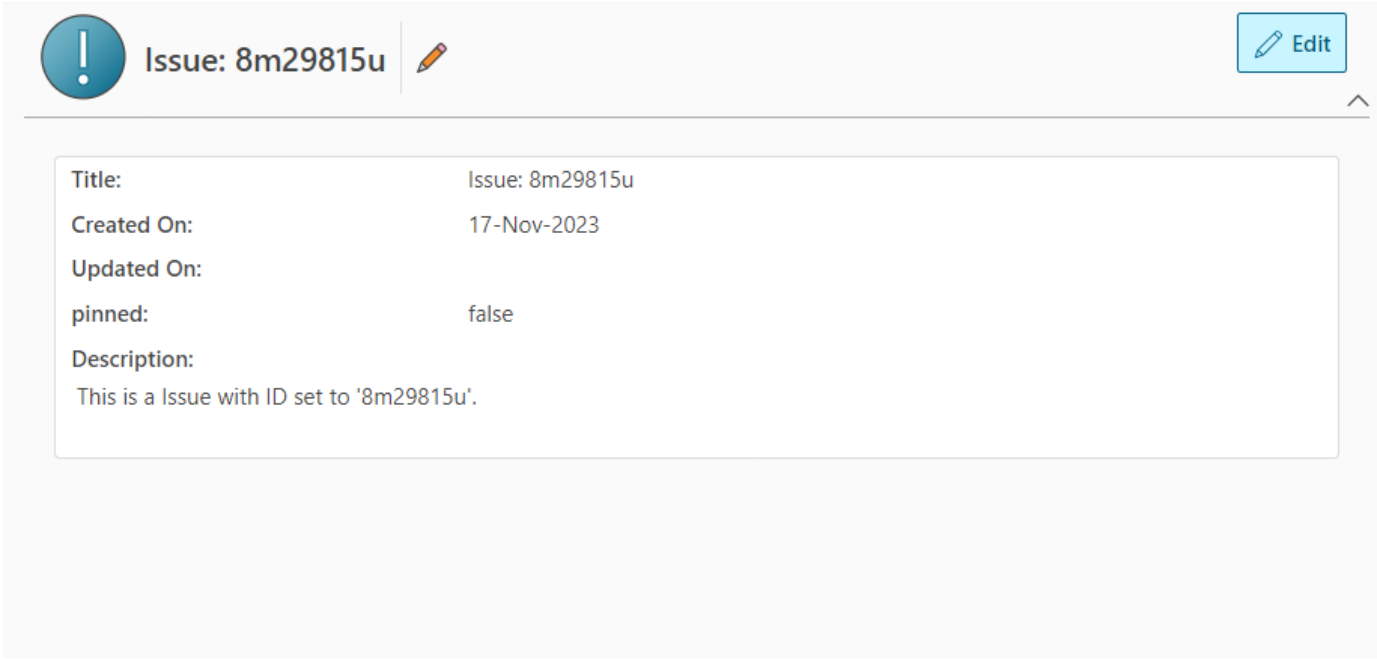
To manage the editing of an issue, three commands are needed:

- An **Edit** command to begin editing
- A **Cancel** command to cancel editing and discard your changes
- A **Save** command to save your changes

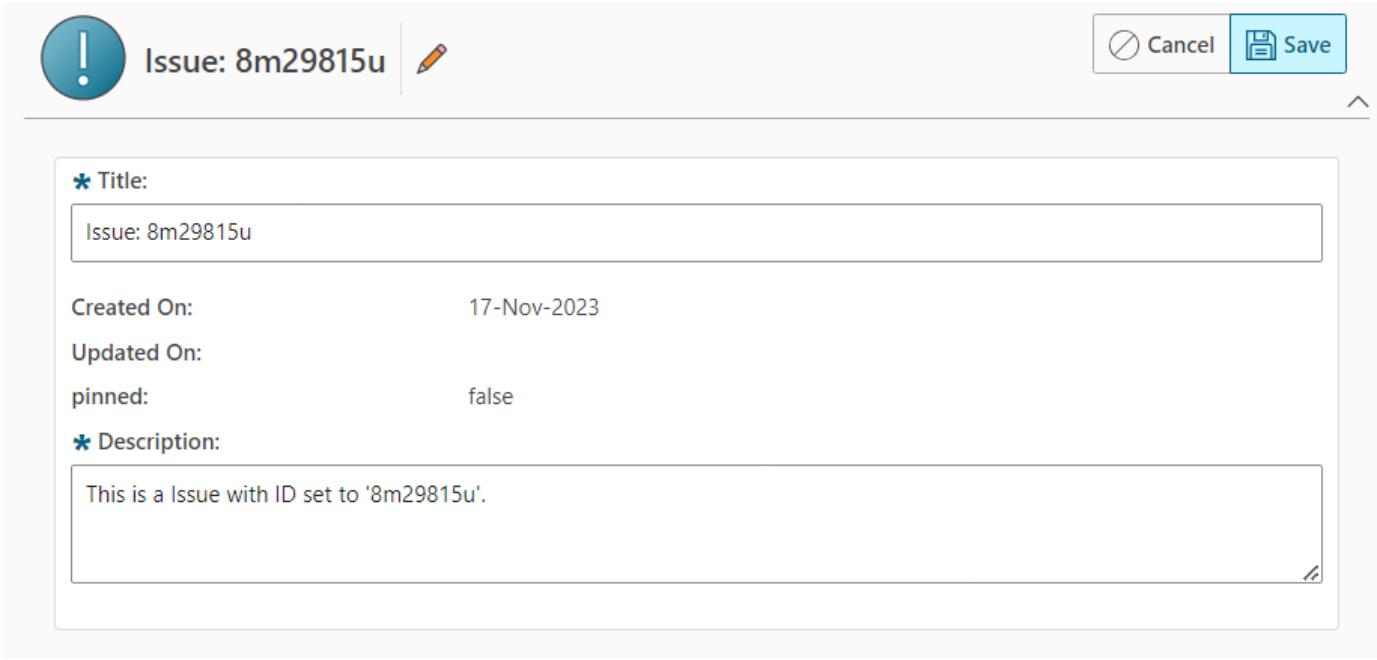
All these commands are typically placed in the command bar in the summary header (`aw_summaryHeader` anchor).

Also, by default the **Edit** command is displayed:


Siemens Web Framework
Unpublished work. © 2025 Siemens




And when it is clicked, the **Cancel** and **Save** commands are displayed instead:




A typical pattern to manage in-place editing of data is the following:

- Define the above-mentioned editing commands in the `commandsViewModel.json` file.
- Manage the editing state via a property defined in the **application context**.
- Execute the appropriate actions, which will be defined in the `IssueDetailsViewModel.json` file.

## Defining the editing commands

The three editing commands are just standard commands that call very simple Event actions that fire a custom event (one custom event per command) such as the following:

```
{
  "startEditItem": {
    "actionType": "Event",
    "method": "Event",
    "inputData": {
      "events": [
        {
          "name": "startEdit"
        }
      ]
    }
  }
}
```

## Defining the editing actions

These custom events are used as triggers to execute specific actions. Now, because the data that is being edited is managed in the `IssueDetailsViewModel.json`, you must define the actions used for editing in this view model file instead of the `commandsViewModel.json` file, like this:

```
{
  "startEdit": {
    "actionType": "JSFunction",
    "method": "updateCtx",
    "inputData": {
      "name": "editInProgress",
      "value": true
    },
    "deps": "js/appCtxService"
  },
  "cancelEdit": {
    "actionType": "JSFunction",
    "method": "updateCtx",
    "inputData": {
      "name": "editInProgress",
      "value": false
    },
    "deps": "js/appCtxService"
  },
  "saveEdit": {
    "actionType": "SampleDataPut",
    "inputData": {
      "params": {
        "id": "{{subPanelContext.selection[0].identifier}}",
        "title": "{{data.item.props.title.dbValue}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "description": "{{data.item.props.description.dbValue}}",
        "type": "Issue",
        "icon": "typeIssue"
      }
    },
    "dataParsers": [
      {
        "id": "itemParser",
        "responseObjs": "data"
      }
    ],
    "outputData": {
      "item": "data",
      "ctx.editInProgress": false
    },
    "events": {
      "success": [
        {
          "name": "primaryWorkarea.reset"
        }
      ]
    }
  }
}
```

The `startEdit` and `cancelEdit` actions are basically used to toggle the value of the `editInProgress` application context. This is a simple boolean flag defined on the application context, and it is used to keep track of whether something is in editing or not. Because it is defined in the application context, this property is also accessible from within the `commandsViewModel.json` file.

`editInProgress` is enough to be used in conditions to determine:

- whether the fields in the form are editable or not
- whether the **Edit** command or the **Cancel** and **Save** commands must be displayed

Actually, these actions could also be defined in the `commandsViewModel.json` file as they do not need to access any local state, but only the application context. However, it makes sense to keep all these editing actions together for maintainability reasons.

The `saveEdit` action is more interesting as it performs many different things:

- It updates the property values that have been modified
- It formats the response of the REST call using the same data parser used for rendering the selected item.
- It sets the `editInprogress` context to `false` (note how this can be done also in the `outputData` section)
- It fires the standard system event to refresh the contents of the primary work area (`primaryWorkarea.reset`)

## Pinning an item

Finally, the **Pin** command can be used to mark an issue item as pinned, so that it is moved at the top of the list.

The interesting thing about this command is that it is a *toggle* command, and it supports a selected state, which is controlled via the `selectWhen` expression defined in its command handler:

```
{
  "pinItemCmdHandler": {
    "id": "pinItemCmd",
    "action": "pinUnpinItem",
    "activeWhen": {
      "condition": "conditions.singleItemSelected"
    },
    "visibleWhen": {
      "condition": "conditions.singleItemSelected"
    },
    "selectWhen": {
      "condition": "conditions.selectedItemIsPinned"
    }
  }
}
```

The action associated to the **Pin** command is similar to the action used to save an item, but note how in this case all the data needed is available in the `commandContext` associated to this command, and therefore this action can be defined in the `commandsViewModel.json` file:

```
{
  "pinUnpinItem": {
    "actionType": "SampleDataPut",
    "inputData": {
      "params": {
        "type": "{{commandContext.provider.type}}",
        "id": "{{commandContext.selectionData.selected[0].identifier}}",
        "pinned": "{{!
commandContext.selectionData.selected[0].props.pinned.dbValue}}"
      }
    },
    "events": {
      "success": [
        {
          "name": "primaryWorkarea.reset"
        }
      ]
    }
  }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

Also, note that the same action is used for both pinning and unpinning, as the `pinned` property is set by toggling its existing value.

### Wrapping up

This concludes this Getting Started guide for Siemens Web Framework. Although this guide barely scratches the surface of what you can do with Siemens Web Framework, it should be enough to get you started and develop basic enterprise applications.

Once you familiarize yourself with the basics, you should go through the rest of the Siemens Web Framework documentation, in particular:

- **Creating User Interfaces**, for more detailed information on how to create specific parts of your user interfaces.
- **Configuration and Extensibility**, for more information on how to leverage Siemens Web Framework declarative capabilities to deliver extensible and configurable user interfaces.
- **Reference**, for comprehensive reference documentation of all public components, services, types, attributes and configuration points.
- **Storybook**, to preview and experiment with all the public components provided by Siemens Web Framework.

## Changelog

All notable changes to Siemens Web Framework packages are listed here.

### 7.2.0

### Enhancements

- Aw-tab-bar: New prop `showSingleTabAsHeader` introduced to display a single tab as an ordinary title
- The subPanelContext support is now available in `TabsViewModel.json` to provide handler specific data.
- The `aw-panel-section` component now supports hiding commands on the panel section. It can be controlled by setting the `hide-commands-when-collapsed` property to `true`. The property defaults to `false`.
- The `aw-listbox` component (LOV) will hide the MRUs when the total number of entries in the list becomes less than 6, excluding the empty option, regardless of filters.
- Introduce `aw-echart` component as a license-free charting option.
- getLocalizedTextFromKey function of localeService.js is enhanced to take 2nd parameter as custom locale code, this is an optional parameter. If you need a localized message bundle/value for some

custom locale pass that custom locale code as 2nd parameter to getLocalizedTextFromKey function. Please note that 2nd parameter of getLocalizedTextFromKey is now reserved for custom locale code.

- getLocalizedText function of localeService.js is enhanced to take 5th parameter as custom locale code, this is an optional parameter. If you need a localized message bundle/value for some custom locale pass that custom locale code as 5th parameter to getLocalizedText function.

### Breaking Changes

- 'js/awTableService' was an internal service. As part of modularization, it has been refactored in SWF. Any imports of 'js/awTableService' will be broken. Please refer to published services only

## 7.1.0

### Enhancements

- Command visibility can now be configured at the command definition level using the new `scope` property, that can be set to a generic condition such as a license check.
- A new banner service is avaliable to display a notification banner in the header of all pages, for a given date range.
- The dynamic width of last column of an `aw-splm-table` component can be enabled via the `flexLastColumn` grid option, which will make the last column take up all the space available to the table.
- The `aw-splm-table` component now supports column grouping where multiple columns can be grouped together giving the table a hierarchial header structure. Please refer to the `Column Grouping` example in storybook for more details.
- The `aw-splm-table` component now supports pinning a row to the bottom or top of the table contents which is exclusive of vertical scrolling of the table. This can be achieved using the new setPinRows() published API.
- The `aw-splm-table` component now supports marking a whole column as modifiable/not modifiable.
- The `aw-tree` component now supports node cell commands where a command can be attached to nodes in a tree by providing the tree with a custom anchor. `cellCommandContext` contains the properties for the node cell commands. Please refer to the `Cell Commands` example in Storybook for more details.
- The `aw-navigate-breadcrumb` component now supports editing the last breadcrumb by adding the isLastCrumbEditable property. Please refer to the `Edit` example in Storybook for more details.
- The JSON schemas of some configuration files have been refactored and enhanced, and have now been published as part of the reference documentation. A migration script can be run to migrate all the affected configuration files automatically. However, even if the script is not run, old files will be converted automatically at build time and will continue to work.
- Existing audit checks have been migrated to leverage Eslint.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- A new Layout Manager page level command has been introduced to change layouts and also save and load recent or saved combinations of views and arrangements.
- The `aw-expandable-row` component now supports a floating overflow that can be specified using `"overflow-type": float`. The overflow-type defaults to `push`.
- The `aw-panel-section` component now supports showing commands to the right of the section title which can be specified by setting the `anchor-position` property to `RIGHT`. The anchor-position property defaults to `LEFT`.
- The dialogService API has been enhanced to support a new method called `updateDialogDimensions` which can be used to update the dialog size programmatically. It takes two parameters, a `popupId` and the `popupOptions` object containing the new height and width values.
- A new layout slot has been added to the `aw-sublocation` component called `aw_workareaHeader` which can be contributed by application teams in the Information Architecture page level toolbar. This slot goes in between the header properties and the workarea toolbar.
- The `aw-navigation-breadcrumb` has been enhanced to show styling based on the Information architecture UX and can be consumed with the `aw-sublocation` component.
- As part of our initial modularization effort, the following new NPM packages have been moved out of the @swf/core package: @swf/engine, @swf/essentials, @swf/elements, @swf/table. Please refer to the Packages page in the documentation for details.

## Deprecations

- The `AwDefaultSublocation` component is now deprecated. Use `AwStandardSublocation` instead.

## 3rd Party Updates

- glob removal
  - Removed the glob from tooling and replaced its uses with fast-glob.
- customize-cra-5 removal
  - Removed the customize-cra-5 from tooling and handled its uses through a custom script coming from @swf/react-scripts.
- xml2js removal
  - Removed the xml2js from tooling and handled its uses through a custom script.
- Highcharts has been updated to version 11.3.0 (was 9.0.1), see the upgrade documentation for more information.

## 7.0.0


Unpublished work. © 2025 Siemens

## Breaking Changes

- `@swf/tooling` is no longer automatically installed as a dependency of `@swf/core`. This will break existing existing projects when upgrading to version 7.0.0. To fix this, add `@swf/tooling` as a devDependency in your project `package.json` file. Refer to the upgrade documentation for more information.
- The NPM scripts for building, starting and testing an application have been updated. For more information, see the upgrade documentation.
- The `aw_rightWall` command bar anchor has been removed from all default view templates. If you were leveraging this anchor for your commands, you should review your command placement based on UX guidelines. A new anchor called `aw_workarea_toolbar`has been added to add commands to the new workarea toolbar (page-level toolbar) whish should be used instead of the right wall in most cases.
- The `aw_userSession` command bar anchor and user avatar have been moved to the header. If you are using one of the default location templates, such as CommonLocation or AwDefaultLocation, there is no impact (any command that specifies this anchor as placement will be moved automatically), but if you are using your own custom location or custom view, you must add the `aw-session-control` component manually in your view, otherwise you will not be able to access user-related information and commands.
- Audit has been enabled for SPLM table. Any use of unsupported column provider or grid options for table will fail the build. See the documentation for the list of supported options.
- Audit shall now enforce that names of .js files are not duplicated across the Siemens Web Framework packages and the application code. Ensure that the .js files you introduce have unique names within your application code and also the Siemens Web Framework core js file names
- Audit check enabled for any usage of private APIs from splm table which are not from table published services. Please see the documentation for list of published services.

## Enhancements

- A new `tabsViewModel.json` configuration file has been created to managing tabs in a more dynamic and extensible way, similar to commands. The new `aw-tab-bar` component can be used instead of `aw-tab-set` to leverage this new configuration file.
- The `aw-splm-table` component now supports accurate scrolling: the size of the scrollbar reflects the position of loaded data within the whole dataset, not only the current page.
- A new `@swf/cli` optional package is now available and can be used to manage Siemens Web Framework projects in a more productive way. It provides a set of commands able to generate a lot of common boilerplate code as well as additional commands for reporting and gathering information about your project.
- It is now possible to specify your own custom component (e.g. a table) to be used as a dropdown for an LOV (`aw-listbox` component). Refer to Storybook for an example on how to achieve this.
- The management of view modes and layouts has been improved, and it is now possible to display up to three different view within a page that leverages one of the default sublocation components.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Additionally, a new layout chooser command can be configured to switch layouts (i.e. the combination of views displayed on a page) and view modes now only affect the current view only.

- A new `aw-summary-header` component can be leveraged to display information about the currently-selected object in secondary workarea tabs, as well as configuring dedicated quick access commands, useful to trigger edit in place or bring the whole secondary work area to full screen.
- The styling of the `aw-panel-section` component has been reviewed: section titles are not displayed with all-caps anymore, and sections now have a border.
- Group commands containing a single child command will now be rendered as single commands if the new `hideOnSingleChoice` placement property has been set to `true`.
- Reference documentation for all the public components, types, configuration files, and services is now available. Note that especially for services only those that are unequivocally considered public have been documented, and more may be added in future releases.
- Showcase has been decommissioned in favor of a new UI Pattern Library based on Storybook.

## Deprecations

- The `AwSidenav` component is now deprecated. Use `AwDialog` instead. Similarly, a `dialog` action should be used instead the `activateCommandPanel` method.
- The `AwColumnChart` component is now deprecated. Use `AwChart` with chartType `column` instead.
- The `AwBalloonPopup` component is now deprecated. Use `AwPopup` instead.
- The `AwBreak` component is now deprecated. Use `<br>` instead.
- The `AwCheckboxList` component is now deprecated. Use `AwWidget` with hint `checkboxoptionlov` instead.
- The `AwCommandPanelSection` component is now deprecated. Use `AwPanelSection` instead.
- The `AwImage` component is now deprecated. Use `AwPic` instead.
- The `AwLinkPopupResponsePropertyValue` component is now deprecated. Use `AwLink` instead.
- The `AwLinkWithPopupMenu` component is now deprecated. Use `AwLink` instead.
- The `AwLinkWithPopup` component is now deprecated. Use `AwLink` instead.
- The `AwNavigatePanel` component is now deprecated. Refer `Atomic data` for panel data exchange.
- The `AwNumeric` component is now deprecated. Use `AwWidget` with hint `integer` instead.
- The `AwPopup2` component is now deprecated. Use `AwPopup` instead.
- The `AwPopupCloseButton` component is now deprecated. Use `AwPopup` instead.
- The `AwPopupDraggable` component is now deprecated. Use `AwPopup` with draggable option instead.
- The `AwPopupPanel2` component is now deprecated. Use `AwPopup` instead.
- The `AwPopupPanel` component is now deprecated. Use `AwPopup` instead.
- The `AwPopupPrimaryButton` component is now deprecated. Use `AwButton` instead.

- The `AwPropertyBoolean` component is now deprecated. Use `AwWidget` with type boolean instead.
- The `AwPropertyImage` component is now deprecated. Use `AwIcon` instead.
- The `AwPropertyNumeric` component is now deprecated. Use `AwWidget` with type integer instead.
- The `AwPropertyStringVal` component is now deprecated. Use `AwWidget` with type string instead.
- The `AwPropertyString` component is now deprecated. Use `AwWidget` with type string instead.
- The `AwPropertyVal` component is now deprecated. Use `AwWidget` instead.
- The `AwProperty` component is now deprecated. Use `AwWidget` instead.

### 3rd Party Updates

- The Noty library which was used for notification messages has been removed.

## 6.3.0

### Breaking Changes

- The @swf/step-definitions package has been renamed to @swf/mars-plugin-swf

### Enhancements

- The following new components have been implemented:
  - AwSummaryHeader: a component used to display information about the selected object in Secondary Work Area.
  - AwCellHeader: a component used to display information about selected object in Panels.
  - AwExpandableRow: a container for constraining child elements to one row. Content can be expanded to multiple rows on user action.
  - AwEmptyWorkarea: a component used to display information when an area is empty to give more guidance to users.
  - AwCompile: a component which is capable of rendering a SWF component by accepting view and viewmodel as a string.
- Command Styling:
  - Commands can now be displayed as buttons using the new 'style' attribute on placement. 'primary' and 'base' are the only applicable values for the 'style' attribute.
  - Group Commands can now be displayed without the arrow chevron, by specifying `"showChevron": false` on the command placement


Siemens Web Framework
Unpublished work. © 2025 Siemens

- Command Labels: visibility of command titles can be managed by the new 'showTitle' attribute on command placement. 'always' and 'never' are the only applicable values for the 'showTitle' attribute. If this attribute is not specified, the command will follow global label setting.
- Command Overflow: A group command can now be specified to take on the 'overflow' of its parent command bar as well as its specified child commands. This is driven by the `"isOverflow": true` attribute on the command placement.
- @mars-plugin-swf, a new waitLogic extension can be added to your project's mars.config.js
  - ```
    waitLogicExtensions: [
    'node_modules/@swf/mars-plugin-swf/src/runtime/wait.js',
    ],
    ```
  - The following waitOptions can be added to the globalWaits object: 'swfLoadingWheel' and 'swfProgressIndicator'; to leverage the custom wait logic functionality specific to SWF applications for test automation.

## 3rd Party Updates

- Removed to gulp-jsbeautifier to address Remote Prototype Pollution Vulnerability of js-beautify. Code formatting after running migration scripts has to be done manually.
- Node.js upgraded to v18
  - With node.js 18.x, npm version updated to 9.x.
  - Windows: Required to upgrade to Node.js 18.x
  - Linux: Since some Linux distributions has compatibility issue, preferred Node.js 18.x and required Node.js 16.x

## Deprecations

- The AwSidenav component are now deprecated and any new usages will be flagged as an error in audit. The direction is to use AwDialog instead.
- The AwRow and AwColumn components are now deprecated. Use AwFlexRow and AwFlexColumn instead, respectively.

## 6.2.0

## Breaking Changes

- Audit check added for any usage of private APIs from pltable modules which are not from published folder.
- Removed createElement and createNgElement from splmTableUtils recommended to use from htmlUtils and ngUtils respectively.
    ```

- Removed getImgUrl from splmTableUtils recommended to use from awIconService.
- The datepicker for non-english locales is showing incorrect date during edit due to incorrect parsing of date.

## 3rd Party Updates

- glob-parent override
  - Override to glob-parent 6.0.2 to address Remote Regular Expression Denial of Service Vulnerability
- set-value override
  - Override to set-value 4.0.1 to address Remote Prototype Pollution Vulnerability
- nth-check override
  - Override to nth-check 2.1.1 to address Remote Denial of Service Vulnerability
- css-what override
  - Override to css-what 6.1.0 to address Remote Regular Expression Denial of Service Vulnerability

## Enhancements

- Movable and Resizable Panels We have introduced a new component "AWDialog" that extends the behavior of AwPopup and addresses limitations with AwSidenav.

  AWdialog supports:

  - Resize
  - Drag
  - Remembers last location

  It can float the dialog and also push it to be part of a page. Push panels are convertible to Float anytime to make some extra room for the content.

# 6.1.0

## Breaking Changes

- Removed js/splmTableNative from swf. Any references to its indirect includes should be replaced directly with respective files.

**Note**

As part of this release, we provided a published service in table with all the exposed APIs that are intended for consumers – any usage of APIs from pltable module that are not in splmTablePublishedService is not recommended. In future, there will be audit checks added for the


Siemens Web Framework
Unpublished work. © 2025 Siemens

same. This is not a breaking change in 6.1.0.0 but we highly recommend consumers to use the published wrappers/methods and avoid using internal table APIs.

## 3rd Party Updates

- react upgrade
  - Update to react 18.2.0 to address End of Life Notification (EOL) of React 17
- react-dom upgrade
  - Update to react-dom 18.2.0 to address End of Life Notification (EOL) of React 17
- yargs upgrade
  - Update to yargs 17.5.1 to be on latest
- @pmmmwh/react-refresh-webpack-plugin upgrade
  - Update to @pmmmwh/react-refresh-webpack-plugin 0.5.5 as part of react-scripts 5 upgrade
- copy-webpack-plugin upgrade
  - Update to copy-webpack-plugin 10.2.4 as part of react-scripts 5 upgrade
- customize-cra-5 Addition
  - Added customize-cra-5 5.0.0 as part of react-scripts 5 upgrade
- customize-cra removal
  - Removed customize-cra as part of react-scripts 5 upgrade
- dataloader upgrade
  - Update to dataloader 2.1.0 as part of react-scripts 5 upgrade
- monaco-editor upgrade
  - Update to monaco-editor 0.33.0 as part of react-scripts 5 upgrade
- monaco-editor-webpack-plugin upgrade
  - Update to monaco-editor-webpack-plugin 7.0.1 as part of react-scripts 5 upgrade
- @testing-library/jest-dom upgrade
  - Update to @testing-library/jest-dom 5.15.1 as part of react-scripts 5 upgrade
- react-refresh upgrade
  - Update to react-refresh 0.12.0 as part of react-scripts 5 upgrade
- @testing-library/react upgrade
  - Update to @testing-library/react 12.1.5 as part of react-scripts 5 upgrade
- @testing-library/user-event upgrade
  - Update to @testing-library/user-event 13.5.0 as part of react-scripts 5 upgrade

- node-libs-browser Addition
  - Added node-libs-browser 2.2.1 to provide all the polyfills to react-scripts 5 as it doesn't come with polyfills by default.
- glob-parent override
  - Override to glob-parent 5.1.2 to address Remote Regular Expression Denial of Service Vulnerability
- chokidar override
  - Override to chokidar 3.5.3 to address End of Life Notification (EOL) Vulnerability
- kind-of override
  - Override to kind-of 6.0.3 to address SQL Injection Vulnerability Vulnerability
- @swf/san-sdk Addition
  - Added @swf/san-sdk 3.2.0 to upgrade Siemens Software Analytics
- ansi-regex override
  - Override to ansi-regex 5.0.1 to address Remote Regular Expression Denial of Service Vulnerability
- axios upgrade
  - Update to axios 0.27.2 to address Remote Information Disclosure Vulnerability
- js-beautify upgrade
  - Update to js-beautify 1.14.2 to address Regular Expression Denial of Service Vulnerability

## 6.0.0

### Breaking Changes

- There are breaking changes in functionalities such as form validation. For more information, see the upgrade documentation.

### 3rd Party Updates

- NodeJs upgraded to v16.
  - With node 16, npm version is also updated to 8.x. When you run npm install` npm 7 onwards, it now automatically installs peer dependencies mentioned in the package.json files of the libraries. This can cause error during npm install. To workaround this issue, node has provided legacy-peer-deps options which can be passed to npm install. This will fallback to old behaviour where peer dependencies wont be installed. For more information visit **legacy-peer-deps**.
    There are two ways to enable legacy-peer-deps option:
    1. Pass this option to npm install command like npm install --legacy-peer-deps.
    2. Add this option to .npmrc file.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- UnhandledPromiseRejections will be treated as Errors and will cause the node process to terminate.

## Enhancements

- Images lazy load
  - To improve data download, we have implemented on demand loading of SVG Images.
  - Support to import SVG image via ES6 import is provided. SVG image can be imported as a Component or String.
  - The on demand loading of svgs might increase the network chattiness for applications that use large number of svg images. To mitigate this, a configuration point to bundle multiple SVG images together in kit.json.

# 5.1.0

## 3rd Party Updates

- htmltojsx removal
  - Removed htmltojsx as it was using **ua-parser-js** (had Malicious Releases Vulnerability) and **json-schema** (had Remote Prototype Pollution Vulnerability).

# 5.0.0

## Breaking Changes

- This release is after migrating the underlying technology from AngularJs to React.js.
- There are multiple breaking changes which can be referred to from the upgrade documentation.

## 3rd Party Updates

- all third parties that were previously used have been either modified or removed. The details of 3rd parties used can be referred from package.json or @ doc\ReadMe_OSS_SWF5.0.0.html.

**Note**

If modifications in code are not reflecting on the browser after build, delete the node_modules/.cache folder, and do a rebuild.


Unpublished work. © 2025 Siemens

Unpublished work. © 2025 Siemens

# 2. Core concepts

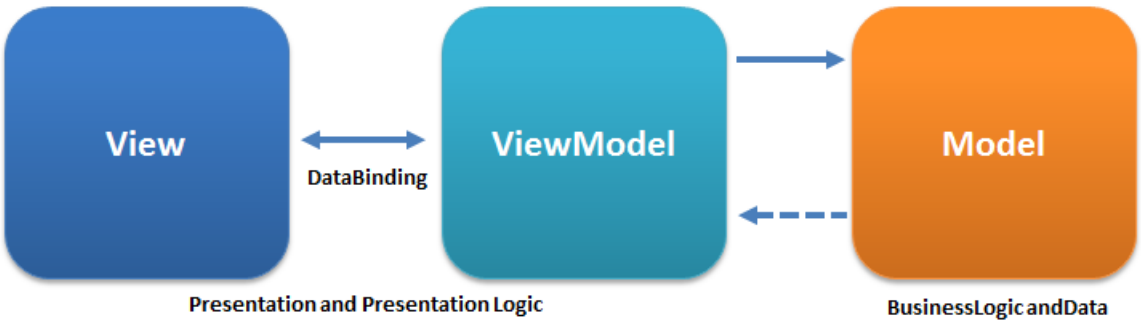
**Declarative programming** is a non-imperative style of programming in which programs describe their desired results without explicitly listing commands or steps that must be performed. Unlike Imperative programming where you code the steps to achieve something (How part); declarative programming focuses on you describing what it should be like (What part).

| Imperative | Declarative |
| --- | --- |
| Given that I am not logged in<br>And I visit the homepage<br>And I click on 'Sign up'<br>And I fill in user name as 'Tom'<br>And I fill in password with 'password'<br>And I fill in confirm password with 'password'<br>And I click on 'Sign up'<br>Then I should be taken to my homepage<br>And I should see 'Hello Tom' | When I sign up for a new account<br>Then I should be taken to my homepage<br>And I should see a greeting message |

Siemens Web Framework follows the declarative programming paradigm. It allows for a concise, codeless and prescriptive definition of the application's user interface and its layout as a HTML, through its rich and reusable building block elements. It drives the application flow through JSON configurations which helps map server responses via data providers, actions, messages, events and their inputs and outputs with the user.

## MVVM

The architectural pattern being followed in Siemens Web Framework is Model–View–ViewModel **MVVM**.




- Model - the representation of actual data and/or information we are dealing with.
- View - the structure, layout, and appearance of what a user sees on the screen. It is the presentation of the data.
- ViewModel - the abstraction of the view exposing the properties and commands.

The communication between the View and the ViewModel is automatic using implicit data binding.

## Siemens Web Framework Declarative Engine

The declarative engine provided by Siemens Web Framework is a runtime which takes care of rendering and managing the web application built on top of it.

It has two main artifacts:

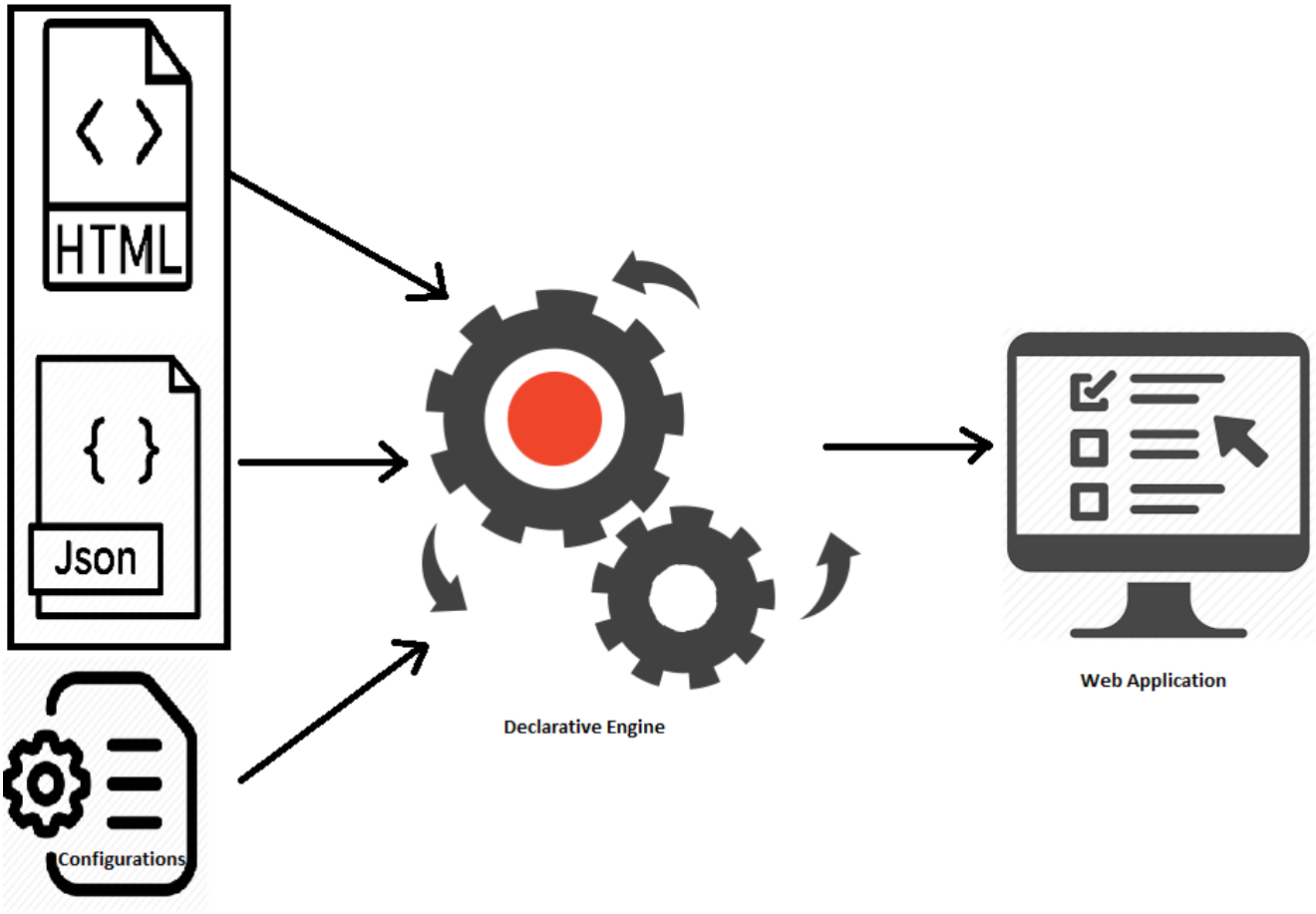
- The **View**, represented as an HTML file which consists of various UI Elements. These UI elements can be HTML elements, elements defined in Siemens Web Framework UI library or custom elements introduced by the consumer application.
- The **ViewModel**, represented as a JSON file. It has manifold capabilities like defining data properties, performing data retrieval, firing or listening to events, handling messages, conditions, localization etc.

Additionally, **Services** are used to manage the business logic referenced in the ViewModel.

A View and a ViewModel constitute a **Component**, which defines the contract, look-and-feel and implementation of a **UI Element**.

Other declarative artifacts are the configuration points which when defined act as an extension to the OOTB behavior provided by the Siemens Web Framework. An example is commandsViewModel.json which is the house for configuring all commands, their definitions, placements, operations etc.


Siemens Web Framework
Unpublished work. © 2025 Siemens




These artifacts only cover the '**What**' part of the web application i.e. what UI elements to render, what data to render, what operations to perform on specific user gestures etc.

The Declarative runtime interprets them and directly renders the web page. The '**How**' part is completely managed by this declarative engine. This is how consistent UI and behavior is achieved throughout the application.

## Why declarative UIs?

Declarative UI is a capability provided by the Siemens Web Framework that allows for a concise, codeless and prescriptive definition of the UI view content, its layout, routing and behaviors. Actions, messages, events, service calls and their inputs/outputs can be mapped and described.

It provides an abstracted means of declaratively defining the UI content and its behaviours. *How* it is implemented is hidden/abstracted.

There are multiple drivers for why we are introducing declarative UI.


Unpublished work. © 2025 Siemens

## Performance - minimize client code base

Since Siemens Web Framework is used for creating web client Solutions we need to operate over high latency WAN and be tolerant of low bandwidth scenarios. We don't want the user to wait a long time for the browser to load MBs of code from the appserver and we need to work within the resource constraints of the browser and in thin contexts such as for tablets and smart phones.

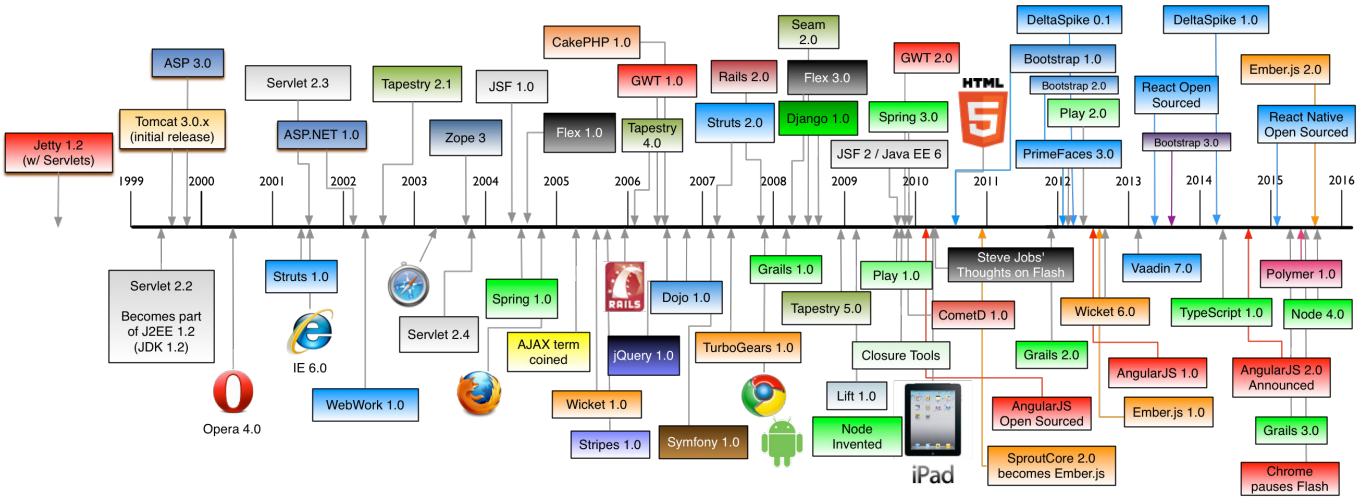
Declarative offers us the ability to significantly reduce the amount of code that is written and downloaded. It allows large portions of the code base to be replaced with a much lighter weight declarative definition to achieve the same end results.

## Efficiency – develop and iterate

We want application developers to be able to quickly and easily develop UI. Declarative 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 working with application developer to wire up the view model based on the intent.

## Maintainability – insulate teams from the underlying technology implementation. Decoupling enables evolution without ripple

The web world continues to evolve rapidly. Web frameworks come and go and frequently have breaking releases from one release to the next. In the past all developers required exactly the same version of the framework and key packages. When we need to update or change these to evolve this impacts all client code developed and several solutions/products that are impacted by such changes and a large amount of code that must be updated. Our ability to evolve is significantly diminished.




Declarative provides an abstraction that application developers can consume such that the underlying technology can be evolved independently.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Consistency – adhere to Ux standards and leverage templates

Declarative insulates the developer from the styling and CSS. The declarative framework handles these pieces for you and you can concentrate on the intent. By all teams using the same set of UI elements we achieve greater consistency than having many thousands of lines of code to achieve the same intent across each usage.

### Flexibility - support the needs of multiple Solutions

Different products and solutions are targeted for different end users. They require a range of capabilities, layout and content patterns. Siemens Web Framework declarative UI provides a common set of basic building blocks, the ability to augment that vocabulary for specialized elements where required and the means to layout with greater flexibility. At the same time we maintain a common look and feel for Siemens PL web client solutions.

## Main declarative UI artifacts

Siemens Web Framework relies on several *declarative UI artifacts* to define user interfaces. The most important are the following:

- **Component** A **Component** defines a portion of a user interface, like a reusable **UI element**. It is defined by a View and a ViewModel.
- **View** The **View** defines the visual representation of a Component.
- **ViewModel** The **ViewModel** defines the configuration and internal logic of a Component via **Actions**.
- **Services** **Services** are JavaScript modules exposing methods that can be used to implement Actions.
- **Command** **Commands** encapsulate Actions as objects and provide an interactive way for the user (e.g. buttons) to work with their product, process, or other data.

## Component

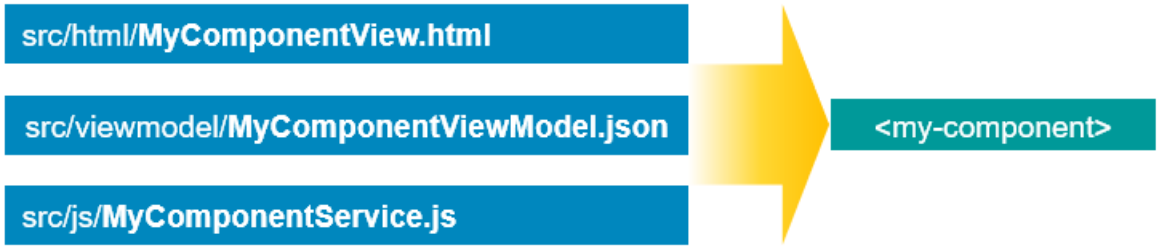
A *component* defines a new, reusable, **UI Element** that can be used in **views**.


Unpublished work. © 2025 Siemens

Components are defined by:

- A **view model**
- A **view**
- Optionally, one or more **services** providing a codeful implementation of some of the actions referenced in the view model

At build-time, the *View Compiler* is used to transform these declarative artifacts into a single **React** component:




**Tip**

As an alternative to a view template, it is also possible to use a *render function* as an action specified via the `render` **lifecycle hook**. Please note however using a declarative view template is recommended whenever possible.

## Naming Rules

When creating a new component, keep in mind the following naming rules:

- The component name must be in PascalCase and be composed of at least two words
- The view file must start with the component name in PascalCase and end with `View.html`.
- The view model file must start with the component name in PascalCase and end with `ViewModel.json`.
- There are no enforced naming conventions for component Services, as long as they are referenced correctly in view model actions. If one service, in particular, is used exclusively for a specific component, its name should start with the component name in PascalCase and end with `Service.js`.
- The resulting UI element tag will be set to the name the component in kebab-case, so for example, if your component is called `MyComponent` the corresponding tag will be `<my-component>`.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Properties

Because Components are inherently reusable, as of Siemens Web Framework 5, the view model schema has been extended to support the definition of component properties using the `props` section. In this way, the view model contains the full definition of the contract of a component/UI element.

Consider the following view model defining the contract for the `aw-checkbox` UI element:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "render": {
            "method": "awCheckboxRenderFunction",
            "deps": "js/AwCheckboxService"
        }
    },
    "data": {},
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    },
    "props": {
        "action": {
            "type": "action"
        },
        "prop": {
            "type": "field"
        },
        "is-negated": {
            "type": "object"
        }
    }
}
```

In this case:

- The component relies on a render function specified via the `render` lifecycle hook instead of a declarative view.
- Three properties are defined for this component: `action`, `is-negated`, and `prop` (note that some props may also be optional).

## Binding Types

Because the view template syntax is very simple and resembles HTML, it is necessary to provide some additional hinting to the view compiler so that properties are processed as needed.


Unpublished work. © 2025 Siemens

As a result, when defining properties in the `props` section of your view model you must also specify one of the binding types listed in the following table:

| Binding Type | Evaluation |
|---|---|
| object | The property value is treated as a JavaScript expression. |
| string | The property value is treated as a raw string (not evaluated). |
| action | The property value is treated as an Action. |
| field | The property value is treated as a Field. |

Consider the following `aw-checkbox` usage in a view template:

```
<aw-checkbox prop="data.title" action="notifyCheck"/></aw-checkbox>
```

In this case, based on the `props` section of the `AwCheckboxViewModel.json` file in the previous section, the compiler knows that:

- `data.title` must be processed as field binding, so the value of the corresponding field will be retrieved from the container view model and passed to the checkbox component.
- `notifyCheck` must be processed as an action binding, so the corresponding action will be retrieved from the container view model and executed when the checkbox is selected.

### Lifecycle Hooks

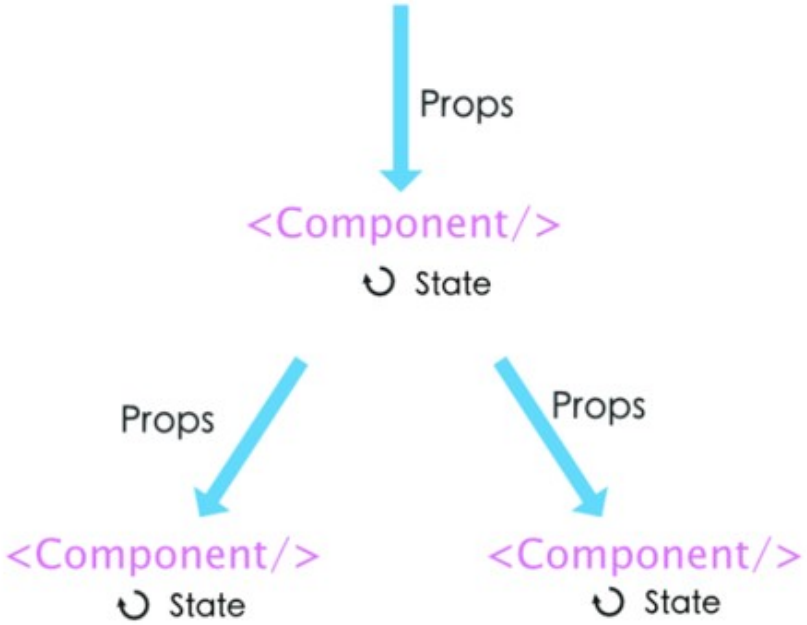
**Lifecycle hooks** were already introduced for view models in previous versions of Siemens Web Framework, and they are now even more important when creating reusable components. In particular, the following new hooks have been introduced specifically to support Components:

- `onUpdate` this hook executes an action whenever the component is updated.
- `render` if this hook is specified, it must define a render function that will be used to render the component *instead* of specifying a declarative view template.

## Properties vs. state

When creating new **components**, it is crucial to understand the difference between *properties* (also called *props*) and *state*.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Properties

The *properties* of a component are:

- read-only parameters passed to the component
- passed (one-way) from the parent to the child component as HTML element attributes
- can be JavaScript objects, expressions, or functions

## State

The *state* of a component:

- is comprised of private data owned and maintained by the Component.
- can only be modified using specific methods exposed by the Component.
- cannot be accessed/modified outside the Component
- can be passed as properties to child components.

## Changing Properties or State

It is important to remember that:

- data flows one and only one way i.e. **unidirectionally**
- a change in either properties or state results in the component's virtual DOM being re-rendered

### Summary

| | Properties | State |
|---|---|---|
| Can get initial value from the parent Component | Yes | Yes |
| Can be changed by the parent Component | Yes | No |
| Default values can be set inside the Component | Yes | Yes |
| Can change inside the Component | No | Yes |
| Can set initial values for child Components | Yes | Yes |
| Can change in child Components | Yes | No |

## Rendering components

A component can be rendered in two ways:

- By specifying a declarative **view**
- By specifying a JSX render function

The following sections outline the main differences between these two methods.

### View

When you are using a **declarative view**:

- The syntax is the same as an ordinary declarative view
- The View Compiler transpiles these into a render function
- The imports are auto-generated and it is no longer required to be specified in the imports section of the view model
- The naming convention followed is **kebab-case**

### JSX render function

When using a JSX render function instead of a declarative view:

- JSX syntax is used
- Access to the view model, sub panel context, application context and any passed-in props is available
- The imports are required directly from the view model
- The naming convention followed is **PascalCase**


Siemens Web Framework
Unpublished work. © 2025 Siemens

- **Fields** must be passed explicitly via `props.fields.fieldName` instead of `data.fieldName`. Furthermore, when the **data binding type** of a prop is set to `field` (like for property widgets like AwTextbox), the field must be spread, like this:

```
<AwTextbox {...props.fields.myProp1}></AwTextbox>
```

...while in a declarative view, the above would be written like this:

```
<aw-textbox prop="data.myProp1"></aw-textbox>
```

**Note**

It is recommended to use views over handwritten JSX render functions whenever possible.

## Render function signature

Every single component has the exact same render function signature, and the following arguments are passed to it:

- `props` The props passed to the component.
- `viewModel` The fully-populated view model of the component.
- `ctx` application context as defined in the `ctx` section of its view model.
- `elementRefList`
- `subPanelContext`
- Any other property that is sent to the component

Below is an example of a compiler-generated render function based on a view:

```
/*
<div class='sw-logoContainer'>
    <div class='sw-logoImage'>
        <aw-icon icon-id='miscSiemensLogo' exist-when='conditions.logoEnabled'></aw-icon>
    </div>
</div>

*/
const renderFn = ( { viewModel: { data, dataProviders, dispatch, conditions, selectionModels  }, grids, ctx, actions, fields, messages, i18n, prop: { subPanelContext }, formProp } ) =>
<Fragment>
    <div className="sw-logoContainer">
        <div className="sw-logoImage">
            <AwIconExistWhen iconId="miscSiemensLogo" existWhen={AwParseService.instance( "conditions.logoEnabled" )( { data, ctx, i18n, subPanelContext, conditions, formProp, selectionModels } )
            </AwIconExistWhen>
        </div>
    </div>
</Fragment>;

/*
define component
*/
const Component = createComponent( declViewModelJson, renderFn, undefined, ctxImportObj );

export default Component;
```

# Mapping React hooks to declarative constructs

This page is mainly targeted at developers who have React background and are looking for some help with mapping React hooks to Declarative constructs.

## useState

The **useState hook** is mapped to `viewModel.data` (the local state of the component) and `viewModel.dispatch` (the method to update the local state of the component i.e. React's `setState`).

Consider the following React component:

```
const {useState} = React;

const AwColorPicker = () => {
    const [state, setState] = useState("#000000");

    const setColor = value => {
        setState( value );
    };

    return (
      <div>
         <input type='color' value={state} onChange={e =>
setColor( e.target.value )} />
         <div style={{backgroundColor: state, width: '50px',
height:'50px'}}></div>
      </div>);
};
ReactDOM.render(<AwColorPicker />, document.getElementById('app'));
```

This is using an HTML input of type `color` with a callback to be invoked whenever a new color is selected from the `AwColorPicker` component. The same color is saved as the local component state and `setState` is the API provided by React to update it.

The same can be achieved using Siemens Web Framework APIs by making the following changes:

1. Create component level state i.e. the mapping for `state` in `useState`. In the view model, define the following:

   ```
   {
       "data":{
           "_color": "#000000"
       }
   }
   ```

   This ensures a local component state with name `_color` is created and initialized with a default value.


Siemens Web Framework
Unpublished work. © 2025 Siemens

2. To update the state, add the following code:

```
const setColor = value => {
    viewModel.dispatch( { path: 'data._color', value: value } );
};
```

`viewModel.dispatch` is an equivalent to setState API of useState React hook which will update the local component state with the desired value.

Overall, this is what the view model of the converted component would look like:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "render": "render"
    },
    "actions": {
        "render": {
            "method": "awColorPickerRenderFunction",
            "deps": "js/AwColorPickerService"
        }
    },
    "data":{
        "_color": "#000000"
    }
}
```

and the JavaScript service exposing the methods referenced in the view model:

```
import AwFlexRow from 'viewmodel/AwFlexRowViewModel';
import AwFlexColumn from 'viewmodel/AwFlexColumnViewModel';

export const awColorPickerRenderFunction = props => {
    const { viewModel } = props;
    const setColor = value => {
        viewModel.dispatch( { path: 'data._color', value: value } );
    };

    return (
      <AwFlexRow>
            <input type='color' className='sw-section-content w-4 h-2'
value={viewModel.data._color} onChange={e => setColor( e.target.value )} />
            <AwFlexColumn className='w-8' color={viewModel.data._color}>
            </AwFlexColumn>
      </AwFlexRow>
    );
};
```

Here is a before-and-after comparison:

## useRef

The **useRef hook** maps to `elementRefList`, which is available in the render function of the component as well as any of the lifecycle actions defined in that component. This lets you have the DOM reference of the specified element.

Consider the following React component:

```
const {useRef} = React;

const AwFocusInputBox = () => {
   const inputElement = useRef();

   const focusInput = () => {
    inputElement.current.focus();
  };

   return (
      <div>
         <input type='text' ref={inputElement}></input>
         <button onClick={focusInput}Focus Input></button>
      </div>);
};
ReactDOM.render(<AwFocusInputBox />, document.getElementById('app'));
```

This is using an HTML input of type `text` and HTML button with a callback to be invoked whenever the button is clicked. The textbox becomes focused every time the button is clicked. This has been achieved via useRef hook wherein the DOM reference of input text box is persisted and focused on the click action of the button.

The same can be achieved using Siemens Web Framework APIs by making the following changes:

1. Create component with render a function using the `elementRefList` option. In the view model, define the following:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "render": "render"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    },
    "actions": {
        "render": {
            "method": "awFocusInputBoxRenderFunction",
            "deps": "js/AwFocusInputBoxService",
            "options": {
                "elementRefList": [ "inputBox" ]
            }
        }
    }
}
```

This ensures an `elementRefList` with name `inputBox` is available in the render function.

2. In order use the same element reference, define a render function as follows:

```
export const awFocusInputBoxRenderFunction = ( props ) => {
    const inputElementRef = props.elementRefList.get( 'inputBox' );
     return <div>
        <input type='text' ref={inputElementRef}></input>
        <AwBreak></AwBreak>
        <AwButton size='auto' action={props.actions.focusInput}>Focus
Input</AwButton>
    </div>;
};
```

Note that `props.elementRefList` provides the DOM reference which has been binded with the input text box. And the callback on the button has been wrapped via declarative actions which will do the same job of focusing the reference DOM element

Overall, here is how the view model of the converted component would looks like:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "onMount": "initialize",
        "render": "render"
    },
    "actions": {
        "render": {
            "method": "awFocusInputBoxRenderFunction",
            "deps": "js/AwFocusInputBoxService",
            "options": {
                "elementRefList": [ "inputBox" ]
            }
        },
        "initialize": {
            "actionType": "JSFunction",
            "method": "initialize",
            "inputData": {
                "elementRefList": "{{parameters.elementRefList}}"
            },
```

```
            "outputData": {
                "elementRefList": ""
            },
            "deps": "js/AwFocusInputBoxService"
        },
        "focusInput": {
            "actionType": "JSFunction",
            "method": "focusInput",
            "inputData": {
                "elementRefList": "{{data.elementRefList}}"
            },
            "deps": "js/AwFocusInputBoxService"
        }
    }
}
```

and the JavaScript service exposing the methods referenced in the view model:

```
import AwButton from 'viewmodel/AwButtonViewModel';
import AwBreak from 'viewmodel/AwBreakViewModel';

export const awFocusInputBoxRenderFunction = ( props ) => {
    const inputElementRef = props.elementRefList.get( 'inputBox' );

    return <div>
        <input type='text' ref={inputElementRef}></input>
        <AwBreak></AwBreak>
        <AwButton size='auto' action={props.actions.focusInput}>Focus
Input</AwButton>
    </div>;
};

export const initialize = elementRefList => {
    return elementRefList;
};

export const focusInput = elementRefList => {
    const inputElementRef = elementRefList.get( 'inputBox' );
    inputElementRef.current.focus();
};
```

Here is a before-and-after comparison:


Siemens Web Framework
Unpublished work. © 2025 Siemens

## useEffect

The **useEffect Hook** maps to the `onUpdate` **view model lifecycle hook** which specifies the observer or the dependencies that will cause the corresponding actions to be executed when their value changes.

Consider the following React component:

```
const {useState, useEffect} = React;

const AwCounter = () => {
const [count, setCount] = useState(0);

  useEffect(() => {
     console.log(count);
  },[count]);

    return (
      <div>
        <p>You clicked {count} times</p>
         <button onClick={() => setCount(count + 1)}Click Me></button>
      </div>);
};lifecyclective
ReactDOM.render(<AwCounter />, document.getElementById('app'));
```

This is using an HTML button with a callback that is invoked whenever the button is clicked. The number of times the button is clicked is printed on the screen. Every time the count is incremented, there is a corresponding `useEffect` hook with dependency on `count` which means that when the `count` changes, the value gets printed in the browser console.

The same can be achieved using Siemens Web Framework APIs by making the following changes:

1. Create component with a `render` and an `onUpdate` lifecycle hook. In the view model, define the following:

```
{
    "lifecycleHooks": {
        "render": "render",
```

```
        "onUpdate": {
            "observers": [ "viewModel.data._count" ],
            "action": "showCount"
        }
    }
}
```

This ensures that the `showCount` will be invoked when `_count` is modified.

Overall, this is what the view model of the converted component would look like:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "render": "render",
        "onUpdate": {
            "observers": [ "viewModel.data._count" ],
            "action": "showCount"
        }
    },
    "data":{
        "_count": 0
    },
    "actions": {
        "render": {
            "method": "awCounterRenderFunction",
            "deps": "js/AwCounterService"
        },
        "setCount": {
            "actionType": "JSFunction",
            "method": "setCount",
            "inputData": {
                "count": "{{data._count}}"
            },
            "outputData": {
                "_count": ""
            },
            "deps": "js/AwCounterService"
        },
        "showCount": {
            "actionType": "JSFunction",
            "method": "showCount",
            "inputData": {
                "count": "{{data._count}}"
            },
            "deps": "js/AwCounterService"
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

and the JavaScript service exposing the methods referenced in the view model:

```
import AwButton from 'viewmodel/AwButtonViewModel';
import AwBreak from 'viewmodel/AwBreakViewModel';

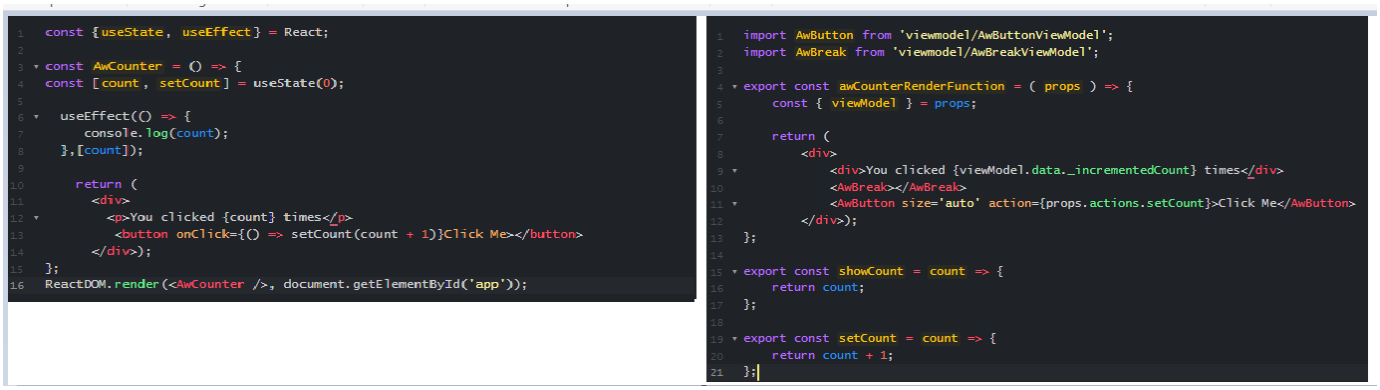
export const awCounterRenderFunction = ( props ) => {
    const { viewModel } = props;

    return (
        <div>
            <div>You clicked {viewModel.data._count} times</div>
            <AwBreak></AwBreak>
            <AwButton size='auto' action={props.actions.setCount}>Click Me</
AwButton>
        </div>);
};

export const showCount = count => {
    console.log(count);
};

export const setCount = count => {
    return count + 1;
};
```

Here is a before-and-after comparison:




## Summary

The following table lists the mapping between React and declarative constructs:

| React Construct | Declarative Construct |
|---|---|
| state | viewModel.data object |
| setState | viewModel.dispatch method |

| | |
|---|---|
| useRef | elementRefList option |
| useEffect | onUpdate lifecycle hook |

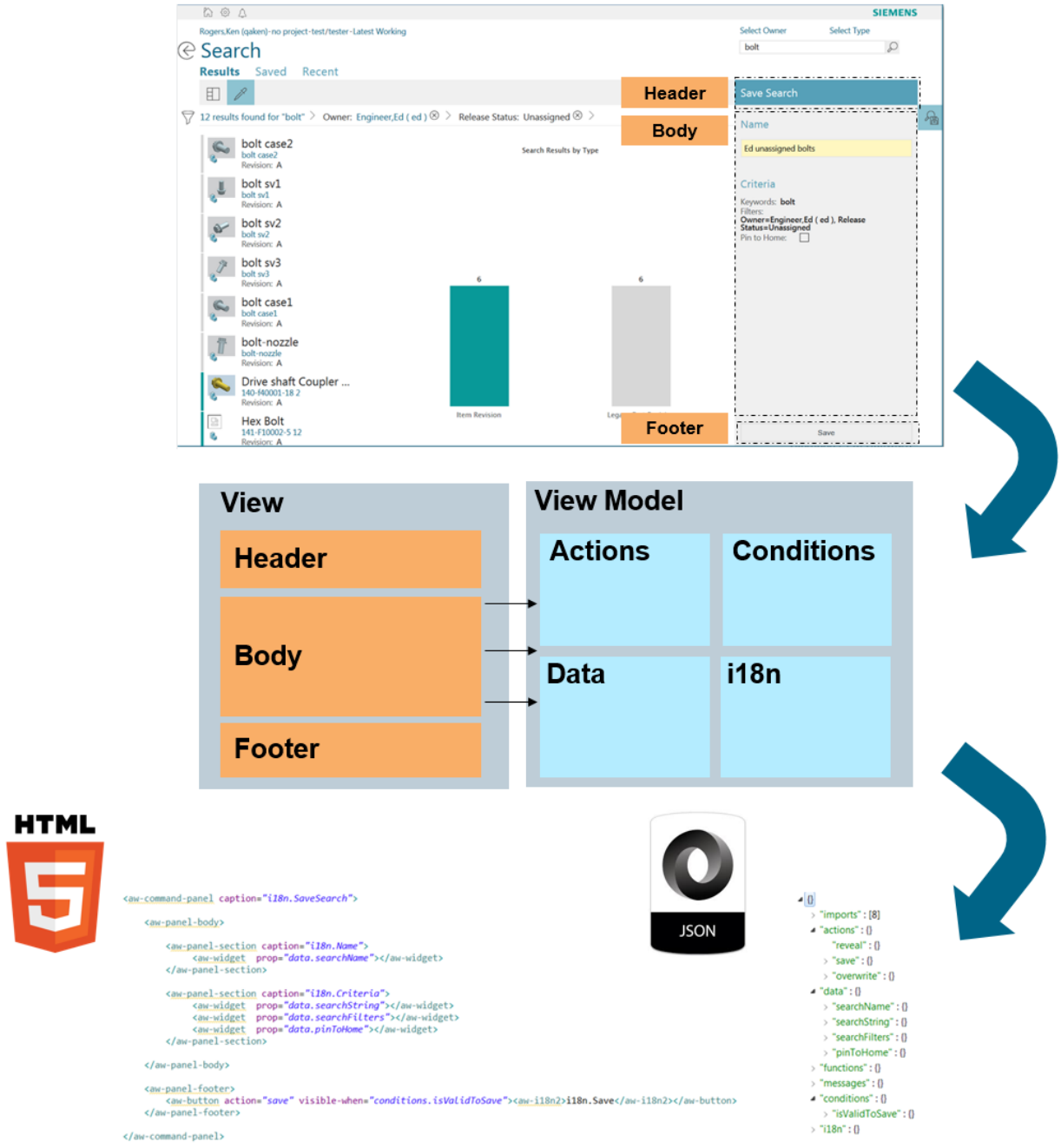
## View

A declarative definition of the UI has two main artifacts - the **View** and **ViewModel**.

This provides an abstracted means of declaratively defining a Page or Panel and its behaviours.

- We don't expose CSS classes, raw divs or any implementation details
- *How* it is implemented is hidden/abstracted


Unpublished work. © 2025 Siemens




The View is markup: simple HTML.

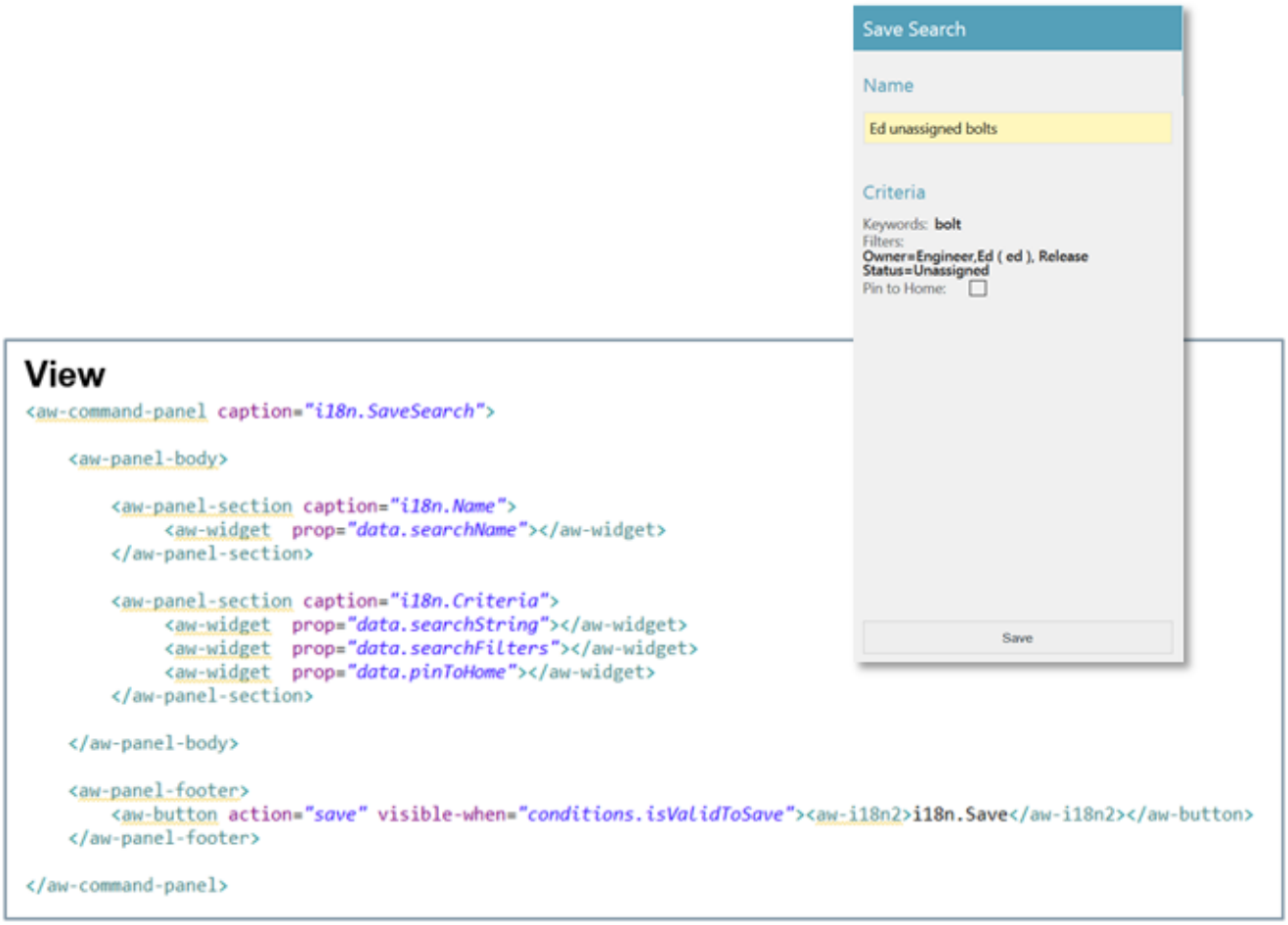
Vocabulary for the View is made up of **UI Elements**. You can extend that vocabulary by introducing new elements.

It is responsible for:

- View hierarchy - sections, content
- It maps to *data*, *actions*, *conditions* and *localizations* that are defined in the ViewModel

- *visible-when* clauses allow control of the visibility of UI elements
- *enable-when* clauses allow control of the enablement of UI elements
- It can declaratively refer to live **application context**
  - *selected* - the selected object
  - Additional contexts can be registered

Must be semantically meaningful




## View model

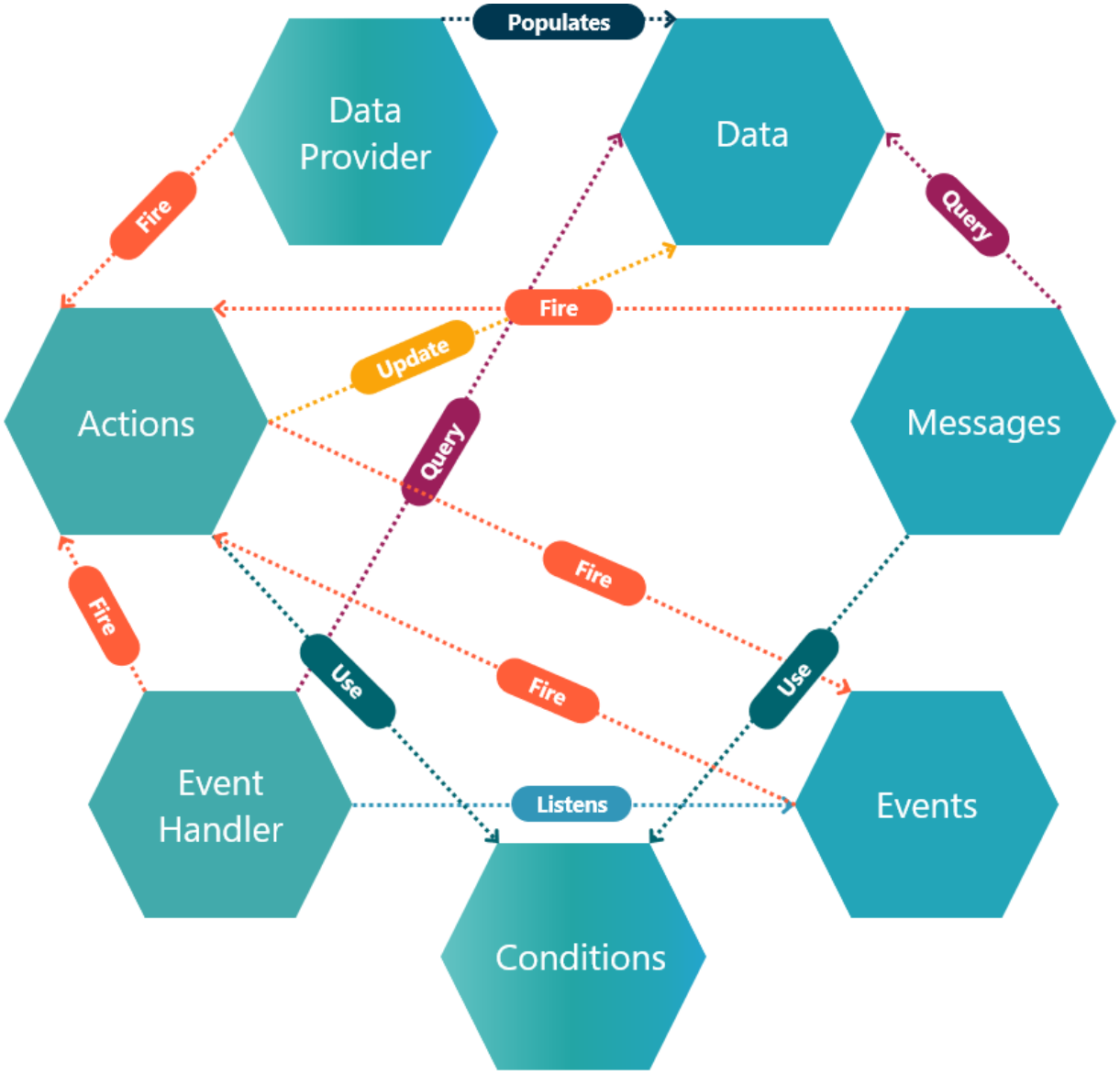
A view model is a JSON configuration file that is used to manage the state of a view/component. There is a live data binding between the **view** and the parameterized ViewModel.

It is declaratively defined allowing the mapping of inputs/outputs for **actions** (such as service calls), firing and listening to **events**, **conditional behaviours**, **messages**, view model data querying, update and

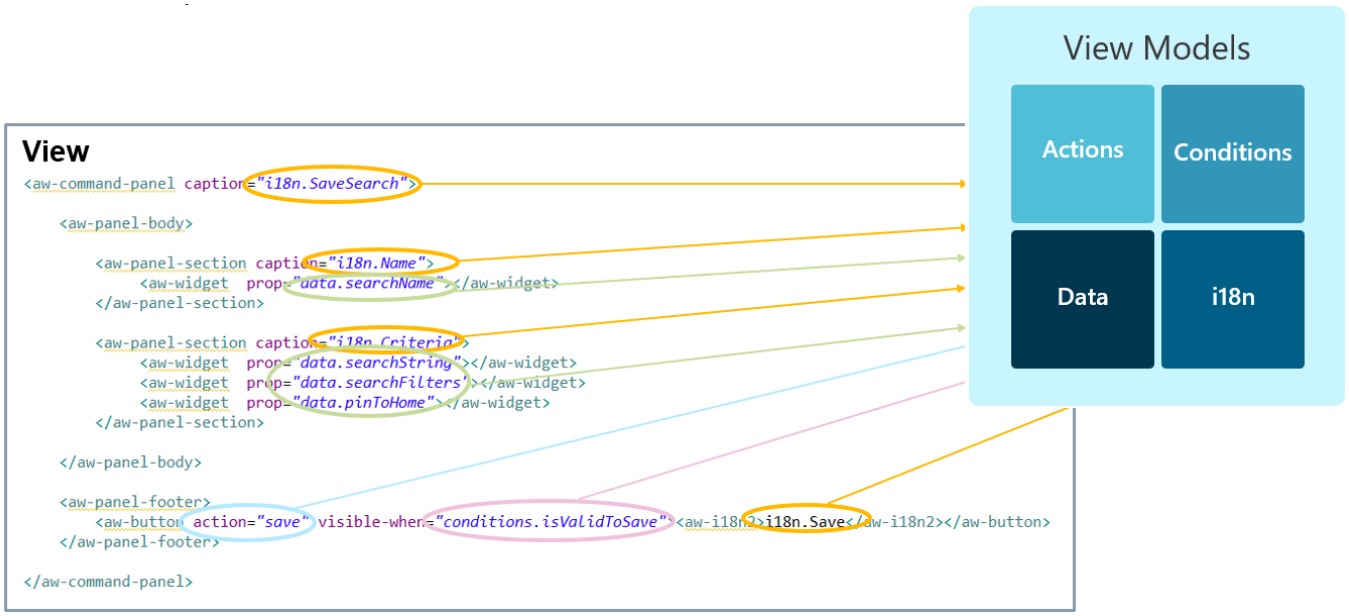

Unpublished work. © 2025 Siemens

**localization** of user-facing text. The ViewModel adheres to a schema. The Siemens Web Framework **audit tooling** validates ViewModel against its schema and its logical integrity with its corresponding View.

The conceptual model to adopt is one of a kind of finite state machine.




The following illustrates how the View maps to relevant sections of the ViewModel:




For more information on the different sections of a ViewModel and constructs used inside ViewModels, see the following sections:

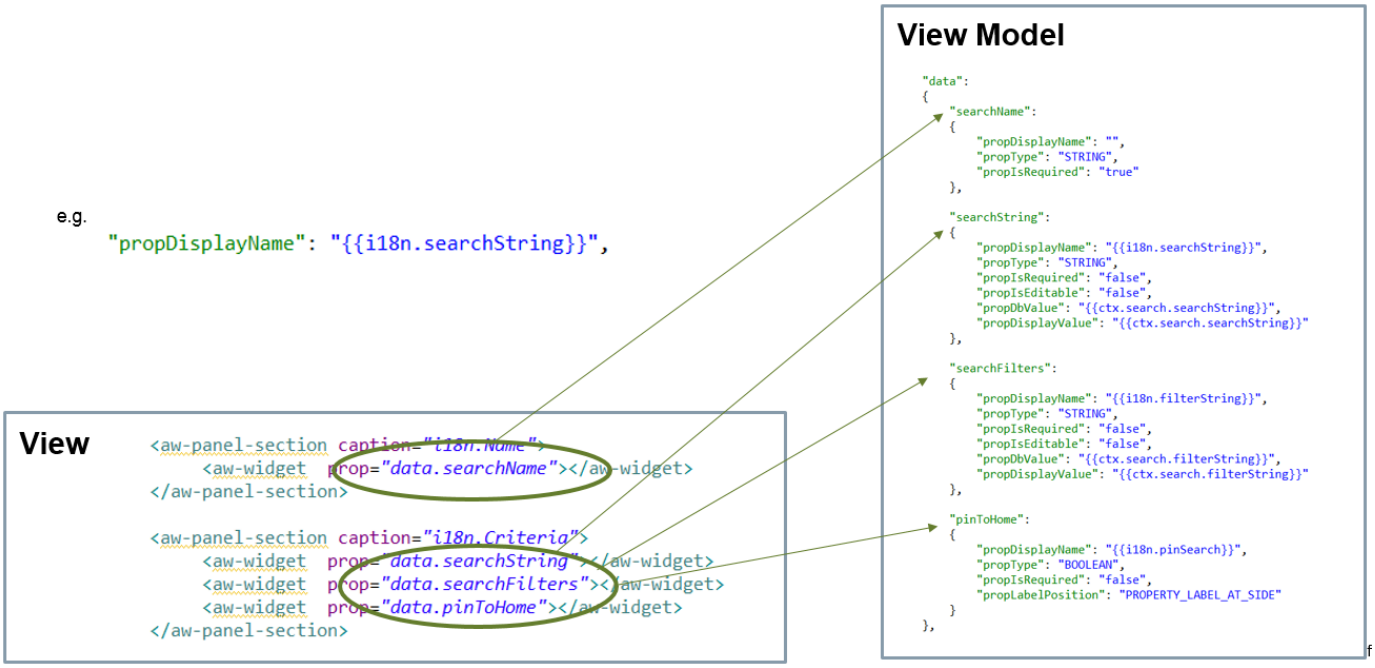
- **Data**
- **Actions**
- **Functions**
- **Data Providers**
- **Data Parsers**
- **Event**
- **Lifecycle Hooks**
- **I18n and L10n**
- **Messages**
- **Conditions**
- **Context**

## Data

The **View** can refer to data and view state in the **ViewModel** `data` section.

The braces {{ }} allows declarative binding to the live data


Siemens Web Framework
Unpublished work. © 2025 Siemens




Here the "searchName" property defined in the ViewModel is being bound with the "aw-widget" in the View. Thus, any update to the widget would update the ViewModelProperty as well. Similarly, if the ViewModelProperty is updated, the widget would display the same. This ViewModelProperty is created based on the definition provided in the `data` section.

For "searchName", the ViewModelProperty type would be of type 'STRING', with an empty display name of the property and the property will be marked as 'required' which means that the property's value is required for successful completion of the operation. Please refer **ViewModelProperty** for further information on ViewModelProperty.

## Fields

Fields are a new type of JavaScript object that has been introduced to be able to deal with the **changes to data mutations** introduced with version 5 of Siemens Web Framework.

In particular, they are useful (and necessary) whenever the value of a property defined in a view model needs to be modified by a child component, for example, every time that a *property widget* like `aw-widget` or `aw-textbox` etc., is used.

Essentially, a field is an object containing:

- all the required attributes for a given *property widget* that will be passed to its underlying `input` or `textarea` HTML element.
- a derived Subset of the read-only data of an individual **ViewModelProperty** instance providing backing store to the HTML input element attributes.

- callback functions responsible for mutating the ViewModelProperty state (e.g., `onChange`, `onBlur`, `onClick`) to support **inverse data flow**.
- An `update` function able to safely mutate the value of the ViewModelProperty and other properties.

Additionally, a field provides a direct link between a **ViewModelProperty** and metadata to the framework underlying **form validation** engine.

### Field properties

Fields contain a lot of information, some of which is for internal framework use. The most important properties of a field object are the following:

- `name` the internal name of the property
- `onChange` a callback method that takes an event, triggered on change. The default implementation is to update the ViewModelProperty value and also validate the field.
- `onBlur` a callback method that takes an event, triggered on blur. It updates the `setTouched` component state and triggers field validation.
- `onClick` a callback method that takes an event, triggered on mouse left-click.
- `update` a function that can be called to update the underlying ViewModelProperty object.
- `value` the current value of the property.
- `label` the current property display name property
- `typex` the current property type (i.e. STRING, BOOLEAN, INTEGER, etc.)
- `error` the current field validation error, if any.
- `required` whether the property is required or not.
- `maxLength` the maximum length of the property value.
- `placeholder` the placeholder text to display in input elements.
- `disabled` whether the property is disabled or not.
- `dirty` whether the property value has been modified or not.
- `aria-label` the display name of the property.
- `vmo` a reference to the ViewModelObject containing the property (if any).
- `fielddata` additional field data.

### Field creation

Because fields provide the ability to update the value of a property from a child component, they can only be created by the framework at specific times during the component lifecycle.


Siemens Web Framework
Unpublished work. © 2025 Siemens

In other words, there is no programmatic way to create a field object, but a field object is created *automatically* whenever:

- A ViewModelProperty is defined directly under the `data` section of a ViewModel.
- A ViewModelObject is defined directly under the `data` section of a ViewModel. In this case, a field is created for each ViewModelProperty contained in the ViewModelObject.
- A list of ViewModelProperty objects is defined directly under the `data` section of the ViewModel.
- A field has been specified within a complex object by defining **atomic data**.

### Field update

An `update` method is present on every field object and it can be used to safely mutate the value or other properties of the underlying ViewModelProperty object, even from child components. The `update` method also provides a way to trigger validation and mark a field as modified.

Consider the following example:

```
export const updateNameField = ( nameField, newValue ) => {
    // nameField.update( valueToBeUpdated, ViewModelProps , formProps );
    nameField.update( newValue, { isRequired: true }, { markModified :
false, runValidation : true } );
};
```

In this case, the `updateNameField` function can be used to safely mutate a field object (`nameField`) and:

- Change its value to `newValue`
- Set the `isRequired` flag to `true`.

Note that the second parameter of the `update` method is VMP Props and can be set to an object containing the ViewModelProperty to be changed and their new value (like `isRequired` in the previous example).

Note that the third parameter of the `update` method is form props which is use to set form/field properties i.e. `markModified` will update field with blue background and updates form as dirty if provided as `true` and `runValidation` will run field level validation and updates form's validity based on outcome.

Default values for `markModified` is false and `runValidation` is true.

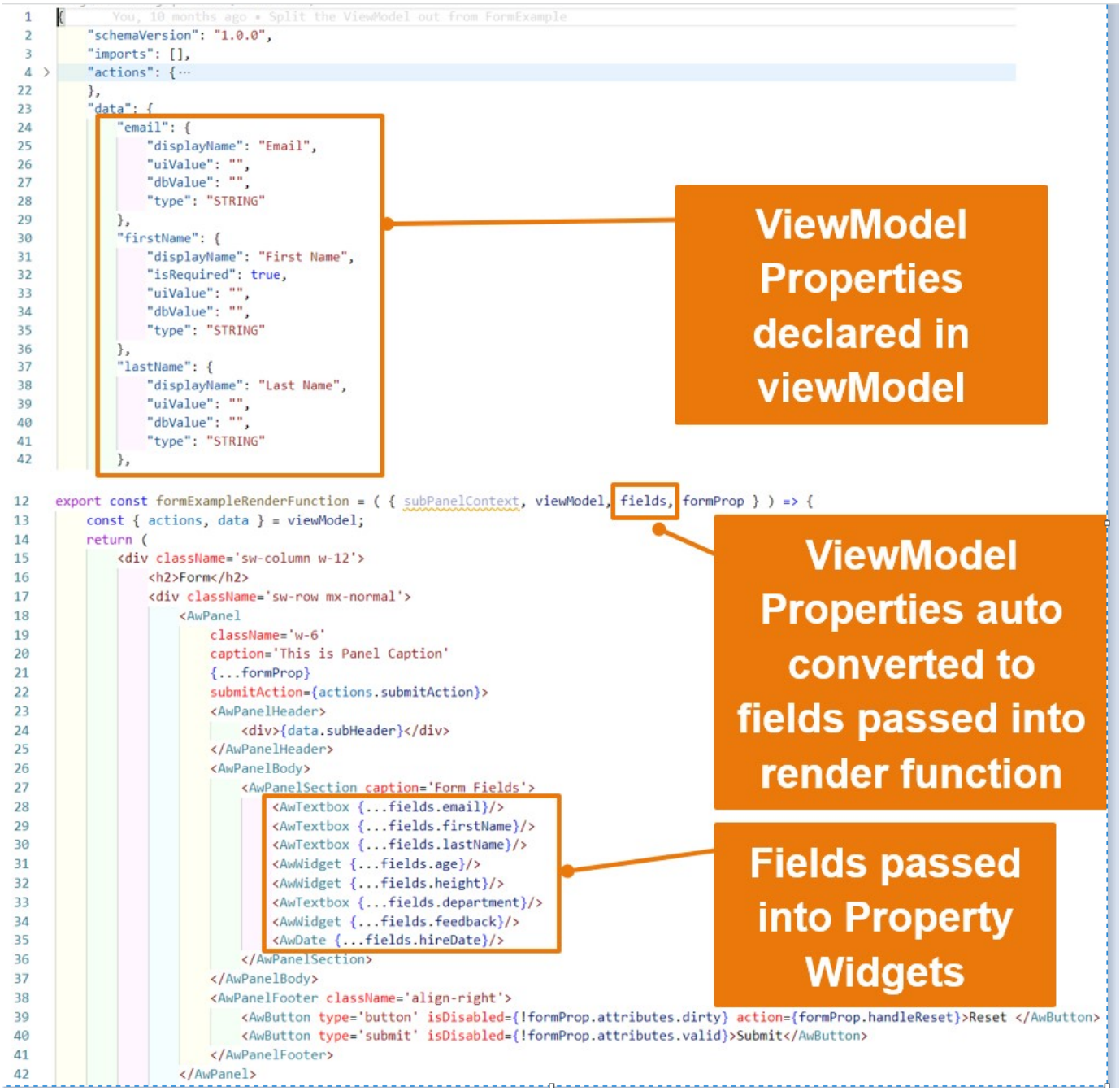
Truth Table

| markModified | runValidation | Outcome |
|---|---|---|
| true | true | Field becomes dirty(blue background), Form becomes dirty, Field is validated, Form is validated |
| false | true | Field does not become dirty(blue background), Form does not become dirty, Field is validated, Form is validated |
| true | false | Field becomes dirty(blue background), Form becomes dirty, Field is not validated, Form is not validated |
| false | false | Field does not become dirty(blue background), Form does not become dirty, Field is not validated, Form is not validated |

### Usage

Fields that have been automatically created within a view model will be exposed in a `fields` property of the object passed to the component render function:


Siemens Web Framework
Unpublished work. © 2025 Siemens



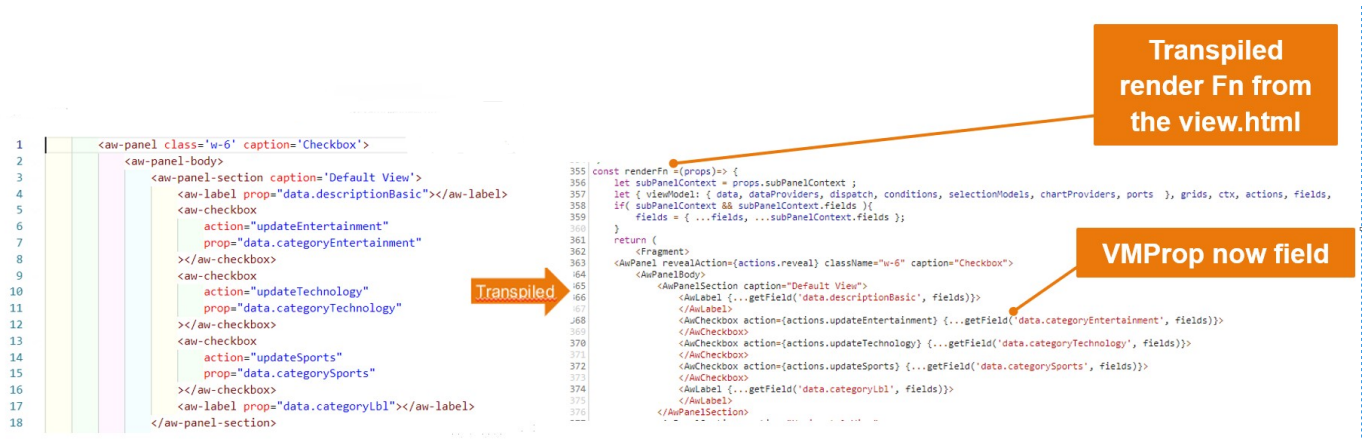

Additionally, the view of a component is transformed into a render function by the view compiler, `data.*xyz*` references will be *automatically converted* to the corresponding field object if the data binding of the component is set to `field`.

Consider the following example view model for `aw-checkbox`:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
```

```
        "render": {
            "method": "awCheckboxRenderFunction",
            "deps": "js/AwCheckboxService"
        }
    },
    "data": {},
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    },
    "props": {
        "action": {
            "type": "action"
        },
        "prop": {
            "type": "field"
        },
        "is-negated": {
            "type": "object"
        }
    }
}
```

In this case, the binding of `prop` property is of type `field`. In this case, the view compiler knows that the value specified for that property must be converted to a field:




**Note**

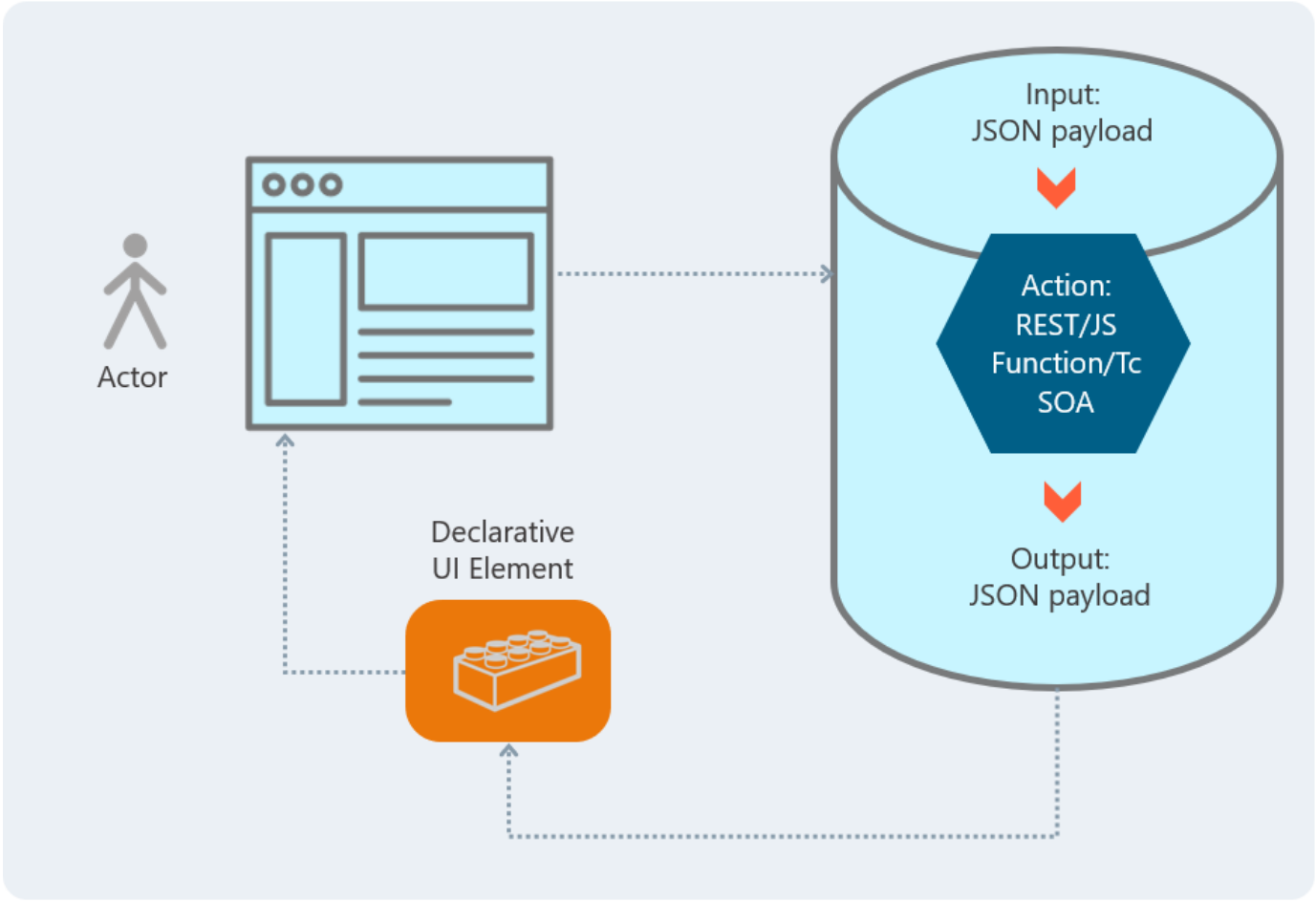
Although the view compiler automatically retrieves fields from the specified data references, you can also directly specify fields.*xyz* to access the list of fields created within a specific view model.


Unpublished work. © 2025 Siemens

# Actions

*Actions* are payloads of information that send data from your application to your store/back-end. Siemens Web Framework let developers define `Actions` declaratively inside the **ViewModel**. Siemens Web Framework supports different type of actions (REST, JSFunction, JSFunctionAsync, Event or TcSoaService) through attribute `actionType`. All actions have `inputData` which specifies the JSON input for the action and `outputData` for mapping the output response.

Actions can be used in **dataproviders** or **UI elements** such as an aw-button.




## Action Schema - Valid attributes

- `actionType` - RESTService, **JSFunction**, **JSFunctionAsync**, Event, **Navigate** & TcSoaService
- `inputData` - JSON data for the action input
- `outputData` - JSON data for the action output
- `events` – **events** triggered in response to the execute of the action
- `actionMessages` – display user **messages** after the execute of the action, conditions support


Unpublished work. © 2025 Siemens

- `dataParsers` (Optional)- To convert the response objects into objects consumable by Siemens Web Framework UI Elements i.e. maps raw JSON output from action response to ViewModelProperty Objects

## Examples

### Example of RESTService GET method call

```
"search": {
    "actionType": "RESTService",
    "method": "GET",
    "inputData": {
        "request": {
            "method": "GET",
            "withCredentials":false,
            "url": "https://some.rest.service.provider/xyz"
        },
    }
    "dataParsers": [ {
        "id": "{{nasa}}",
        "responseObjs": "data.path.to.objects"
    } ],
    "outputData": {
        "searchResults": "data.path.to.objects"
    }
    "events": {
       "success":  [ {
            "name": "complete",
            "eventData": {
                "source": "toolAndInfoPanel"
            }
        } ]
    },
    "actionMessages": {
       "success":[
         {
             "message": "invalidSearchString",
             "condition": "data.movieFound==='False'"
         }
         ]
     }
},
```

### Example of RESTService POST method call

```
"loginAction": {
            "actionType": "RESTService",
            "inputData": {
                "request": {
                    "method": "POST",
                    "withCredentials": false,
```


Unpublished work. © 2025 Siemens

```
                    "url": "assets/php/auth.php",
                    "data": { "username": "{{data.username.dbValue}}",
"password": "{{data.password.dbValue}}" }
                }
            },
            "outputData": {
                "loginResponse": "data"
            }
        }
```

Notice you need to pass `data`

### Example of JSFunction

```
"actions": {
    "getCountryList": {
        "actionType": "JSFunction",
        "method": "getCountryList",
        "inputData": {
            "data": "{{data.filterBox.dbValue}}"
        },
        "outputData": {
            "searchResults": "countries",
        },
        "deps": "js/searchCountryService"
    }
}
```

Service returns countries.

### Example of JSFunctionAsync

```
"actions": {
    "getCountryListAsync": {
        "actionType": "JSFunctionAsync",
        "method": "getCountryListAsync",
        "inputData": {
            "data": "{{data.filterBox.dbValue}}"
        },
        "outputData": {
            "searchResults": "countries",
            "totalFound": "totalFound"
        },
        "deps": "js/searchCountryService"
    }
}
```

Returns a promise with resolved countries and totalFound.

### Example of Event

```
"actions": {
        "anEventAction": {
            "actionType": "Event",
            "method": "Event",
            "inputData":
            {
                 "events":
                 [
                     {
                         "name": "eventNameOne",
                         "eventData":{"someKey":"someValue"},
                         "excludeLocalDataCtx":true
                     },
                    {
                         "name": "eventNameTwo",
                         "eventData":{"someKey":"someValue"},
                         "excludeLocalDataCtx":true
                     }
                 ]
            }
        }

    }
```

**Note**

From **Siemens Web Framework v4.2.0** onwards, object conditions and expressions are supported on action messages and events.

### See also

- Data Parser
- Custom Actions
- Batch Action

## JSFunction action

When you see a ActionType : JSFunction, it maps to your typical javascript function. If there are sequence of instructions needs to be executed independently with a set of specific input or no input at all, one can define Action of type JSFunction and map it to javascript function. It can be think of as js function where only object evaluation or modification is done synchronously.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Example - ViewModel**

```
    "actions": {
        "buttonAction": {
            "actionType": "JSFunction",
            "method": "buttonAction1",
            "inputData": {
                "data": "Button1"
            },
            "outputData": {
                "message": "message"
            }
            "deps": "js/showcaseButtonActions"
        }
    }
```

**Example - JavaScript Service**

```
    export let buttonAction1 = function( data ) {
        var message =  data + ' has been clicked'
        return message;
    };
```

Whatever needs to be mapped on `outputData` of action must be returned.

## JSFunctionAsync action

In JavaScript we have a concept of asynchronous programming i.e. where you don't want a piece of work to block the execution flow and instead it is okay that it runs in parallel to main flow. Once the parallel work is done, one can post process the end result. Usually in such a situation post processing is done with `.then` block.

Example :

```
    export let getCountryListAsync = function( filterString ) {
        var deferred = AwPromiseService.instance.defer();
        console.log('Start Async Action');
        if( valid ) {
            AwHttpService.instance.get( 'assets/test/staticresponses/
listFilterResponse.json' ).then( function( response ) {
                console.log('Execute Async Action');
                var result = response.data.countries.filter( filterString );
                return deferred.resolve( {
                    countries: result,
                    totalFound: result.length
                } );
            } );
        } else {
```

```
            deferred.resolve( { invalid: true } );
        }
        console.log('Do something here');
        return deferred.promise;
    };
```

Here `getCountryListAsync` is an async call which will be independent of the main thread. Once getCountryListAsync method is invoked , “Start Async Action” will be printed. It would next initiate the async action but wouldn't wait for it to execute. Instead 'Do something here' is printed. Only once the promise is resolved, an Async call will be completed and note that “Execute Async Action” is printed on console. Piece of work which is dependent on response are usually found in `.then` callback function.

A promise must be returned as done at last and response that needs to be mapped in outputData of action shall be wrapped in promise's resolve.

The above use case is of single async action. If you have 'n' number of async action then you must return each async call and resolve in last one as follows :

Example for chain of Async Actions

```
export let uploadFiles =  function( fromData ) {
    const url = 'custom url';
    let deferred = AwPromiseService.instance.defer();
    AwHttpService.instance.post( url, fromData, {
        headers:{
            'Content-Type': 'multipart/form-data'
        }
    } ).then( response => {
        // custom logic to process the response logic
    } ).then( result => {
            deferred.resolve( result );
    } ).catch( error => {
        deferred.reject( error );
    } );
    return deferred.promise;
};
```

All async actions should return promise and what ever needs to map in output data should be mentioned in resolve.

## Navigate action

You can navigate to a specific page (state) within running application or an external website (URL) using an action of the type `Navigate`.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters supported with Navigate actionType

- `actionType:` It has to be `Navigate`
- `navigateTo:` URL either refers to state, or refers to fully qualified URL pointing to outside website
- `navigationParams:` Parameters passed on to URL
- `navigateIn:` Whether navigate to new tab, new window and refresh same page. Valid options a) newWindow, b) newTab
- `options:` (optional) While navigating to the new window, attributes for the window can be provided as “options”.

### Example#1 - Navigate in new tab

The attribute `navigateIn` needs to be set to `newTab`

```
"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"
}
```

### Example#2 - Navigate in new window

The attribute `navigateIn` needs to be set to `newWindow`. Optionally you can provide optional parameters {(X, Y) coordinates for new window)} via attribute `options`.

```
"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
    }
}
```

Let's understand more on how to use Navigate action type to solve real world use-cases

#### Scenario#1 - Navigate to a specific page (state) within application

Let's say you want to define a command, say `Awp0ShowObject` which would take user to a page (state), say `com_siemens_splm_clientfx_tcui_xrt_showObject`.

#### 1. Associate a command handler for `Awp0ShowObject` command

When user clicks on `Awp0ShowObject` command, the command handler shall triggering the action specified on handler, in this case it's `showObject`

```
 "commandHandlers": {
      "Awp0ShowObjectCommandHandler": {
          "id": "Awp0ShowObject",
          "action": "showObject"
      }
  }
```

#### 2. Define `showObject` action of the type Navigate

```
  "actions": {
      "showObject": {
          "actionType": "Navigate",
          "navigateTo": "com_siemens_splm_clientfx_tcui_xrt_showObject",
          "navigationParams": {
              "uid" : "{{ctx.selected.uid}}"
          }
      }
  }
```

#### Navigating to an external website (URL)

Let's say you want to define a command, say `Awp0ShowHelp` which would take user an external website

#### 1. Associate a command handler for `Awp0ShowHelp` command

#### 2. Define `showHelp` action of the type Navigate

```
"showHelp": {
    "actionType": "Navigate",
    "navigateIn": "newTab",
    "navigateTo": "https://docs.plm.automation.siemens.com/tdoc/aw/4.1/
aw_html_collection#uid:index",
    "navigationParams": {}
}
```


Unpublished work. © 2025 Siemens

## Popup Action

You can configure an action of type **popup** to display a custom component or view in a popup and customize the behavior by specifying an `options` parameter in **inputData** that must be set to a PopupOptions, as shown in the following example.

### Example

```
{
  "actions": {
    "displayPopup": {
      "actionType": "popup",
      "inputData": {
        "options": {
          "view": "popup",
          "isModal": false,
          "clickOutsideToClose": false,
          "draggable": true,
          "placement": "bottom-start",
          "caption": "Caption",
          "anchor": "closePopupAnchor",
          "preset": "modal"
        }
      },
      "outputData": {
        "popupId": "id"
      }
    }
  }
}
```

**Note**

You must wrap the initiator of the popup in an **aw-popup-wrapper** component. For more information and examples, refer to Storybook.

**Tip**

For more information for migrating your code if you are using the old `popupService` API, see New popup action.

## Dialog Action

You can configure an action of type **dialog** to display a custom component or view in a dialog and customize the behavior by specifying an `options` parameter in **inputData** that must be set to a DialogOptions, as shown in the following example.


Unpublished work. © 2025 Siemens

### Example

```
{
    "actions": {
        "openPushDialogActionFromRight": {
            "actionType": "dialog",
            "inputData": {
                "options": {
                    "view": "popup",
                    "caption": "Push Dialog",
                    "width": "STANDARD",
                    "height": "FULL",
                    "parent": ".sw-custom-dialogContainer",
                    "placement": "right",
                    "push": true
                }
            }
        }
    }
}
```

**Tip**

For more information and examples, refer to **Storybook**.

## Command Action

You can configure an action of type **Command** to execute a command programmatically. The specified command will get executed if it is visible and enabled.

### Example

```
{
    "actions": {
        "invokeCommandProgramatically": {
            "actionType": "Command",
            "inputData": {
                "commandId": "Tkr0AddCmd",
                "commandContext": {
                    "customState": "{{props.customState}}"
                }
            }
        },
    }
}
```

In this case, `Tkr0AddCmd` is the example command which needs to be executed when the `invokeCommandProgramatically` custom action is invoked.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Command** actions take the following arguments as `inputData`:

| Arguments | Description | Mandatory |
|---|---|---|
| commandId | The command Id of the command to be executed | Yes |
| commandContext | Any additional information which needs to be passed to command for its execution | No |

## Batch action

A *Batch Action* is group of actions executed in sequence. The batch action executes in the exact sequence defined in the configuration and can be configured to pass data from one action to another.

### Defining a Batch action

You can define new batch action in `actions` object your ViewModel, using the following syntax:

**syntax of 'batch action'**

```
{
  "actions": {
    "BatchActionName": {
      "actionType": "batchJob",
      "steps": [
        {
          "action": "NameOfAction1",
          "outputArg": [
            "outArg1",
            "outArg2",
            "outArg3"
          ]
        },
        {
          "action": "NameOfAction2",
          "inputArg": {
            "inArg1": "{{actionData.outParam1}}",
            "inArg2": "{{actionData.outParam2}}"
          },
          "condition": "conditions.nameOfCondition"
        }
      ]
    }
  }
}
```

- In above syntax, your batch action is `BatchActionName`, the `actionType` is set to `batchJob`, and the `steps` property is an array of objects, each containing the following properties:


Unpublished work. © 2025 Siemens

- `action` : Name of an action within the same ViewModel *(mandatory)*.
- `outputArg` : Array of strings which is mapped with the `outputData` of the action in sequence and forwarded to the next action *(optional)*.
- `inputArg` : An object to configure the input arguments of the action (you can use `actionData` to reference the `outputData` of the preceding action) *(optional)*.
- `condition` : The action will be executed or skipped based on the value of the specified condition *(optional)*.

```
From Siemens Web Framework v4.2 onwards, condition expressions are
supported in batch action, as shown in the following example:
```

```
{
  "actions": {
    "BatchActionName": {
      "actionType": "batchJob",
      "steps": [
        {
          "action": "NameOfAction",
          "condition": "ctx.sublocation.clientScopeURI === 'Awp0Actions'"
        }
      ]
    }
  }
}
```

**Example**

The following is a complete example of two Batch Actions with all their referenced actions:

```
{
  "actions": {
    "loadShowcaseJsonBatch": {
      "actionType": "batchJob",
      "steps": [
        {
          "action": "getDataFromServer",
          "outputArg": [
            "dataFromServer"
          ]
        },
        {
          "action": "processReceivedData",
          "inputArg": {
            "rawData": "{{actionData.dataFromServer}}"
          }
        }
      ]
    },
```


Unpublished work. © 2025 Siemens

```
    "getDataFromServer": {
      "actionType": "RESTService",
      "method": "POST",
      "request": {
        "method": "POST",
        "url": "http://www.omdbapi.com/",
        "i": "tt3313686",
        "plot": "full",
        "r": "json"
      },
      "outputData": {
        "showcaseJson": "data.showcaseJson"
      }
    },
    "processReceivedData": {
      "actionType": "JSFunction",
      "method": "processRawData",
      "parameters": {
        "rawData": "{{result.showcaseJson}}"
      },
      "inputData": {
        "showcaseJson": "{{parameters.rawData}}"
      },
      "deps": "js/showcaseButtonActions"
    },
    "executeFileOperation": {
      "actionType": "batchJob",
      "steps": [
        {
          "action": "getFileTicket"
        },
        {
          "action": "uploadRawFile"
        }
      ]
    },
    "getFileTicket": {
      "actionType": "RESTService",
      "method": "GET",
      "inputData": {
        "request": {
          "method": "GET",
          "url": "https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/
photos?name=docPOP"
        }
      },
      "outputData": {
        "searchResults": "data.ticketId"
      }
    },
    "uploadRawFile": {
      "actionType": "RESTService",
      "inputData": {
        "request": {
          "method": "POST",
          "url": "{{Constants.fmsUrl}}",
```

```
            "headers": {
              "Content-type": "undefined"
            },
            "data": "{{data.formData}}"
          }
        }
      }
    }
}
```

In this case, two Batch Actions have been defined:

1. `loadShowcaseJsonBatch`: It will execute `getDataFromServer` and `processReceivedData` in sequence, passing some data.
2. `executeFileOperation`: It will execute `getFileTicket` and `uploadRawFile`.

If one of the referenced actions fails, the failed response is available in the subsequent actions condition context through the `response` object, and it can be used in condition expressions like this:

```
{
  "conditions": {
    "validateTheData": {
      "expression": "response.actionReponse !== 'executed successfully'"
    }
  }
}
```

### Why events are not supported in batch action

The actions referenced by a Batch Action are executed in a sequence. The output of one action could be passed to next action in a batch Job sequence.

Note that it does not matter whether the action defined in the steps of Batch action is a `JSFunction` or `JSFunctionAsync`: if an action follows an asynchronous action, it will wait for its completion before being executed, and the Batch Action completes when all the actions referenced in its steps are executed.

If events were supported inside a Batch Action, then an event could invoke another action which could possibly be another Batch Action as well. This may result in of a chain of Batch Actions resulting into an uncertainty around the sequence in which the actions will get executed. To avoid this, events are not supported in Batch Actions.

This validation that a Batch Action cannot fire events is captured by Siemens Web Framework **auditing script** and reported as an error.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### *dataProvider* Actions

Actions of type `dataProvider` are treated in a particular way, as these action eventually update the component state and trigger a re-render. For this reason, if a Batch Action is comprised of one or more `dataProvider` actions, the other actions will *not* wait until the `dataProvider` actions are completed.

In the following example, `userAction` will not wait till the `initDataProvider` action is completed. Actions of type `dataProvider` are used to initailize a dataprovider, therefore `userAction` will be invoked once the corresponding dataProvider instance is created.

```
{
  "actions": {
    "initDataProvider": {
      "actionType": "dataProvider",
      "method": "dataProviderName"
    },
    "userAction": {
      "actionType": "JSFunction",
      "method": "sampleMethod",
      "deps": "js/sample"
    },
    "batchAction": {
      "actionType": "batchJob",
      "steps": [
        {
          "action": "initDataProvider"
        },
        {
          "action": "userAction"
        }
      ]
    }
  }
}
```

### Example in Storybook

See the `aw-button` example in **Siemens Web Framework Storybook**.

## Custom action

Custom actions are user-defined actions. They are the extensions of the Siemens Web Framework's actions i.e developers can write their new actions on the top of basic Siemens Web Framework's actions or on their own pre-defined custom actions.

### Defining custom actions

You need to define a template file (i.e. JSON file) for your actions named 'actionTemplateDefs.json' in their module. Inside 'actionTemplateDefs.json' you need to provide your custom action definition.

#### Example `actionTemplateDefs.json`

```
"StartEditAction": {
        "actionType": "JSFunction",
        "method" : "execute",
        "inputData": {
            "handleContext": "{{inputData.myHandleContext}}",
            "view": "{{inputData.myView}}"
        },
        "deps":"js/startEditGroupCommandService"
    }
```

- In above example, your custom action is 'StartEditAction'. you are building your custom action on the top of Siemens Web Framework's basic action i.e. 'JSFunction', so you need to follow the 'JSFunction' schema format.

### How to use Custom Action in ViewModel JSON

- You can use your defined custom 'StartEditAction' as an action type like other action types as below.

```
"actions": {
        "startEditSummary": {
            "actionType": "StartEditAction",
            "inputData": {
                "myHandleContext": "TABLE_CONTEXT",
                "myView": "{{ctx.ViewModeContext.ViewModeContext}}"
            }
        }
}
```

### Schema Validation

- Since, you have defined your custom action template, when you run your audit, it will fail as it doesn't know about new custom actions. So, you need to define a 'declarativeCustomActionTemplateDefsSchema.json' inside the schema folder of your module.
- We are using ajv 6.0.0, so you need to follow below pattern inside custom action template schema.

#### Example of `declarativeCustomActionTemplateDefsSchema.json`

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "description": "This is the schema the declarative Custom Action
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
Contract (in JSON format) must adhere to. The Siemens Web Framework audit
check validates all Custom Action Contract JSON files against this schema.
For explanation of the keywords of this schema, please visit https://
ajv.js.org/keywords.html",
    "definitions": {
        "actionDef": {
            "oneOf": [ {
                    "if": {
                        "properties": {
                            "actionType": {
                                "const": "StartEditAction"
                            }
                        }
                    },
                    "then": {
                        "$ref": "#/definitions/StartEditActionDef"
                    },
                    "else": false
                }
            ]
        },
        "StartEditActionDef": {
            "type": "object",
            "description": "Actions can be of different actionType (REST,
JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
            "additionalProperties": false,
            "required": [
                "actionType"
            ],
            "properties": {
                "actionType": {
                    "type": "string",
                    "description": "start edit",
                    "const": "StartEditAction"
                },
                "inputData": {
                    "type": "object",
                    "description": "JSON data for the action input. For ex.
'request' header can be the inputData in case of 'RESTService' actionType
with method 'Get'."
                },
                "outputData": {
                    "type": "object",
                    "description": "JSON data for the action output. As
part of this, the result of the action is mapped to view model data."
                },
                "events": {
                    "$ref": "#/definitions/actionMessagesDef",
                    "description": "Events can be fired and listened/
handled in declarative View Models. They are a means of a View Model
communicating with other View Models or UI elements. 'Actions' can fire
events."
                },
                "actionMessages": {
```


Unpublished work. © 2025 Siemens

```
                    "$ref": "#/definitions/actionMessagesDef",
                    "description": "User messages. It can be made condition
based"
                }
            }
        }
    },
    "type": "object",
    "additionalProperties": false,
    "patternProperties": {
        "\\b[A-Za-z]([A-Za-z0-9]*)*\\b": {
            "$ref": "#/definitions/actionDef"
        }
    }
}
```

Things you need to write are:

- inside 'oneOf', make an entry for your custom Action (i.e. "const": "StartEditAction")
- provide and define a reference for your custom Action for properties validations (i.e. "$ref": "#/definitions/StartEditActionDef")

Basic support provided by framework to all action types are:

```
"outputData": {
    "type": "object",
    "description": "JSON data for the action output. As part of this, the
result of the action is mapped to view model data."
},
"events": {
    "$ref": "#/definitions/actionMessagesDef",
    "description": "Events can be fired and listened/handled in declarative
View Models. They are a means of a View Model communicating with other View
Models or UI elements. 'Actions' can fire events."
},
"actionMessages": {
    "$ref": "#/definitions/actionMessagesDef",
    "description": "User messages. It can be made condition based"
}
```

- You are flexible to include them inside your schema for validation.

### Example in Storybook

aw-balloonpopup example in **Siemens Web Framework Storybook**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Functions

It is a construct of View model. It has 3 properties:

- `functionName` The name of the function. This is a mandatory field and it has to be string.
- `parameters` An array of parameters, which may contain expressions.
- `deps` The JavaScript file which contains the definition of the functionName.

## Example

```
"functions": {
    "getMovieListUrl": {
        "functionName": "getMovieListUrl",
        "parameters": [
            "{{data.filterBox.dbValue}}",
            "{{data.dataProviders.performSearch}}"
        ],
        "deps": "js/getMovieListService"
    },
    "getSortedList": {
        "functionName": "getSortedList",
        "parameters":[
            "{{ctx}}",
            "{{data.filterBox.dbValue}}",
            "true"
        ],
        "deps": "js/getMovieListService"
    }
}
```

**Note**

The function definition should exist in the same javascript dependency that is referred by the action. Referring to the example above, *getSortedList* function should exist in *getMovieListService*.

## Where can you use functions in a ViewModel?

Functions can be used in:

- InputData
- OutputData
- DataParseDefinition

## How do you use functions within a ViewModel?

### InputData

In the action input user can directly use JSON data or indirectly through output of the function which should retuen JSON data.

```
"actions": {
    "search": {
        "actionType": "RESTService",
        "serviceName": "",
        "method": "GET",
        "inputData": {
            "request": {
                "method": "GET",
                "url": "{{function:getMovieListUrl}}",
                "withCredentials": false,
                "headers": {
                    "Content-type": "undefined"
                },
                "data": ""
            }
        },
        "outputData": {
            "totalFound": "data.searchResults.length",
            "searchResults": "data.Search",
            "movieFound": "data.Response"
        }
    }
}
```

### OutputData

JSON data or function is used for the action output also. When the function is used in outputData make sure the function declaration is present in the same view model then its parameter definition will be taken into account. Moreover, when function is used in `outputData`, the first argument is the response and rest of it are parameters.

```
"actions": {
    "search": {
        "actionType": "RESTService",
        "serviceName": "",
        "method": "GET",
        "inputData": {
            "request": {
                "method": "GET",
                "url": "{{function:getMovieListUrl}}",
                "withCredentials": false,
                "headers": {
                    "Content-type": "undefined"
                },
                "data": ""
            }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        },
        "outputData": {
            "totalFound": "data.searchResults.length",
            "searchResults": "{{function:getSortedList}}",
            "movieFound": "data.Response"
        }
    }
}
```

### DataParseDefinition

It contains the parser definitions. Each parser definition must define a `props` property. Each prop, which is of a specific JSON structure, is converted to a ViewModelProperty. Function is also used in `value` and `displayValue` properties.

```
"dataParseDefinitions": {
    "nasa": {
        "props": {
            "roverName": {
                "name": "roverName",
                "displayName": "Rover",
                "value": "{{function:getRoverName}}",
                "displayValue": "{{function:getRoverName}}",
                "type": "STRING"
            }
        }
    }
}
```

## Data providers

A Data Provider, as the name suggests, is a single placeholder where we configure different ways to fetch data and pass the same to underlying component. It is basically an abstraction layer for components which fulfils component's demand to load/paginate data from the server.

### Components which use dataProvider

In Siemens Web Framework, dataProvider is used with following components to meet their data requirements.

1. aw-list (Please refer to `aw-list` example in **Siemens Web Framework Storybook**)
2. aw-splm-table (Please refer to `aw-splm-table` and `aw-splm-table tree` example in **Siemens Web Framework Storybook**)
3. aw-listbox (Please refer to `aw-listbox` example in **Siemens Web Framework Storybook**)
4. aw-widget with LOV (Please refer to `List under property widgets` example in **Siemens Web Framework Storybook**)

## Configuring a simple dataProvider

1. To configure a dataProvider, we need to define an empty dataProvider section in relevant xxxViewModel json.

```
dataProviders: {

}
```

1. Add a dataProvider say "NasaDataProvider" inside dataProviders section

```
dataProviders: {
   "NasaDataProvider": {
   }
}
```

1. Add few necessary attributes in the NasaDataProvider

```
dataProviders: {
   "NasaDataProvider": {
      "action": "fetchData",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}"
   }
}
```

`action`: Represents one of the artifacts('fetchdata') defined in actions section in the same view-model. In the 'fetchdata' artifact, we can define different ways to load the data from server. For more information, refer to the next section DataProvider and its Actions.

`response`: Data returned from the server. Must be an array containing view-model objects.

`totalFound`: This length of the data. This flag is mainly used for pagination. More on pagination later.

Below is a sample of dataProvider

```
   "dataProviders":
    {
        "imageDataProvider":
        {
            "action": "loadData",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}"
        }
    },
    "actions":{
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "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}}"
            }
        }
    }
```

## DataProvider and its Actions

In addition to `action`, the Data Provider supports other actions to facilitate loading and pagination.

1. `initializeAction`: Loads the first page of results. For tables and lists, configuration of this action is optional. If initializeAction is not configured for list or table, the dataProvider will instead use its "action" to load the first page of results. For trees, `initializeAction` is required to load the first page of data.
2. `nextAction`: This action is invoked when the scrollbar reaches the bottom most position of the dataset. nextAction will then load the next page of data. This facilitates pagination. When the user scrolls to the bottom most position of the dataset and there are more items to reveal, the dataProvider will call either (a) nextAction if it is configured or (b) "action" if there is no valid "nextAction"
3. `previousAction`: This action is invoked when the user scrolls to the top scrollbar position of the dataset, if there is more data above to be revealed. When invoked, this action will load the data above the current top most position.
4. `focusAction`: This action is used to handle cases where there is some search or filter applied to the dataset. When this action is invoked, it will reveal the row or object which is supposed to get the focus."focusAction" handles cases where some object is selected that is not currently loaded in the dataProvider.

`previousAction` and `focusAction` are codependent. When the focusAction is invoked, it directs the list or table to the selected item based on the search or filter applied by the user. If there are more results above the focused object in the dataset, previousAction will be invoked to load the items above when the scrollbar reaches the topmost position. If no focusAction is defined in your ViewModel, previousAction is not required.

For more information on working with Data Provider in Tables and Trees, refer to **Working with Tables**

## DataProvider types

There are basically three types of data-provider

1. Static
2. Action
3. TcLOV ( Specific to Active Workspace.Out of discussion)

## Static DataProvider:

These type of dataProvider can accept a static list of data, which can be configured in "viewmodel" easily.

```
"dataProviders":
   {
       "locationLink":
       {
           "dataProviderType": "Static",
           "response":
           [
              "{{data.Siemens Web Framework}}",
              "{{data.Zeus}}"
           ],
           "totalFound": 2
       }
  },
"data":
   {
       "Siemens Web Framework":
       {
           "displayName": "Siemens Web Framework",
           "type": "STRING",
           "dbValue": "Siemens Web Framework",
           "dispValue": "Siemens Web Framework"
       },
       "Zeus":
       {
           "displayName": "Zeus",
           "type": "STRING",
           "dbValue": "Zeus",
           "dispValue": "Zeus"
       }

   }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Action DataProvider

The commonly used data-provider. The above described "imageDataProvider" is an action type data-provider. We do not require to define "dataProviderType" for action type data-providers.

### Few more configurable parameters

### selectionModelMode:

This flag is used to indicate selection mode in Data Provider. Valid values are "single"/"multiple"/"none". Default is "single".

### commandsAnchor:

This flag is used to specify a command anchor bar. Please refer to `aw-command-bar` example in **Siemens Web Framework Storybook**) and visit **More on Commands Anchor**.

### preSelection:

This flag is used to indicate, if the newly added object will be shown as selected.

```
dataProviders: {
    "sampledataProvider" : {
        "action": "fetchDataFrmServer",
        "response": "response from the action",
        "totalFound" : "length of the data",
        "selectionModelMode": "single/multiple/none, default is single",
        "preselection": "true/false, default is true"
    }
}
```

### preferredChoicesAction

This is an optional action for retrieving preferred list choices for LOV. These preferred choices are bubbled to the top of the list and can be used to show recently used choices in a long list or other more static preferred option. This action populates an object on the dataProvider called preferredChoices. Instead of using the service, a static option can be set on this object directly (see example for aw-listbox in **Siemens Web Framework Storybook**).

### Events fired by Data Provider

1. {

   }.`selectionChangeEvent`: Whenever a object selection change happens in aw-splm-table or aw-list, data-provider gets notified and fires this event with latest selected object.


Unpublished work. © 2025 Siemens

2. {

   }.`modelObjectsUpdated`: data-provider fires this event whenever the underlying view model collection is updated.

3. {

   }.`selectAll`: When all the objects inside data-provider is selected.

4. {

   }.`selectNone`: When all the objects inside data-provider is de-selected.

### Events listened by Data Provider

1. {

   }.`multiSelectModeAction`: data-provider listens to this event to trigger change in active status of multiSelectEnabled flag on respective selection model. It takes `multiSelect` as input event data where *true* will activate multi-select mode, and *false* will deactivate the same.

2. {

   }.`selectAction`: When all the objects inside data-provider is to be selected/deselected at once. It takes `selectAll` as input event data where *true* will select all object and *false* will clear the selection.

```
action: {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [
                    {
                        "condition": "conditions.trueCondition",
                        "name": "yourDataProvider.multiSelectModeAction",
                        "eventData": {
                            "multiSelect": true
                        }
                    },
                    {
                        "condition": "conditions.falseCondition",
                        "name": "yourDataProvider.selectAction",
                        "eventData": {
                            "selectAll": false
                        }
                    },
                    {
                        "condition": "conditions.falseCondition",
                        "name": "yourDataProvider.multiSelectModeAction",
                        "eventData": {
                            "multiSelect": false
                        }
                    }
                ]
```



Unpublished work. © 2025 Siemens

```
            }
        }
```

### How to configure a data-provider as part of some action in view-model

```
actions:{
        "reveal":
        {
            "actionType": "dataProvider",
            "method": "imageDataProvider"
        }
}
```

### How to call(initialise) multiple data-providers as action in view-model

```
"actions":
    {
        "reveal":
        {
            "actionType": "dataProvider",
            "methods": [ "getHistory", "getFavorites", "performSearch" ]
        }
}
```

### How to pass additional data as input to data-provider.

```
    {
        "imageDataProvider":
        {
            "action": "loadData",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "inputData": {
                "someData": "{{ctx.abcd}}"
            }
        }
    }
```

### Selection Model in data-provider

Data-provider comes with a default selection Model. The primary responsibility of a selection model is to keep track of selected objects, which includes

1. Containing list of objects selected inside a data-provider.
2. Keeping the Internal State Information of the selection i.e multi select state, selection mode and selection status.

Here are few basic methods for the selection model.

### Selection Model Mode in data-provider

```
{
    "imageDataProvider":
    {
        "action": "loadData",
        "response": "{{data.searchResults}}",
        "totalFound": "{{data.totalFound}}",
        "selectionModelMode": "multiple/single/none"
    }
}
```

single is the default value for selectionModelMode.

### How to customize selection Model

User can hook-in customized selection model based on the application requirement. This can be done through inputdata in data provider

```
{
    "imageDataProvider":
    {
        "action": "loadData",
        "response": "{{data.searchResults}}",
        "totalFound": "{{data.totalFound}}",
        "inputData": {
            "selectionModel": "{{data.selectionModel}}"
        }
    }
}
```

### How to configure pagination in data-provider

#### Pagination in data-provider

### How to configure a data-provider to be used in aw-splm-table

#### To configure a data-provider to be used with aw-splm-table

### Sorting and Filtering

DataProvider does not encourage sorting and filtering at client side. These heavy processing should take place at server side, rather than at client side..


Siemens Web Framework
Unpublished work. © 2025 Siemens

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 data-provider.

Below is an example for the same.

```
{
    "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://ssd-api.jpl.nasa.gov/sentry.api?
www=1&ps-min=-6&h-max=24",
                    "searchInput": {
                        "searchFilterMap": "{{ctx.activeFilterMap}}",
                        "searchSortCriteria":
"{{data.dataProvider.gridDataProvider.sortCriteria}}",
                        "startIndex":
"{{data.dataProviders.gridDataProvider.startIndex}}"
                    }
                }
            }
        }
    }
}
```

## Dataprovider as edit handler

Data-provider can act as an edit-handler, which can be leveraged to edit tables. Data-provider implements the following methods of edit handler

1. startEdit
2. cancelEdit
3. saveEdits
4. isDirty

5. isEditInProgress

Data-provider as edit handler enables user to hook in their own start/save edit action in declarative manner. It also gives flexibility to write own function to merge the start edit response.

For Configuration details, please follow the link below:

**Dataprovider as editHandler**

### Dataprovider in Batch Action

Batch Action can have a step which is of type dataProvider.

For more details on batch actions, see **this page**.

For an example of a batch action with step of type dataProvider, see the `aw-scrollpanel`` example in **Siemens Web Framework Storybook**.

### Dataprovider in Custom Action

Custom actions can have a step which is of type dataProvider. Also, dataProvider can have actionType set to a custom action.

For more details on custom actions, see **this page**.

For an example of a custom action of type dataProvider see the `aw-scrollpanel` example in **Siemens Web Framework Storybook**.

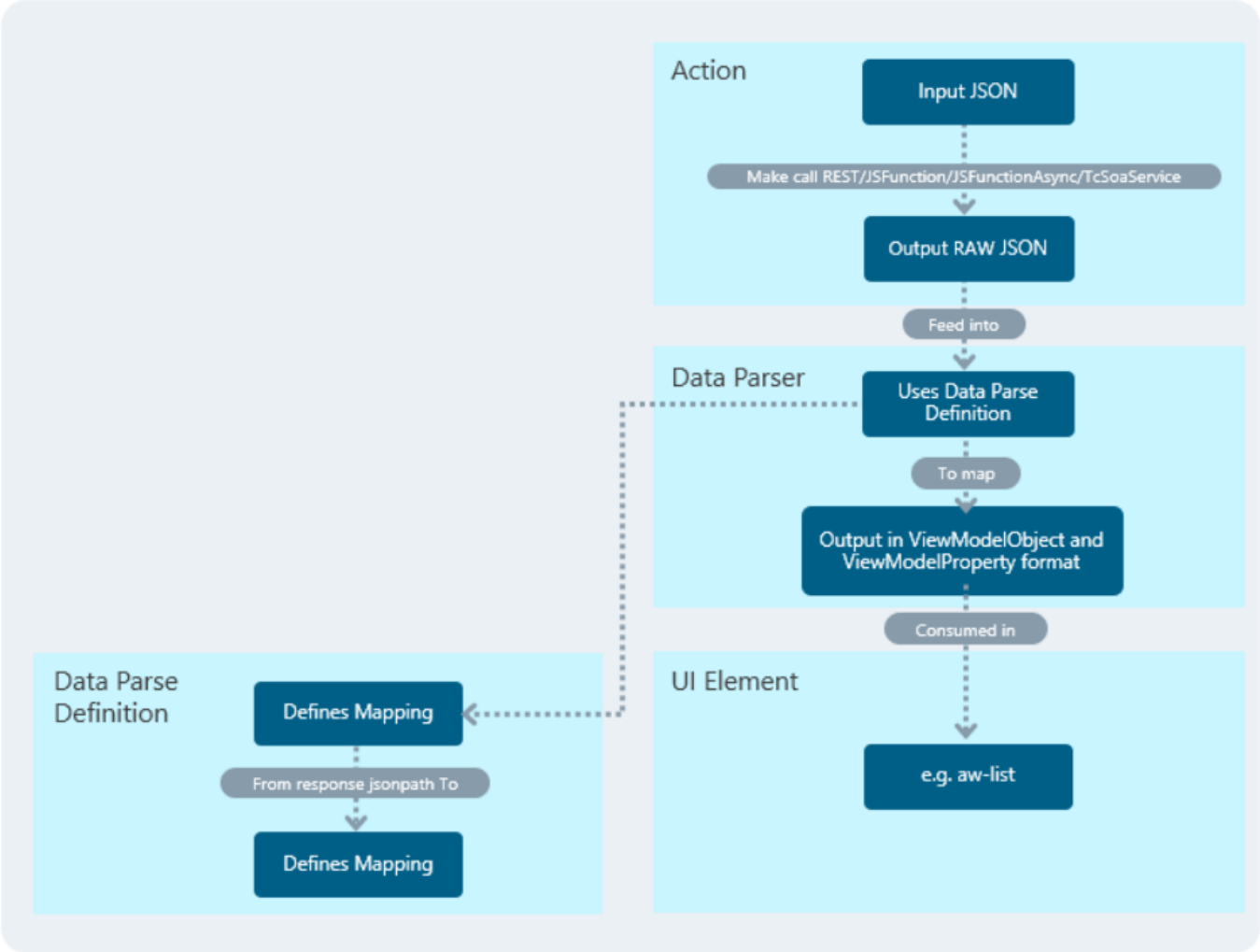
## Data parsers

Data Parsers Convert the response objects into objects consumable by Siemens Web Framework **UI Elements** i.e. maps raw JSON output from action response to `ViewModelObjects` with different `ViewModelProperty`.

It is an array of objects, each having these properties below.

- *id* - specifies which dataParseDefinition to use.
- *responseObjs* - path to the response object(s) to convert. It can be either an array of objects, or a single object.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## RAW response structure from REST call


```
▼{data: {…}, status: 200, headers: ƒ, config: {…}, statusText: "OK", …}
  ▶config: {method: "GET", transformRequest: Array(1), transformResponse: Array(1), paramSerializer: ƒ, jsonpCallbackParam: "callback", …}
  ▼data:
    ▼photos: Array(25)
      ▼0:
        ▶camera: {id: 23, name: "CHEMCAM", rover_id: 5, full_name: "Chemistry and Camera Complex"}
         earth_date: "2012-08-16"
         id: 3132
         img_src: "http://mars.jpl.nasa.gov/msl-raw-images/proj/msl/redops/ods/surface/sol/00010/soas/rdr/ccam/CR0_398380645PRCLF0030000CCAM04010L1.PNG"
        ▶rover: {id: 5, name: "Curiosity", landing_date: "2012-08-06", launch_date: "2011-11-26", status: "active", …}
         sol: 10
        ▶__proto__: Object
      ▶1: {id: 58870, sol: 10, camera: {…}, img_src: "http://mars.jpl.nasa.gov/msl-raw-images/proj/msl/r…edr/ccam/CR0_398381687EDR_F0030000CCAM05010M_.JPG", earth_date: "
      ▶2: {id: 58871, sol: 10, camera: {…}, img_src: "http://mars.jpl.nasa.gov/msl-raw-images/proj/msl/r…edr/ccam/CR0_398381577EDR_F0030000CCAM05010M_.JPG", earth_date: "
      ▶3: {id: 58872, sol: 10, camera: {…}, img_src: "http://mars.jpl.nasa.gov/msl-raw-images/proj/msl/r…edr/ccam/CR0_398381468EDR_F0030000CCAM05010M_.JPG", earth_date: "
```


## dataParseDefinitions

It provides declarative mapping between raw response object and ViewModelObject with associated property called as **ViewModelProperty**. Every viewModelObject should have a unique identifier, in uid, and ideally this should be generated from server side, and mapped in client using the key words `"identifier": "{{id generated from server side }}"` in view model. Each parser definition

must define a 'props' and mapping jsonpath from response to populate property. Each prop is converted to a ViewModelProperty. If 'props' is not defined in parser definition, ViewModelObject will be created without any property.The json structure of each prop is as below.

```
"dataParseDefinitions": {
    "nasa": {
        "identifier": "{{id}}",
        "thumbnailUrl": "{{img_src}}",
        "cellHeader1": "{{rover.name}}",
        "cellHeader2": "{{earth_date}}",
        "cellProperties": {
            "Camera": {
                "key": "Camera",
                "value": "{{camera.name}}"
            }
        },
        "props": {
            "cameraName": {
                "name": "cameraName",
                "displayName": "Camera Name",
                "value": "{{camera.full_name}}",
                "displayValue": "{{camera.full_name}}",
                "type": "STRING"
            },
            "earthDate": {
                "name": "earthDate",
                "displayName": "Earth Date",
                "value": "{{earth_date}}",
                "displayValue": "{{earth_date}}",
                "type": "DATE"
            }
        },
        "indicators": [{
            "image": "{{camera.statusIndicator}}",
            "tooltip": "{{camera.translateStatus}}"
          }]

    }
},
```

- `name` - name of the property. This will be mapped to propertyName of ViewModelObject.
- `displayName` - display name of the property. This can be localized.
- `value` - value of the property. This will be mapped to dbValue for ViewModelObject.
- `displayValue` - display value of property. This can be localized.
- `type` - type of the property. It can hold values like STRING,DATE etc.

**Note**

If the objects need to be shown in aw-default-cell, then the following are optional properties
- `thumbnailUrl` - image returned from REST API which is shown in cell.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- `cellHeader1` - title of cell.
- `cellHeader2` - header of the cell.
- `cellProperties` - other properties to be displayed in cell.
- `indicators` - Array of images with tooltip can be used to visual status on the cell.

## Response structure after data parsing

Note that the original structure is maintained. The “data.photos” array earlier contained an array of raw objects. After conversion, the “data.photos” contains an array of `ViewModelObject` each containing `ViewModelProperties`.Each `ViewModelObject` has only those properties that defined in parser definition and will not derive any additional property. Optionally each object contains some cell related information (e.g.cellHeader1, cellHeader2, indicators, thumbnailUrl etc ).

Created `ViewModelObject` gives following functions to modify viewmodelproperty :

1. getDirtyProps : Gives list of property that are modified.
2. getSaveableDirtyProps : Gives list of property values that are modified.
3. clearEditiableStates : Resets editable state of all property to false.

In the same action, the “data.photos” are assigned to the searchResults in ViewModel.

```
"actions": {
    "reveal": {
        "actionType": "dataProvider",
        "method": "mars"
    },

    "search": {
        "actionType": "RESTService",
        "method": "GET",
        "inputData": {
            "request": {
                "method": "GET",
                "withCredentials": false,
                "url": "https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=10&page=1&api_key=HBSSgn96JKVFVBkukrUV9VH1J4QRhohR4SrcYHoe"
            }
        },
        "dataParsers": [{
                "id": "nasa",
                "responseObjs": "data.photos"
            }
        ],
        "outputData": {
            "searchResults": "data.photos"
        },
        "deps": "js/curiosityPhotosListPanelUtil"
    },
```

The “seacrhResults”is assigned to the dataProvider which is used in the view

```
"dataProviders": {
    "mars": {
        "action": "search",
        "response": "{{data.searchResults}}",
        "totalFound": 50
    }
},
```

```
<aw-scrollpanel>
    <aw-list dataprovider="data.dataProviders.mars">
        <aw-default-cell vmo="item" hideoverlay="true"></aw-default-cell>
    </aw-list>
</aw-scrollpanel>
```

## More example on Data Parser Definition

In following example, `typeIconUrl` is provided directly:

```
"dataParseDefinitions": {
    "nasa": {
        "identifier": "{{id}}",
        "typeIconUrl": "{{img_src}}",
        "cellHeader1": "{{rover.name}}",
        "cellHeader2": "{{earth_date}}",
        "props": {
            "cameraName": {
                "name": "cameraName",
                "displayName": "Camera Name",
                "value": "{{camera.full_name}}",
                "displayValue": "{{camera.full_name}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "type": "STRING"
            },
            "earthDate": {
                "name": "earthDate",
                "displayName": "Earth Date",
                "value": "{{earth_date}}",
                "displayValue": "{{earth_date}}",
                "type": "DATE"
            }
        }
    }
},
```

In the following example, `typeIconUrl` will be derived from modelType and typeHierarchy.

```
"dataParseDefinitions": {
    "nasa": {
        "identifier": "{{id}}",
        "modelType": "{{response.modelType}}",
        "typeHierarchy":["Item_Revision"],
        "cellHeader1": "{{rover.name}}",
        "cellHeader2": "{{earth_date}}",
        "props": {
            "cameraName": {
                "name": "cameraName",
                "displayName": "Camera Name",
                "value": "{{camera.full_name}}",
                "displayValue": "{{camera.full_name}}",
                "type": "STRING"
            },
            "earthDate": {
                "name": "earthDate",
                "displayName": "Earth Date",
                "value": "{{earth_date}}",
                "displayValue": "{{earth_date}}",
                "type": "DATE"
            }
        }
    }
},
```

## Events

Events can be fired and listened/handled in declarative **ViewModels**. Events are one means of a ViewModel communicating with other ViewModels or **UI elements**.

### Firing Events

- A viewModel can trigger events and it can subscribe to the events.

- **Actions** can fire events to trigger another action or any message to be shown.
- Events can be raised with eventData OR a message and a condition statement.

### Firing events in an action

Following example shows declarative way of raising event in action by using action type Event.

```
"sampleAction": {
    "actionType": "Event",
    "method": "Event",
    "inputData": {
        "events": [ {
            "name": "sampleActionComplete"
        } ]
    }
}
```

Where `name` is the name of the event (required), and it is referenced by interested listeners.

### Firing events with event data

`eventData` is an object for data to provide on the event. The structure of the eventData is specific to a given event type.

Example:

```
"events": {
    "success":  [ {
        "name": "complete",
        "eventData": {
            "source": "toolAndInfoPanel"
        }
    } ]
}
```

Where:

- `name` is the name of the event (required), and it is referenced by interested listeners.
- `eventData` is a JSON object to provide additional data with the event.

### Firing events when a condition is satisfied

Events can be fired when a condition is satisfied. The events with condition evaluated as true will be raised.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Example:

```
"sampleAction":{
    "events": {
       "success":  [ {
            "name": "complete",
            "eventData": {
                "source": "toolAndInfoPanel"
            },
            "condition": "conditions.isUpdated"

        },
        {
            "name": "complete",
            "eventData": {
                "source": "toolAndInfoPanel"
            },
            "condition": "conditions.isRemoved"
        }]
    }
}
"conditions": {
    "isUpdated": {
        "expression": "data.isItemUpdated"
    },
    "isRemoved": {
        "expression": "data.isItemRemoved"
    }

}
```

Where:

- `name` is the name of the event (required), and it is referenced by interested listeners.
- `eventData` - event data is a JSON object for data to provide on the event.
- `condition` is a **condition** that needs to be satisfied for the event to be fired.

### Firing events specifying the `excludeLocalDataCtx` flag

`excludeLocalDataCtx` is an optional flag that can be used when local data context not to be stored in the event data.

```
"actions": {
    "stopProgress": {
        "actionType": "Event",
        "method": "Event",
        "inputData": {
            "events": [
                {
```

```
                    "name": "progress.complete",
                    "excludeLocalDataCtx": true
                }
            ]
        }
    }
}
```

Where:

- `name` is the name of the event (required), and it is referenced by interested listeners.
- `excludeLocalDataCtx` is boolean value that excludes the local context from event data.

**Tip**

If any kind of two view models communicating with each other using events, they might need to use this excludeLocalDataCtx.

## Listening to events

The ViewModel can register to react to events by declaring an `onEvent` section. It can then conditionally execute a declarative action.

## Subscribing to an event

Consider the following example:

```
"onEvent": [ {
    "eventId": "someEvent",
    "action": "doSomething",

} ]
```

Where:

- `eventId` is the ID of an event that the ViewModel is interested to subscribe to.
- `action` is an action that is defined in the `actions` section of the ViewModel.

## Subscribing to an event on condition

Consider the following example:

```
"onEvent": [ {
    "eventId": "someEvent",
```


Unpublished work. © 2025 Siemens

```
    "condition": "conditions.shouldIDoSomething",
    "action": "doSomething",
    "cacheEventData": true
} ]
```

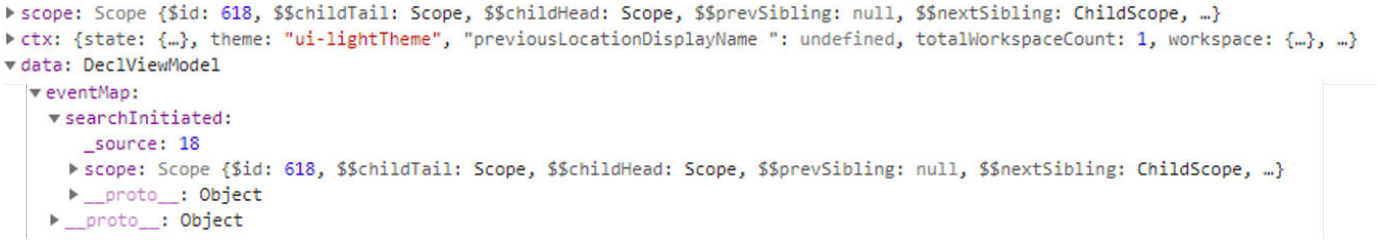
- `eventId` is the ID of an event that the ViewModel is interested to subscribe to.
- `condition` is a condition that is defined in the conditions section of the ViewModel. This condition can have an expression based on current ViewModel data, application context and/or eventData.
- `action` is an action that is defined in the `actions` section of the ViewModel. The action will be executed only if the specified condition is satisfied.
- `cacheEventData` - If set to `true`, the event data is cached inside declViewModel. The value of eventData will be stored as `declViewModel.eventMap[\<eventName\>]` as shown in the following section.

## Caching event data

The main purpose of the `cacheEventData` is to store the context eventData on declViewModel's eventData which can be used as input in triggered action. By default `cacheEventData` is false.

Example:

```
"onEvent": [ {
    "eventId": "searchInitiated",
    "action": "searchIcon",
    "condition": "conditions.invalidWord",
    "cacheEventData": true
} ],
"conditions": {
    "invalidWord": {
       "expression":  {
            "$source": "data.wordName.dbValue.length",
            "$query": {
                "$gt": 50
            }
        }
    }
}
```



For Example, let us take scenario where we have response from RESTService and we want to reuse this data from response in action which will trigger by event. So if we specify the `cacheEventData` to `true` in event object then data will be stored in eventData and can be reused.

Consider the following ViewModel fragment:

```
"actions": {
    "loadData": {
        "actionType": "RESTService",
        "method": "GET",
        "inputData": {
            "request": {
                "method": "GET",
                "url": "/serverUrl"
            }
        },
        "events": {
            "success": [ {
                "name": "dataLoaded",
                "eventData": {
                    "loadedObjects": "{{data.loadedObjects}}"
                }
            } ]
        }
    },
    "updateLoadedObjects":{
        "actionType": "JSFunction",
        "method": "updatePartialCtx",
        "inputData": {
            "name": "contextName",
            "value": "{{data.eventData.loadedObjects}}"
        },
        "deps": "js/appCtxService"
    }
}
 "onEvent":[
    {
        "eventId": "dataLoaded",
        "action": "updateLoadedObjects",
        "cacheEventData": true
    }
]
```

In this case, cached event data can be accessed in any an via `data.eventData`.

**Note**

Object conditions and expressions are supported on action messages and events from **Siemens Web Framework v4.2.0** onwards.


Siemens Web Framework
Unpublished work. © 2025 Siemens

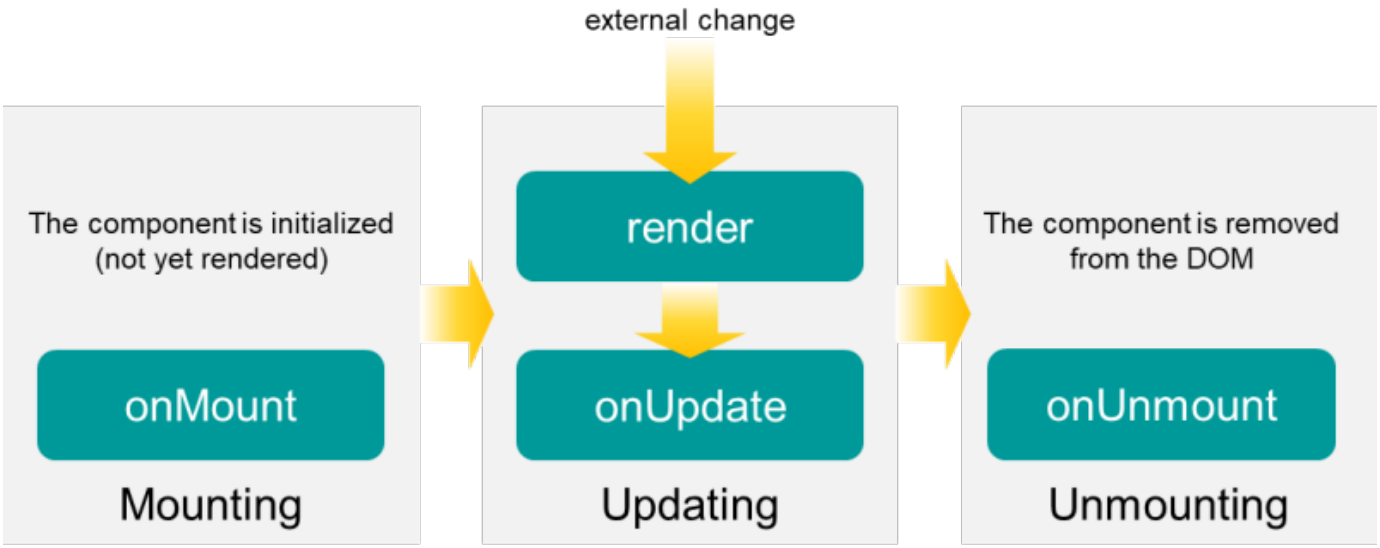
# Lifecycle hooks

**View models** (and therefore **components**) have an internal lifecycle that is internally managed by the declarative engine. Overall, the lifecycle of a view model can be divided into three main stages:

- A *Mounting* phase corresponding to the initialization of a view model, when it is first loaded but its view has not been rendered yet.
- An *Updating* phase occurring when the component internal state is updated and then its view is rendered.
- An *Unmounting* phase occurring when the view model is being destroyed and its corresponding view is removed from the DOM.

The view model lifecycle hooks are the declarative configuration points that enable you to get visibility into the lifecycle stages of a view model and execute declarative **actions** when such stages occur.

The following diagram lists all the lifecycle hooks and their corresponding stages:




## Defining lifecycle hooks

Lifecycle hooks can be defined in the `lifecycleHooks` top-level section of the view model, like this:

```
{
    "lifecycleHooks": {
        "onMount": "init",
        "onUpdate": [ {
            "action": "handleSelectionChange",
            "observers": [ "viewModel.data.selectionData" ]
        } ],
        "render": "render",
        "onUnmount": "cleanup"
```

```
    }
}
```

Apart from the `onUpdate` hook that can be configured to execute multiple actions (each with its own set of *observers*) all other hooks can be configured to execute only one action defined in the same view model. In the previous example:

- the `init` action is executed when the view model is mounted
- the `render` action is executed when the view model is rendered
- the `handleSelectionChange` action is executed after the view is re-rendered, whenever **selection data** changes
- the `cleanup` action is executed when the view model is destroyed

**Tip**

The configuration of lifecycle hooks in a view model is entirely optional, but it may be needed to implement specific use cases, such as providing a codeful alternative to a declarative view using the `render` hook.

## Available lifecycle hooks

The following sections describe all the available lifecycle hooks more in detail.

### onMount

This hook indicates the lifecycle stage of the mounting of the view model in the DOM. Once the view model is constructed/initialized but its view is not yet rendered, the action specified for this hook will be invoked.

This hook can be leveraged to perform any application-specific registrations of global resources, event subscriptions, initializations, reveal operations, and so on.

**Note**

Older versions of Siemens Web Framework used to expose an `onInit` hook. As of version 6.2, you should use the `onMount` hook instead.

### onUpdate

**Note**

This lifecycle hook is available from Siemens Web Framework 5 onwards.


Siemens Web Framework
Unpublished work. © 2025 Siemens

This hook lets users configure the actions that need to be performed asynchronously after every render if the values specified in the `observers` dependency array change. Multiple `onUpdate` hooks can be added via an array of objects or defined directly as an object. Each `onUpdate` hook can support multiple observers. Observers must start with `viewModel` (for component state), `ctx` (for application state), or `props` for any other data received from parent components via props.

Consider the following example:

```
{
    "lifecycleHooks": {
        "onUpdate": [
            {
                "action": "handleSelectionChange",
                "observers": [ "viewModel.data.selectionData" ]
            },
            {
                "action": "processText",
                "observers": [ "props.text" ]
            }
        ],
    }
}
```

In this case:

- The `handleSelectionChange` action will be executed only if `data.selectionData` changes.
- The `processText` action will be executed only if `props.text` changes.

### render

**Note**

This lifecycle hook is available from Siemens Web Framework 5 onwards.

This hook lets users configure the view contents (in JavaScript and JSX format) for the corresponding view model, as an alternative to providing a declarative view. The action receives the input parameters, including view model (component state), context (global state), and any data received from parent components via props.

For example, consider the following simple view model defining a very simple custom cell:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "render": {
            "method": "awSimpleCellRenderFunction",
            "deps": "js/AwSimpleCellService"
        }
```


Unpublished work. © 2025 Siemens

```
    },
    "data": {},
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    },
    "props": {
        "title": {
            "type": "object"
        },
        "image": {
            "type": "object"
        }
    }
}
```

In this case, a `render` lifecycle hook is specified, so the `render` action will be executed to provide a view for the component.

A simple implementation of the `awSimpleCellRenderFunction` method could be similar to the following:

```
export const awSimpleCellRenderFunction = ( props ) => {
    const { title, image } = props;

    if( !title ) { return; }

    return (
        <div className = 'aw-default-cell sw-row'>
            <div className='sw-cell-image'>
                <img src={props.image}></img>
            </div>
            <h2>{title}</h2>
        </div>
    );
};
```

**Note**

Usage of render functions instead of declarative view template should be restricted only to complex components in which a more codeful approach is more appropriate and easier to manage. Whenever possible, use declarative view templates.

### onUnmount

This hook indicates the lifecycle stage of the destruction of the view model. Once the view model is marked for destruction/teardown, the action name specified in this hook will be invoked. This hook can be leveraged to perform any application-specific de-registrations of global resources, event un-subscriptions, data cleanup, etc. Typically the actions specified in this hook will be simple synchronous operations. Any


Siemens Web Framework
Unpublished work. © 2025 Siemens

data mutation, subsequent use of output data, or using batch actions may cause undefined behavior and hence is not recommended.

## Configuring lifecycle hooks as alternatives to Deprecated Events

You can use some of the Lifecycle hooks as an alternative to the following deprecated events:

- `*ViewModelName*.contentLoaded`
- `*ViewModelName*.contentUnloaded`

The audit check has been reporting a deprecated warning for using these events from Siemens Web Framework v4.1 onwards and suggested replacing the usages with lifecycle hooks:

```
thinclient\favoritesjs\src\viewmodel\favoritesListViewModel.json
  favoritesList.contentLoaded :
@deprecated SWF4.1, The <viewName>.contentLoaded event is deprecated.
@alternative ViewModel Construct - lifecycleHooks:onMount. Refer to “ViewModel Lifecycle Hooks” API documentation for details.
@obsoleteIn SWF5.0
thinclient\favoritesjs\src\viewmodel\nativePageViewModel.json
  nativePage.contentUnloaded :
@deprecated SWF4.1, The <viewName>.contentUnloaded event is deprecated.
@alternative ViewModel Construct - lifecycleHooks:onUnmount. Refer to “ViewModel Lifecycle Hooks” API documentation for details.
@obsoleteIn SWF5.0
```

Here are the steps to migrate from these deprecated events and use the corresponding lifecycle hooks instead:

1. In the view model where *ViewModelName*.contentLoaded or *ViewModelName*.contentUnloaded are defined, add a `lifecycleHooks` section:

```
{
    "lifecycleHooks": {
        "onMount": "",
        "onUnmount": ""
    }
}
```

2. Find existing subscription for *ViewModelName*.contentLoaded in the `onEvent` section of the view model.

```
"onEvent": [
     {
         "eventId": "nativePage.contentLoaded",
         "action": "initializePage"
     }
]
```

3. Copy the action name specified to handle *ViewModelName*.contentLoaded and specify it as the action to execute for the `onMount` hook.

```
"lifecycleHooks": {
    "onMount": "initializePage"
}
```

4. Delete the event subscription for *ViewModelName*.contentLoaded.
5. Follow the same steps for *ViewModelName*.contentUnloaded event. And replace it with the `onUnmount`.

Eventually, the deprecated events shown below:

```
"onEvent": [
    {
        "eventId": "nativePage.contentLoaded",
        "action": "initializePage"
    },
    {
        "eventId": "nativePage.contentUnloaded",
        "action": "cleanupPage"
    }
]
```

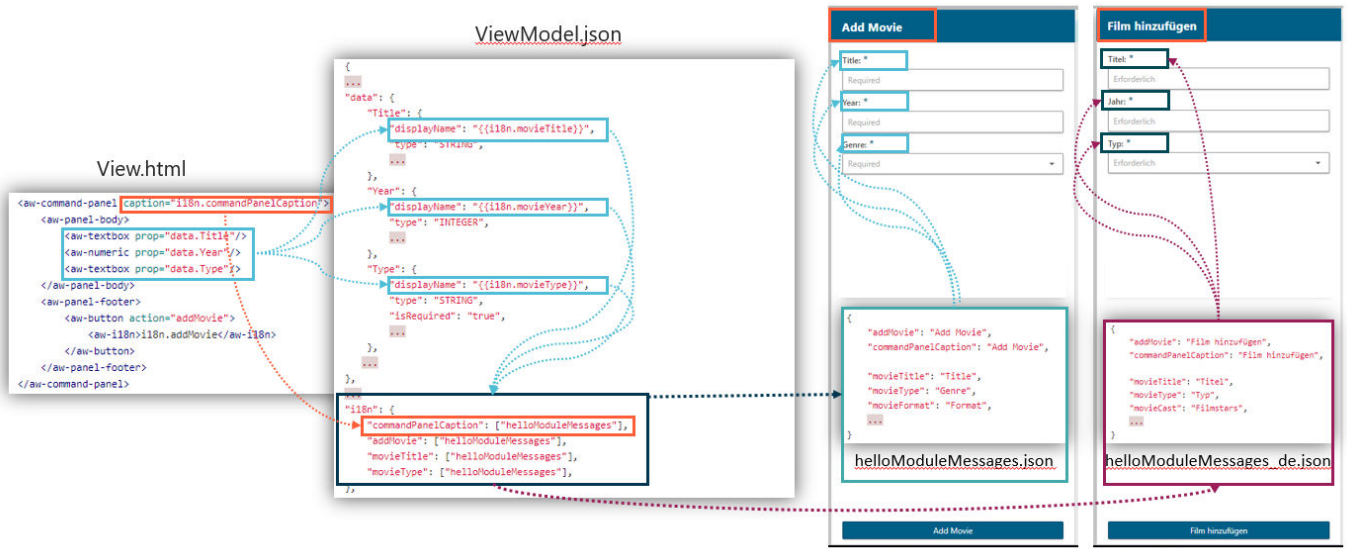
...should be converted to lifecycle hooks:

```
    "lifecycleHooks": {
        "onMount": "initializePage",
        "onUnmount": "cleanupPage"
    }
```

## I18n and I10n

Siemens Web Framework provides a way to externalize your application UI using locale specific JSON files containing key/value pairs, where key is the string to be localized, and value is the localized text to be displayed. The **View** refers to localizations defined in the `i18n` section of the **ViewModel**. i18n text can be provided as a JSON bundle or provided inline.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Adding translated resources

The names of the JSON files in i18n directories must satisfy the following naming convention: `<identifier>Messages_<locale>.json`

Where:

- `<identifier>` is a unique identifier for the file. Preferably prefix it by module name e.g. `helloModuleMessages`.
- `<locale>` is a locale identifier composed by a two-letter lowercase language code and a two-letter uppercase country code (e.g. `ja_JP`) except for the following language codes: `it`, `de`, `fr`, `es`.

**Note**

No locale suffix must be specified for English localization files.

Examples:

- helloModuleMessages.json (English)
- helloModuleMessages_it.json (Italian)
- helloModuleMessages_de.json (German)
- helloModuleMessages_zh_CN.json (Chinese)

## Order of resolving locale information

The framework takes the below path for picking the locale to be used in the application:

- First, it looks for a locale in the query string of the application URL and if found, it extracts the value from it. An example query string looks like `?locale=en_US`
- If the locale is not present in the URL, it checks the `localStorage` for the last used locale in your application. The localStorage is domain specific and shared across every window or tab running at the same origin.
- If the locale is not present in localStorage, it would simply use the `browser's current locale/language` setting.
- If the browser locale doesn't match one of the installed locales, it picks the initial locale (which defaults to en_US) from the installed locales.

**Note**

If any message key is missing for selected locale then we use English value of that message key as localized text to display.

## Adding a New Locale in Customization

To add support for a new locale, say "Dutch" or "du", a corresponding `<identifier>Messages_du.json` file needs to be introduced as a translated resource for all localization files.

An additional custom date format settings can also be configured via `localeDateTime.json`. For more information, see **Custom locale configuration**.

## Customization of supported Locale list

If your application solution supports only a subset of SWF supported locales, you need to add that customized locale list as a value of property name "locale" in site_\<mysolution\>.json file of your solution.

For example :

```
{
    "name": "showcase",
    "solutionName": "showcase",
    "kits": [
        "showcase",
        "active-workspace-showcase"
    ],
    "locale": [ "en_US", "cs_CZ", "de", "es" ]
}
```

An example of Site can be found **here**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

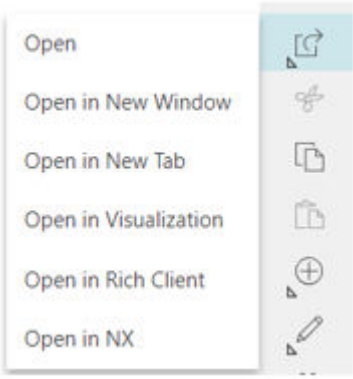
## Parameterized Localization

Consumer of declarative UI can declaratively define and consume parameterized localized user facing text in their ViewModel artifacts. It supports any property (for e.g. command labels, command panel title, command panel section title) which takes i18n values

```
The parametric value is substituted once before element is rendered in the
UI, it will not force any UI refresh if parameter value gets updated later.
```

## Command Labels

### Without parameters



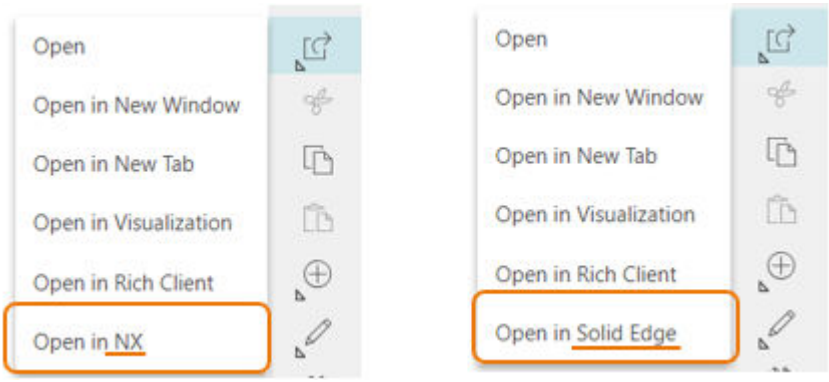

An i18n string inside Messages.json

```
"Ldf0OpenRemoteObjectCommandTitle":"Open in NX"
```

Command definition inside commandsViewModel

```
"Ldf0OpenRemoteObject": {
            "iconId": "cmdOpen",
            "isGroup": false,
            "title": "{{i18n.Ldf0OpenRemoteObjectCommandTitle}}"
        }
```

## Single parameter




An i18n string inside Messages.json

```
"Ldf0OpenRemoteObjectCommandTitle":"Open in {0}"
```

Command definition inside commandsViewModel

```
"Ldf0OpenRemoteObject": {
            "iconId": "cmdOpen",
            "isGroup": false,
            "title": {
                "text": "{{i18n.Ldf0OpenRemoteObjectCommandTitle}}",
                "params": [
                    "{{ctx.selected.props.ldf0RemoteSystem.uiValues[0]}}"
                ]
            }
        }
```

## Multiple parameters

An i18n string inside Messages.json

```
"hidePartialIncomingRelations": "Hide {0} of {1} Incoming Relations",
```

Command definition inside commandsViewModel

```
"Rv1OnNodePartialExpandInPartial": {
            "iconId": "cmdIncomingPartial",
            "isGroup": false,
            "title": {
                "text": "{{i18n.hidePartialIncomingRelations}}",
                "params": [
```


Unpublished work. © 2025 Siemens

```
"{{ctx.graph.commandContextItem.boundObject.visible_in_degree}}",

"{{ctx.graph.commandContextItem.boundObject.filtered_in_degree}}"
                ]
            }
        }
```

## Properties

View.html

```
<aw-label prop="data.parameterizedPlanetProp"></aw-label>
```

Messages.json

```
 "parameterizedPlanetProp": "Planet - {0}"
```

ViewModel.json

```
 "parameterizedProperty": {
           "displayName": {
               "text": "{{i18n.parameterizedPlanetProp}}",
               "params": [ "{{ctx.galleryContextObject}}" ]
           },
           "type": "STRING",
           "isRequired": "false",
           "isEditable": "false",
           "dbValue": "",
           "dispValue": ""
       }
```

## Command panel title/ Command panel section title

View.html

```
<aw-command-panel caption="data.panelTitleWithParam.title"
anchor="data.universalObjectsCommandbarAnchor">
```

Messages.json

```
 "currentPlanet": "Current Planet - {0}"
```

ViewModel.json

```
"panelTitleWithParam": {
          "title": {
              "text": "{{i18n.currentPlanet}}",
              "params": [ "{{ctx.galleryContextObject}}" ]
          }
      }
```

## Messages

Messages are declaratively defined in the **View Model**.

### Configurations based messages

#### Messages

- are launched by events and actions
- cover information, warning, error notifications
- can give the user options
- are localized
- can result in firing actions
- can leverage view model data and conditions

#### Messages configuration includes

- *messageType* - INFO, WARNING, ERROR
- *messageText* - user facing text for the message
- *messageTextParams* - parameters for the user facing message
- *navigationOptions* - array of objects for each option to give the user each with an option identifier, user facing text and action to fire when elected.

In the below example, `"confirmOverwrite"` is warning message and has options for the user to choose between "overwrite" or "cancel" options while `"pinToHome"` is just an information message where no user action is required.

```
"messages": {
    "confirmOverwrite": {
        "messageType": "WARNING",
        "messageText": "{{i18n.nameInUse}}",
        "messageTextParams": [
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "{{data.searchName.dbValue}}"
        ],
        "navigationOptions": [ {
            "option": "Cancel",
            "text": "{{i18n.CancelText}}",
            "action": ""
        },
        {
            "option": "Overwrite",
            "text": "{{i18n.OverwriteText}}",
            "action": "overwrite"
        } ]
    },
    "pinToHome": {
        "messageType": "INFO",
        "messageText": "{{i18n.pinToHome}}",
        "messageTextParams": [
            "{{data.searchName.dbValue}}"
        ]
    }
}
```

## APIs for Notification Messages

Apart from above view model configurations, Messaging service also expose API to configure the notification messages.

- *setMessageVisibility (messageType, value)* This API is used to customize the notification message visibility for Info, Warning and Error messages globally on the application startup. If any of the message types is set to false, the notification message will not be visible for that message type during entire session.
- *setTimeout (messageType, timeout)* This API is used to set default timeout for INFO/ERROR notification messages for entire session.

In the below example, Timeout for INFO and ERROR is set to 10 seconds, so both INFO and ERROR will be shown for 10 seconds throughout the application. Visibility for the WARNING message type is set to false, which means the WARNING messageType will remain hidden throughout the application. Warning message box will be modal. If conumser want any message box to be modal, just add the navigation options to the message definition.

```
    _messagingSvc.setTimeout( 'INFO', 10 );
    _messagingSvc.setTimeout( 'ERROR', 10 );

    _messagingSvc.setMessageVisibility( 'INFO', ture );
    _messagingSvc.setMessageVisibility( 'WARNING', false );
    _messagingSvc.setMessageVisibility( 'ERROR', true );
```

### Mapping the server error directly to client

Siemens Web Framework 4.2 or above, there is a change in the Notification messaging paradigm. See **Upgrading Action Message Configurations** for more details.

In the below example, consumer need to use `errorCode` interpolation to map the error coming from server to client.

```
"<action-name>": {
    "...":"...",
    "actionMessages": {
        "failure": [
            {
                "message": "myFailureMessage"
            }
        ]
    }
},
"messages":{
    "myFailureMessage": {
        "messageType": "ERROR",
        "messageText": "{{errorCode}}"
    }
}
```

## Conditions

*Conditions* provide live logic evaluation. They can be used to perform certain business logic or actions depending on whether the condition evaluates to true or false.

Conditions can be used to serve following purposes:

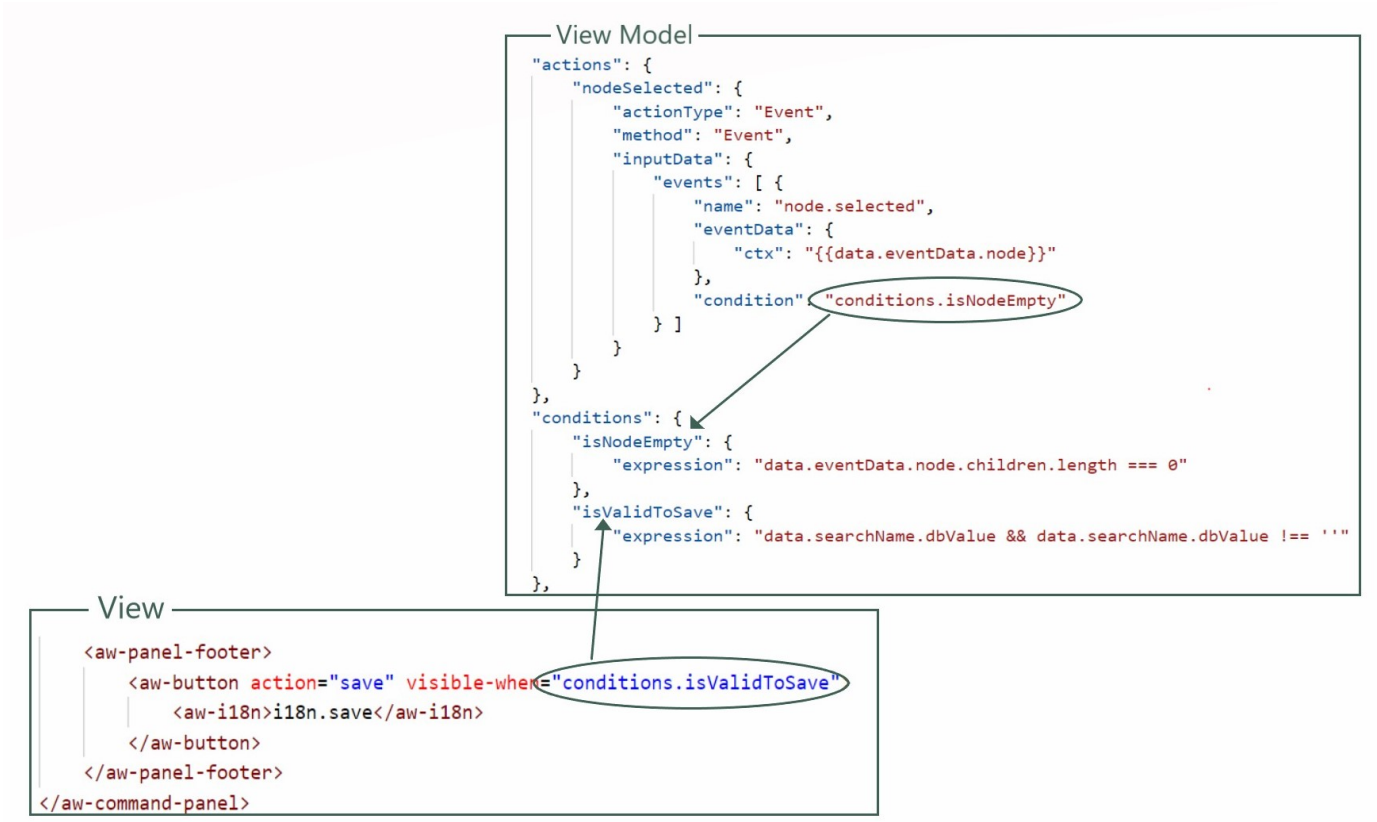
- can be used in *visible-when* clauses
- can be used for event handling (*onEvent* handler)
- data value can be live evaluated by condition
- conditions can refer to **ViewModel** state
- conditions can update ViewModel state
- can leverage Boolean condition expressions
    - &&, ||, ==, !=


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Conditions Usage

Conditions are defined in the viewmodel. They can be consumed in the view or viewmodel using the following syntax:

```
conditions.<expression_name>
```




## Writing conditions using data, props and ctx

We can also write condition based on values in Objects of Data section ,CTX object and also using props

```
"condition": "data.showActionMsg.dbValue"
"condition": "ctx.showMessage === 'true'"
"condition": "props.state==='UK'"
```

## Condition Expression

The condition expression can be defined either as a simple string or in an object format.

Expression as a simple string:

```
{
    "conditions":{
        "expression_name": {
            "expression": "ctx.selected.length > 1"
        }
    }
}
```

Expression as an object:

```
{
    "conditions":{
        "expression_name": {
            "$source": "ctx.selected",
            "$query": {
                "$gt": 1
            }
        }
    }
}
```

## Supported Operators

Following is the list of operators that are supported when the condition expression is defined as an object

1. `$source` : indicates the reference on data context node to be used as starting point for evaluation
2. `$query` : defines the query to be executed on the \$source
3. `$eventSource` : indicates that the specified viewModel is the source of the event
4. `$all` : applicable when \$source is an array. Indicates that query result should be valid on all instances of the array elements.
5. `$and` : indicates logical 'AND' of each query result
6. `$or` : indicates logical 'OR' of each query result
7. `$not` : indicates logical 'NOT' of each query result
8. `$adapt` : indicates that the resulting \$source should be adapted before evaluating the query
9. `$in` : indicates that query should match with at least one of the value from the array
10. `$notin` : indicates that query should not match with any of the value from the array
11. `$eq` : equality operator
12. `$ne` : not equals operator
13. `$gt` : greater than operator


Siemens Web Framework
Unpublished work. © 2025 Siemens

14. `$gte` : greater than or equal to operator
15. `$lt` : less than operator
16. `$lte` : less than or equal to operator
17. `$regexp` : regular expression
18. `$typeOf` : verifies if the viewmodel object belongs to a particular type hierarchy

### Expresssion definition examples using the operators

1. **Use of \$and**: Enable a command handler when 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"
                    }
                }
            ]
        }
    }
}
```

1. **Use of \$or**: Enable a command handler when 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"
                    }
                }
            ]
        }
    }
}
```

1. **Use of \$and and \$or**: Enable a command handler when selected object type is ItemRevision OR ( object_name is Newstuff AND object type is Folder )

```
{
    "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":
```


Unpublished work. © 2025 Siemens

```
                            {
                                "$eq": "Newstuff"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

1. **Use of \$adapt**: Enable a command handler when adapted - selected object type is a Cpd0DesignElement

```
{
    "expression":
    {
        "$source":
        {
            "$adapt": "ctx.selected"
        },
        "$query":
        {
            "$source": "modelType.typeHierarchyArray",
            "$query": {
                "$in": ["Cpd0DesignElement"]
            }
        }

    }
}
```

1. **Use of \$adapt along with \$all**: Enable a command handler when all of the adapted - multi selected object types are Cpd0DesignElement

```
{
    "expression":
    {
        "$source":
        {
            "$adapt": "ctx.mselected"
        },
        "$query":
        {
            "$all":
            {
                "$source": "modelType.typeHierarchyArray",
                "$query": {
                    "$in": ["Cpd0DesignElement"]
                }
            }
        }
```

```
    }
}
```

1. **Use of \$lte**: Enable a command handler when total workspace count is less than or equal to 1

```
{
    "expression":
    {
        "$source": "ctx",
        "$query":
        {
            "$source": "totalWorkspaceCount",
            "$query": {
                "$lte": 1
            }
        }
    }
}
```

1. **Reuse base condition and extend**: Enable a command handler when base condition is true and type of adapted selected object is Cpd0DesignElement. **Note**: 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 evaluation context. Please refer to the '*More on reuse of conditions*' section at the bottom for details.

```
{
    "expression": {
        "$and": [
            {
                "$source": "conditions.wiki_base",
                "$query": {
                    "$eq": true
                }
            },
            {
                "$source": {
                    "$adapt": "ctx.mselected"
                },
                "$query": {
                    "$all": {
                        "$source": "modelType.typeHierarchyArray",
                        "$query": {
                            "$in": ["Cpd0DesignElement"]
                        }
                    }
                }
            }
        ]
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. **Dynamically resolve values on RHS of operator**: Enable a command handler by evaluating condition which has both LHS and RHS that need to be resolved from the data context

```
{
    "expression": {
        "$source": "ctx.selected.type",
        "$query": {
            "$in":
"{{ctx.locationContext.modelObject.modelType.typeHierarchyArray}}"
        }
    }
}
```

1. **Use of \$typeOf**: Enable a command handler when the selected object belongs to either FolderObject or Email in its type hierarchy

```
{
    "expression": {
        "$and": [
            {
            "$source": "ctx.selected",
            "$query": {
                "$typeOf": "FolderObject"
                }
            },
            {
            "$source": "ctx.selected",
            "$query": {
                "$typeOf": "Email"
                }
            }
        ]
    }
}
```

1. **Use to show not empty/null array**: Enable a command handler when the selected object is not null or not empty array

```
{
    "expression": {
        "$and": [
            {
            "$source": "ctx.selected",
            "$query": {
                "$ne": []
                }
            },
            {
            "$source": "ctx.selected",
            "$query": {
                "$notNull": true
```

```
                }
            }
        ]
    }
}
```

1. **Use of \$eventSource**: Perform clear operation only when clearEvent is raised in the same viewModel

```
{
    "onEvent": [{
        "eventId": "clearForm",
        "action": "clearFormEventHandler",
        "condition": "conditions.isClearClickedOnCurrentViewModel"
    }],
    "conditions": {
        "isClearClickedOnCurrentViewModel": {
            "expression": {
                "$eventSource": "self"
            }
        }
    }
}
```

1. **Use of \$not and \$eventSource**: Perform clear operation when clearEvent is not raised in the same viewModel

```
{
    "onEvent": [{
        "eventId": "activateForm",
        "action": "activateFormdEventHandler",
        "condition": "conditions.isActivateClickedOnDifferentViewModel"
    }],
    "conditions": {
        "isActivateClickedOnDifferentViewModel": {
            "expression": {
                "$not": {
                    "$eventSource": "self"
                }
            }
        }
    }
}
```

## Conclusions

- String based condition expression definition can reuse other string based condition expressions
- String based condition expression definition cannot reuse object based condition expressions


Siemens Web Framework
Unpublished work. © 2025 Siemens

- Object base condition expression can reuse string based and/or object based condition expression only when the reused expression is associated with another handler.
- Splitting expressions to achieve modularity and reuse is not supported in object base condition expression definition when the reused expression is not associated with any handler

## Context

**Application Context** can be used in your View render function or HTML template. But you must register interest in any particular path using the appCtx in the “ctx” section of the ViewModel before using AppCtx, in your *View.html, *ViewModel.json, or any dependent JS.

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {},
    "data": {},
    "conditions": {
        "logoEnabled":{
            "expression": "ctx.showLogo"
        }
    },
    "onEvent": [],
    "lifecycleHooks": {},
    "ctx":{
        "showLogo": null
    }
}
```

This will ensure your component viewModel remains in sync with any changes to the appCtx for that key path. The ctx section establishes a direct link between the specified Key (path) in the AppCtx and the View/ViewModel.

A ctx object will get passed into your View's render function with key and values that you registered for only. **The entire AppCtx object will not get passed in -- only the subset you requested.**

View is re-rendered whenever the registered key value in AppCtx is modified via a published API. Be careful what you register for because if you register for too many things you will get re-rendered too often causing performance. Register for too few then your ctx object will not contain enough data. These re-renders are triggered in your component only when you are modifying AppCtx through the AppCtxService published APIs.

The preferred flow w.r.t. AppCtx is the following:-

- Register interest in a given path inside your ViewModel ctx section
- Use the ctx that gets passed into your component's props. Pass that ctx object to any necessary JS functions

- If modifying AppCtx values only use published api from AppCtxService

**Note**

Currently the declarative transpilers will attempt to add ctx.keys automatically to your viewModel. However, it is best to not rely on the transpiler to auto-detect which ctx keys to add to the viewModel and do it manually.

## Derived state

The derived state is an internal component state computed based on the current `props` or information in `ctx`. The state can be configured in the view model through the `ctx` property.

```
{
  "ctx": {
    "valClass": {
      "parser": {
        "method": "getValClassMemo",
        "deps": "js/componentUtils"
      }
    }
  }
}
```

This configuration allows you to ensure that your component is updated whenever the referenced values in `ctx` change while also ensuring that your rendering is optimized to improve performance.

### When to use derived state

Derived state or computed state can be used in either or both of the following two cases.

1. The component is doing some very expensive calculations based on some object/literal passed by props to the component. The computation is costly and needs to be done in every render cycle. A derived state could be used in these cases to avoid the expensive computation in every render.
2. The component has a dependency on some object/literal based on ctx. When the value changes in ctx, we want our component to be re-rendered automatically, and the lasted ctx value should be available in props ( via ctxMin) to the said component.

### Introduction

In a previous iteration of `AwCheckbox` it had this view model:

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "render": {
            "method": "awCheckboxRenderFunction",
            "deps": "js/AwCheckboxService"
        }
    },
    "data": {},
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    }
}
```

And this render function:

```
import AwPropertyLabel from 'viewmodel/AwPropertyLabelViewModel';
import AwCheckboxVal from 'viewmodel/AwCheckboxValViewModel';
import { getValClass } from 'js/componentUtils';

/**
 * render function for AwCheckbox
 * @param {*} param0 context for render function interpolation
 * @returns {JSX.Element} react component
 */
export const awCheckboxRenderFunction = ( { prop } ) => {
    const valClass = getValClass( prop );
    return (
        <AwPropertyLabel {...prop}>
            <AwCheckboxVal className={valClass} {...prop}></AwCheckboxVal>
        </AwPropertyLabel>
    );
};
```

In this example, the valClass is the derived state. This variable is recomputed every render and then used in the view. This approach has no functional issues, and valClass will be set correctly. The issue is that **`AwCheckboxService` will recompute valClass every time the component is rendered**, even if nothing that impacts it has changed. For something simple to calculate, the performance impact is minimal. However, for something expensive to calculate (such as command state), this can cause significant performance issues.

The new derived state configuration can be used to resolve the issues. The first step is to add the `ctx` configuration to the view model:

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {
        "render": {
            "method": "awCheckboxRenderFunction",
            "deps": "js/AwCheckboxService"
```

```
        }
    },
    "ctx": {
        "valClass": {
            "parser": {
                "method": "getValClassMemo",
                "deps": "js/componentUtils"
            }
        }
    },
    "data": {},
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    }
}
```

The `valClass` entry in `ctx` tells the framework that the `valClass` property in the `ctx` (or `ctxMin` temporarily) provided to the render function should be created by the referenced parser. If we look at the implementation for that parser:

```
/**
 * A version of getValClass that uses derived state to improve performance
 *
 * @param {Object} vmDef View model
 * @param {Object} prop Current properties
 * @returns {[DerivedStateResult]} Derived state configurations
 */
export const getValClassMemo = ( vmDef, prop ) => {
    return [ new DerivedStateResult( {
        ctxParameters: [],
        additionalParameters: [ prop.dirty ],
        compute: ( renderContext, isDirty ) => {
            let classes = [ 'sw-property-val' ];
            if( isDirty ) {
                classes.push( 'sw-changed' );
            }
            return classes.join( ' ' );
        }
    } ) ];
};
```

We can see that it is returning a single `DerivedStateResult`. This is a utility type used to help validate that this parser returns the required information. The properties of this type are:

- `ctxParameters` is a string list of anything in `ctx` that the component depends on. This example has no dependency on `ctx`, so it is empty.
- `additionalParameters` is a list of any other parameters required to compute this state. These will be primarily pulled from props. These properties should be as specific as possible and ideally be **primitive** types. The example above shows that only the `dirty` flag is provided, not the entire property.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- `compute` is the function that will be called to get the latest derived state. This function will be called with the same render context as the render function as its first parameter, along with any of the configured `additionalParameters`. This function can be easily extracted as a global function to simplify unit testing.

The combined result of all of the `compute` functions is then set in the `ctx` (or `ctxMin` temporarily), so the render function can access it:

```
import AwPropertyLabel from 'viewmodel/AwPropertyLabelViewModel';
import AwCheckboxVal from 'viewmodel/AwCheckboxValViewModel';

/**
 * render function for AwCheckbox
 * @param {*} param0 context for render function interpolation
 * @returns {JSX.Element} react component
 */
export const awCheckboxRenderFunction = ( { prop, ctxMin: { valClass } } )
=> {
    return (
        <AwPropertyLabel {...prop}>
            <AwCheckboxVal className={valClass[0]} {...prop}></
AwCheckboxVal>
        </AwPropertyLabel>
    );
};
```

The result of all of these changes is that the **AwCheckbox will only compute `valClass` when the `dirty` property changes** instead of every render while simplifying the logic in the render function.

### CTX parameters

One significant difference in Beyond AngularfJS is that a component that depends on data in `ctx` will not be automatically updated when that `ctx` changes. Instead, the framework must be told exactly what in `ctx` a component depends on to update. This can be done with the `ctx` property, with the property you depend on, and a `null` value:

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {
        "render": {
            "method": "awCheckboxRenderFunction",
            "deps": "js/AwCheckboxService"
        }
    },
    "ctx": {
        "selected": null,
        "pselected.props.a.dbValue": null
    },
    "data": {},
```


Unpublished work. © 2025 Siemens

```
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    }
}
```

These dependencies need to specify exactly what is being used. In this example, `selected` means the component will only update when the entire object in the `selected` ctx changes. When an individual property `selected` is updated, the component will not be updated. The `pselected.props.a.dbValue` means that the component will be updated when that specific property changes. Any properties listed in this `ctx` are processed and provided to the component as `ctx` (or `ctxMin` temporarily). **Only data specified in the view model will be included in ctx**, any data not listed will not be included.

In some cases you may not have a static list of ctx dependencies. For example with a pre Beyond AngularJS view such as:

```
<div>
  <span>{{`ctx ${subPanelContext.propToRender} is $
{ctx[subPanelContext.propToRender]}`}}</span>
</div>
```

We have no way of knowing which `ctx` dependencies this view model has outside of runtime. A `parser` function is required to support a use case such as this. In this example

```
`ctx ${subPanelContext.propToRender} is $
{ctx[subPanelContext.propToRender]}`
```

is our derived state. We can call this derived state `stringToRender`. So in our view model JSON we would have:

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {
        "render": {
            "method": "myRenderFn",
            "deps": "js/AwComponentService"
        }
    },
    "ctx": {
        "stringToRender": {
            "parser": {
                "method": "getValueFromCtx",
                "deps": "js/myService"
            }
        }
    },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "data": {},
    "conditions": {},
    "onEvent": [],
    "lifecycleHooks": {
        "render": "render"
    }
}
```

and our parser would look like:

```
const combineCtxData = ( { ctxMin }, ctxName ) => `ctx ${ctxName} is $
{ctxMin[ctxName]}`;
export const getValueFromCtx= ( vmDef, prop: { subPanelContext:
{ propToRender } ) => {
    return [ new DerivedStateResult( {
        ctxParameters: [ propToRender ],
        additionalParameters: [ propToRender ],
        compute: combineCtxData
    } ) ];
};
```

This parser tells the framework that we have a dependency on whatever in ctx `propToRender` is set to and that `propToRender` also needs to be included as an additional parameter to the compute function. Now whenever that ctx changes our component will be updated with the latest `stringToRender` value, leaving us with a very simple render function:

```
export const myRenderFn = ( { prop, ctxMin: { stringToRender } } ) => {
    return (
        <div>
            <span>{stringToRender[0]}</span>
        </div>
    );
};
```

**Limitations**

There are a few key limitations around derived state due to limitations around hooks. The general issue is that the `ctx` that is being watched and the list of derived state cannot change after a component is first rendered. In the above example:

```
const combineCtxData = ( { ctxMin }, ctxName ) => `ctx ${ctxName} is $
{ctxMin[ctxName]}`;
export const getValueFromCtx= ( vmDef, prop: { subPanelContext:
{ propToRender } ) => {
    return [ new DerivedStateResult( {
        ctxParameters: [ propToRender ],
        additionalParameters: [ propToRender ],
        compute: combineCtxData
```

```
    } ) ];
};
```

A change in `propToRender` would result in a change in `ctxParameters`, causing an error thrown by React. This means that `propToRender` cannot change after the component is initialized. The current workaround for this issue is to create a "wrapper" component that will remove and then fully reinitialize the component with the derived state.

If we had a parser function such as:

```
const showProp = ( renderContext, propVal ) => `Hello ${propVal}!`;
export const getValueFromCtx = ( vmDef, prop: { propertyList } ) => {
    return propertyList.map( propVal => new DerivedStateResult( {
        ctxParameters: [],
        additionalParameters: [ propVal ],
        compute: showProp
    } ) );
};
```

The same issue will show up if `propertyList` is ever updated.

### FAQ

#### What is the difference between ctx and ctxMin?

`ctxMin` contains only the parts of `ctx` that your component has identified as a dependency, while `ctx` contains everything in the data store. **If you are using something from `ctx` that is not in `ctxMin` you have a bug currently and will need to configure your view model to add it**. `ctxMin` is planned to replace `ctx` at some point in the future.

#### Why does the parser function return an array instead of a single value?

By breaking your computation into multiple parts you can optimize even further as each `DerivedStateResult` has an independent set of parameters, and the compute function will only be evaluated when those specific parameters change. For example with this parser:

```
const dsc1 = ( { ctx: { myFlagA } } ) => myFlagA === 'red';
const dsc2 = ( { ctx: { myFlagB } } ) => myFlagB === 'blue';
export const myParser = () => [ new DerivedStateResult( {
    ctxParameters: [ 'myFlagA' ],
    compute: dsc1
} ), new DerivedStateResult( {
    ctxParameters: [ 'myFlagB' ],
    compute: dsc2
} ) ]
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

The `dsc1` function will only run when `myFlagA` changes and `dsc2` will only run when `myFlagB` changes.

**When do I need to use ctx?**

Any time that you are using something from `ctx` to modify the UI a ctx entry is required. For example in:

```
import AwIcon from 'viewmodel/AwIconViewModel';

const isLogoEnabled = ( ctx ) => {
    if( ctx.aw_hosting_enabled && ctx.aw_hosting_config ) {
        return ctx.aw_hosting_config.ShowSiemensLogo;
    }
    return true;
};

export const AwLogoRenderFunction = ( { ctx } ) => {
    if( isLogoEnabled( ctx ) ) {
        return (
            <div className='sw-logoContainer'>
                <div className='sw-logoImage'>
                    <AwIcon iconId='miscSiemensLogo'></AwIcon>
                </div>
            </div>
        );
    }
};
```

the ctx being used is `aw_hosting_enabled`, `aw_hosting_config`, and `aw_hosting_config.ShowSiemensLogo`. Therefore the ctx configuration should be:

```
{
    "ctx": {
        "aw_hosting_enabled": null,
        "aw_hosting_config": null,
        "aw_hosting_config.ShowSiemensLogo": null
    }
}
```

Without this configuration, this component would not be updated when any referenced properties are changed, potentially resulting in defects.

The view compiler will attempt to generate the `ctx` entries for your view model if they are not configured. If the example above had used a view instead:

```
<div className="sw-logoContainer" exist-when="ctx.aw_hosting_enabled &&
ctx.aw_hosting_config.ShowSiemensLogo">
    <div class="sw-logoImage">
        <AwIcon iconId="miscSiemensLogo"></AwIcon>
```

```
    </div>
</div>
```

The view compiler would have detected that the `aw_hosting_enabled` and `aw_hosting_config.ShowSiemensLogo` ctx is being used automatically.

#### When do I not need to use ctx?

If your component is using something from a service that does not directly impact the UI, a `ctx` entry is not required. For example if you have an action:

```
{
    "actions": {
        "doMyAction": {
            "actionType": "JSFunction",
            "method": "dostuff",
            "inputData": {
                "0": "{{ctx.myFlag}}"
            },
            "deps": "js/my.service"
        }
    }
}
```

And a UI that will execute that action:

```
import AwIcon from 'viewmodel/AwIconViewModel';

export const AwLogoRenderFunction = ( { actions: { doMyAction } } ) => {
    return (
        <div className='sw-logoContainer'>
            <div className='sw-logoImage'>
                <AwIcon iconId='miscSiemensLogo' onClick={() =>
doMyAction}></AwIcon>
            </div>
        </div>
    );
};
```

You do not need to have `myFlag` in `ctx`.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**What is the difference between derived state and the "onUpdate" hook?**

Derived state and the `onUpdate` hook both have the ability to trigger your action or function when something has changed. For example, in this view model, the `myDerivedContext` and `myOnUpdateContext` will have the same value:

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {
        "getUpdatedData": {
            "actionType": "JSFunction",
            "method": "getUpdatedData",
            "inputData": {
                "props": "{{props}}"
            },
            "outputData": {
                "myOnUpdateContext": ""
            },
            "deps": "js/myService"
        }
    },
    "lifecycleHooks": {
        "onUpdate": {
            "action": "getUpdatedData",
            "observers": [ "viewModel.ctx.myFlag" ]
        }
    },
    "ctx": {
        "myDerivedContext": {
            "parser": {
                "method": "myParser",
                "deps": "js/myService"
            }
        }
    }
}
```

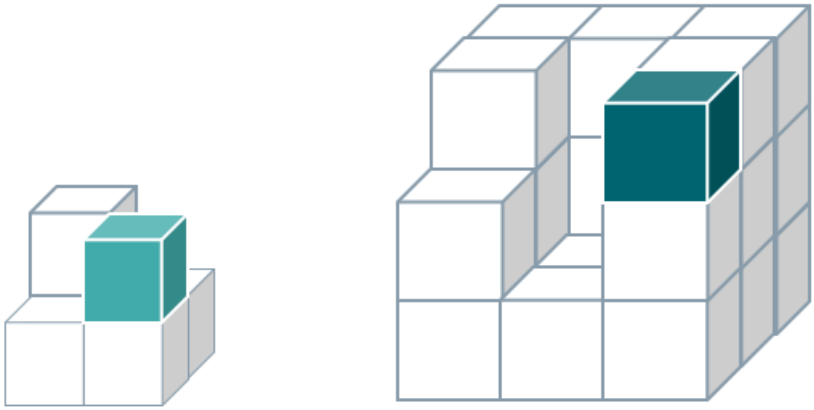
```
export const getUpdatedData = ( flagValue ) => flagValue === 'blue';
export const myParser = () => [ new DerivedStateResult( {
    ctxParameters: [ 'myFlag' ],
    compute: getUpdatedData
} ) ];
```

In both cases, the `getUpdatedData` function will only be called when `ctx.myFlag` changes. Wherever possible, the derived state should be used as it will perform better while simplifying your rendering process. However, if your function does something async (such as make a network call) or has other side effects (such as modifying something in ctx) you will need to use the update hook instead.


Unpublished work. © 2025 Siemens

# UI elements

Vocabulary for the View is made up of **UI elements** (declarative custom elements). UI elements are how we create declarative building blocks and how we teach HTML new tricks. Custom Elements is one part of the W3C Web Components. It is how we extend the vocabulary.

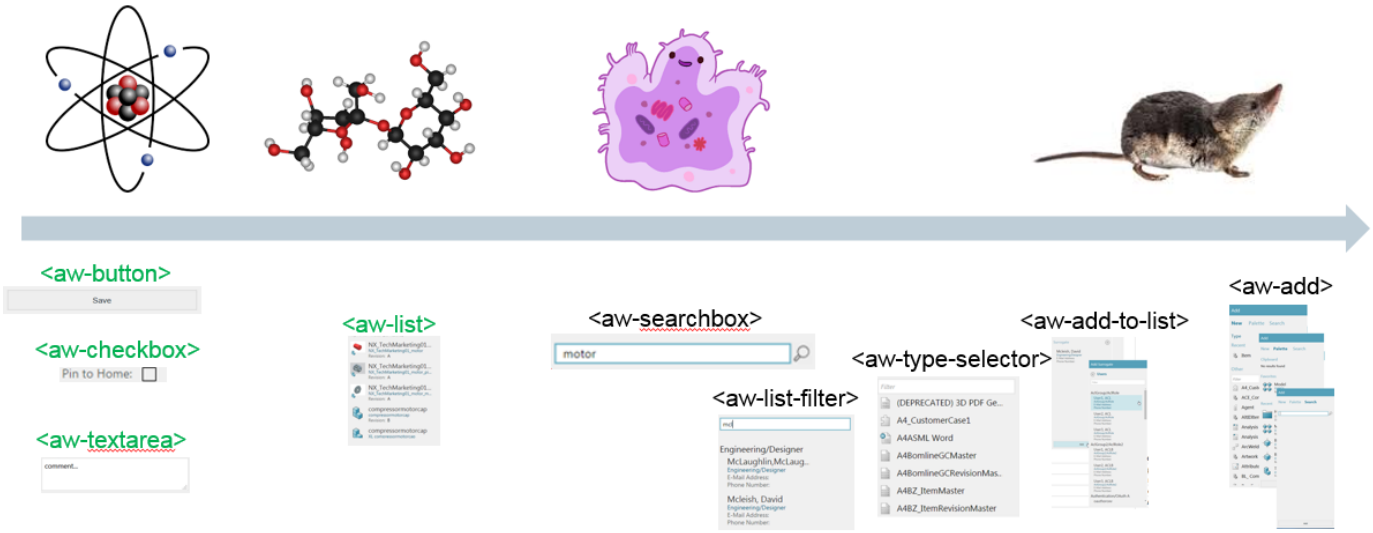


The declarative approach relies on a View which is an HTML Markup and a View Model that has two-way data binding with the View. While HTML5 defines the set of elements and behavior that browsers implement, we can extend that vocabulary with custom elements. The important point to note is that while custom elements can be used by Views declaratively, the definition and implementation of those custom elements (behind the scenes) is very much code-ful. In other words, if we wanted to have a fancy-box custom element , the consumers would simply add in their View declaratively; however, the author would need to create a JavaScript module that implements the fancy-box element and its CSS and HTML partial.

## Atoms, molecules,... small mammals

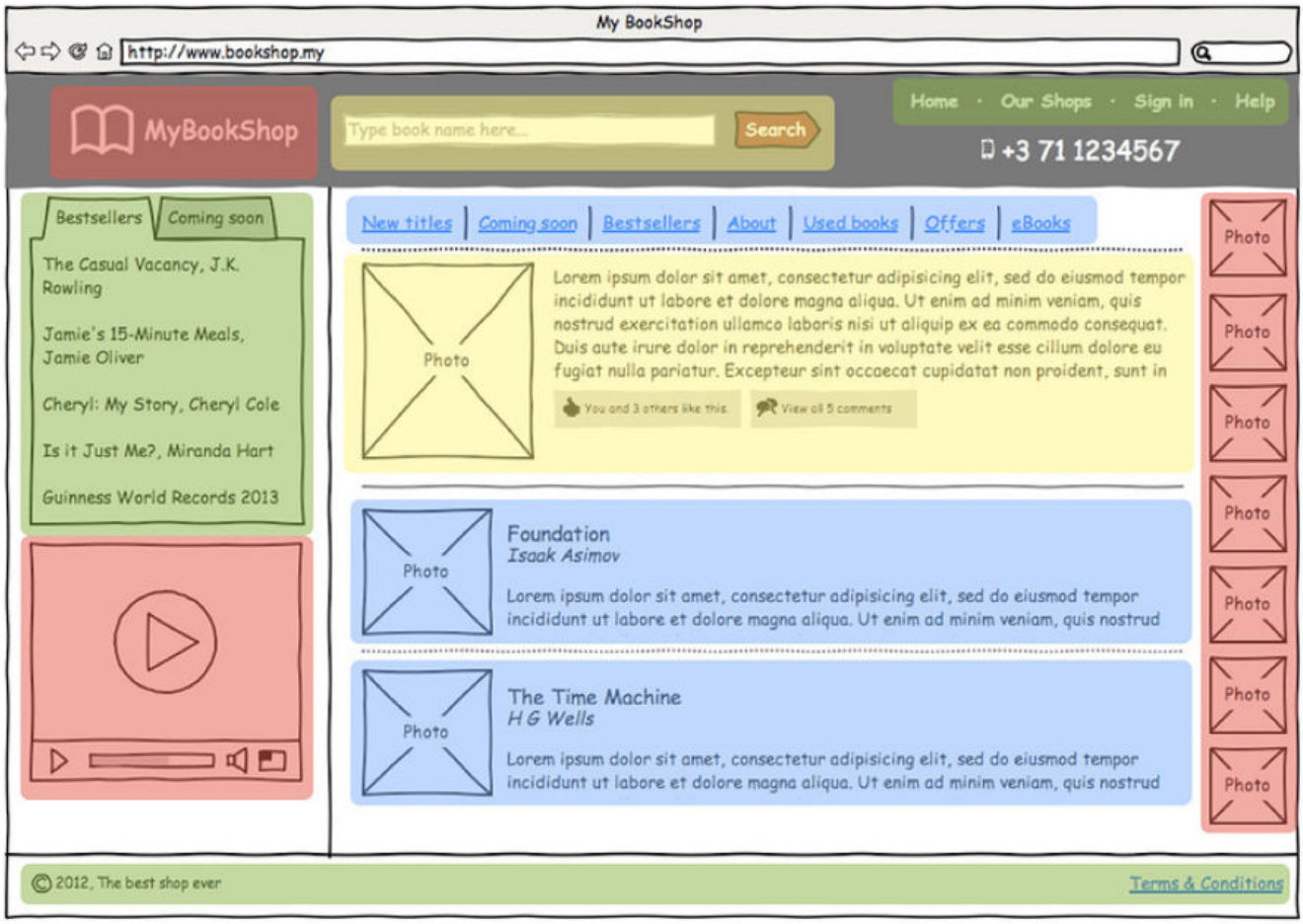
Custom elements are our building blocks. To maximize re-use and consistency we are building a library of UI elements. Those that are applicable to groups across Siemens PL are included with Siemens Web Framework. We are building higher-order elements that build on the atoms and molecules we have created.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## UI elements is bottom up approach

Look at the page below, if you consider page as a complete unit, and of the smaller pieces as belonging to the page. That approach is top-down thinking and results in code that's full of one-offs and special bits that only live a single page. It's not conducive to writing reusable code.




What UI Element asks you to do is step back, and instead of thinking about this at the page level, look at the fact that your page is made up of small chunks of discrete content. This isn't a page. This is a collection of pieces.

You've got a logo, a search bar, navigation, a photo list, a secondary nav, a tabbed box, a video player, etc. These are discrete pieces of content that could be used anywhere in your site. They just happen to be assembled this way on this particular page.

Declarative UI Element is bottom-up thinking. It asks you to start with the reusable building blocks that your entire site is constructed from.

## What UI elements are available?

UI elements are documented and demonstrated in the **Siemens Web Framework Storybook**. It has live examples, view and view model snippets, and additional information for each UI element provided in Siemens Web Framework.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### What if there isn't the UI element I need?

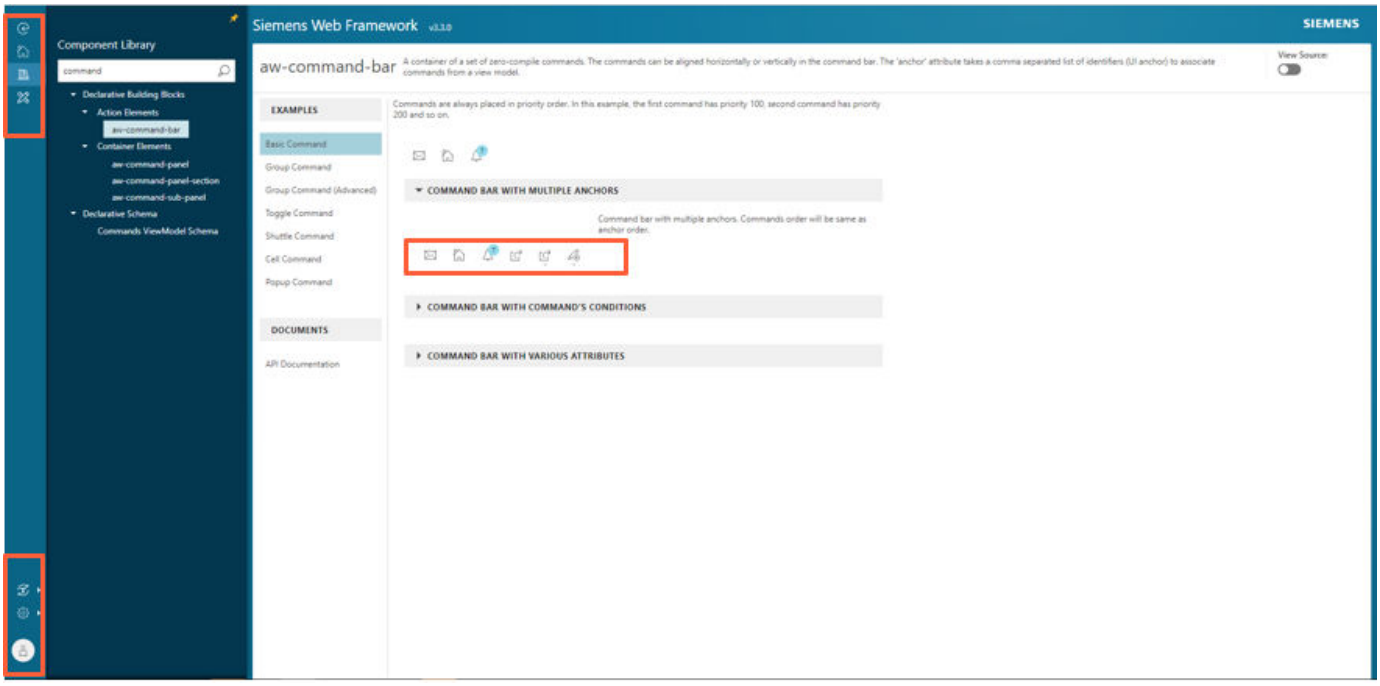
Siemens Web Framework provides base set of building block elements that meets user interface needs from different business needs. The building block elements can be equated to Lego block analogy. Lego blocks come in standard sizes/shapes and colors and one can join them together to achieve complex designs/structures. At times certain designs may not be simply accomplished with available set of Lego blocks, and that leads to requirement for new Lego block.

When specialized elements are required for a particular target industry segment (for example a graphical line balancing element for manufacturing planning engineers), then it is possible to create new UI elements which can then be consumed in your declarative Pages and Panels.

Also, Siemens Web Framework provides a set of **services** which can be used to create your own UI elements.

## Command

Commands in Siemens Web Framework are defined and configured declaratively that provide a graphical means for the user to work with their product, process or other data.

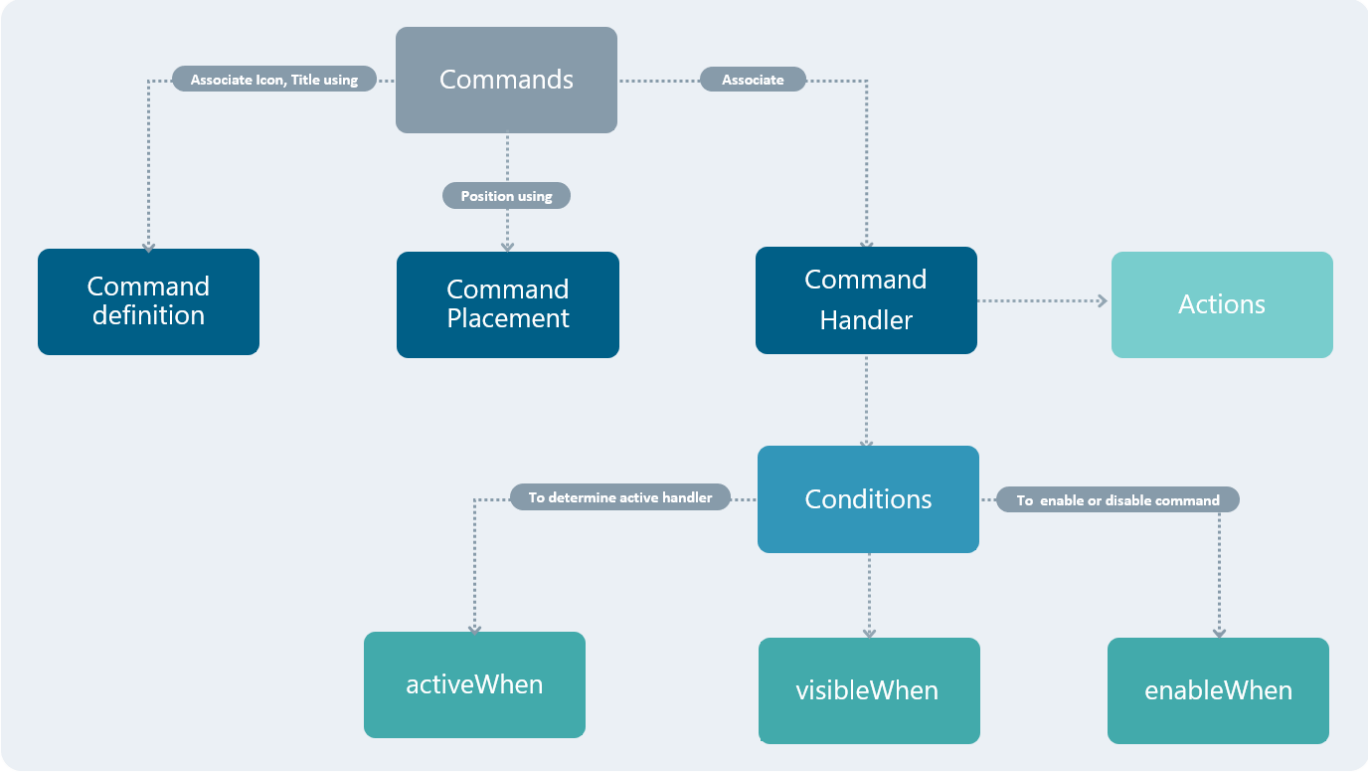



Declarative commands leverage the same View Model concepts such as **localization**, **conditions** and **actions**.

In general, any command needs to have the following while being defined:

- **Appearance** - like an icon, label and a tooltip.
- **Behavior** - such as when it should be presented to the user, when it should be active, visible, enabled, etc.
- **Action** - what the command should do.
- **Position** - where it should be placed in the UI relative to other commands.

In Siemens Web Framework, the View has **command bars** in which **commands** can be placed (the elements). The View Model is responsible for view state such as **data**, **actions**, **conditions** and **localizations** in which we describe the command and its behaviours.




In summary, with declarative commands you can:

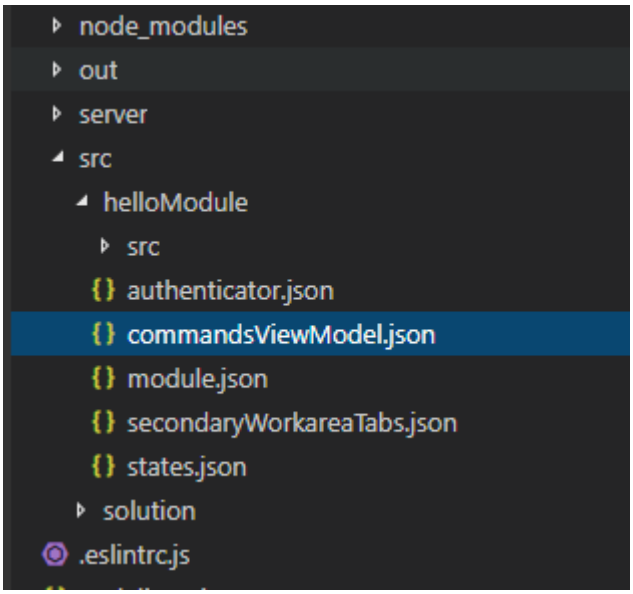
- Define a Command including ID, Icon and Title (aka: tooltip) and Extended Tooltip.
- Define where a command is presented in the UI through well known 'command anchor'(locations) and set the order in which the commands are displayed.
- Define a keyboard shortcut for a Command.
- Define what a command should do (execution) as either a REST invocation, JavaScript function or SOA invocation.
- Define conditional expressions as to when a command is visible and enabled
- Leverage existing Declarative UI View Model concepts such as i18n, Condition, and Action.


Unpublished work. © 2025 Siemens

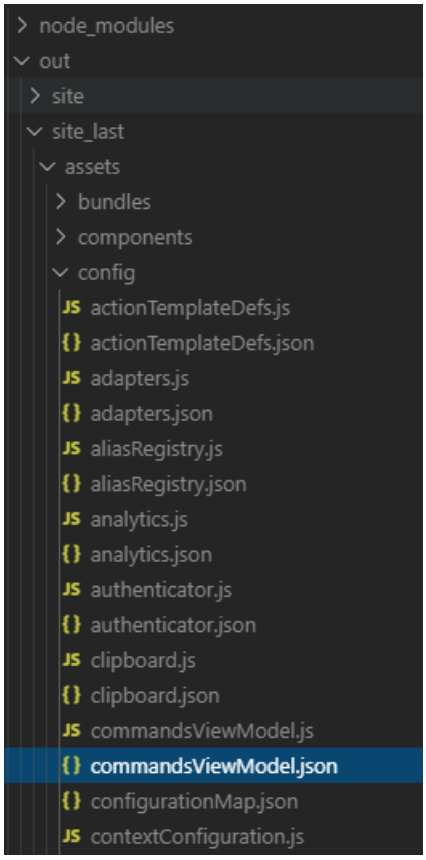
- Use command context to provide additional data that may be necessary to evaluate command visibility or command execution

## How are commands declared

Define your commands view model in a `commandsViewModel.json` file right next your module's module.json file.




When building the *site*, the Siemens Web Framework tooling concatenates all the modules' commandsViewModel.json files into a single `commandsViewModel.json` file.




The browser at runtime returns via an HTTP get action as configured by the Solution configuration.

The `commandsViewModel.json` has the following properties:

## Command Definition

Command definition helps defining a command uniquely. The name of the command (`homeCommand` in example below) is referred as command Id. Command is defined inside commandsViewModel.json file.

```
{
    "commands": {
        "homeCommand": {
                    "iconId": "cmdHome",
                    "title": "{{i18n.homeCommandTitle}}"
                }
        ...
    }
    ...
}
```

The command definition contains the following properties:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Attribute Name | Value Type | Is Required? | Description |
|---|---|---|---|
| **title** | Object | Optional | An **i18n** key or a **parametrized i18n** object or just a non-18n string which is resolved to a value which appears as a tooltip when user hovers over the command. Same is used as command label when labels are turned On. The value is assigned to the ***title*** attribute appears as a rich tooltip for that command. Refer **Extended Tooltip** |
| **description** | String | Optional | An **i18n** key assigned to this is resolved to a value which appears in a rich tooltip along with command *title* when user hovers over the command. Refer **Extended Tooltip** |
| **iconId** | String | Optional | Name of key referring to svg file from image directory. For e.g. **cmdOpen** key refers to `cmd`Open24.svg image. The image name must start with `cmd` and end with `24` |
| **selected** | Object | Optional | It provides a way to **toggle** command **icon** and **title** based on selection state. Refer **Toggle command** |
| **isGroup** | Boolean | Optional | It provides a way to group related commands under a single command. Set to true if grouping is desired. **Default is false**. Refer **Group command** |
| **isShuttle** | Boolean | Optional | It provides a way to group related commands under a single command. Set to true if ***icon and title of selected command*** from group is to be reflected at the group level. It overrides isGroup value. See **Shuttle command** |
| **isRibbon** | Boolean | Optional | It provides a way to group related commands under a single command. Set to true if ***visibility is to be independent of commands in the group***. It overrides isGroup and isShuttle value. |
| **extendedTooltip** | String | Optional | It provides a way to show custom rich tooltip (styled texts, images etc.) in a balloon popup. If not configured, it shows title and 'descsription` attribute value in a rich tooltip. If configured with a custom view, it would show the contents in the custom view. Refer to **Extended Tooltip** |
| **scope** | String | Optional | The expression of the declarative condition which will determine if the command is applicable for further processing of conditions on command handlers like activeWhen/visibleWhen/enableWhen |
| **shortcut** | String | Optional | Keyboard shortcut to be configured for executing a command. Needs to be unique per command |

## Command Placement

A command placement defines actual placement (location) of a command inside application layout. A command can have multiple placements, meaning to say, same command can be shown/placed in more than one location using a unique placement definition within application. A command is placed inside a **aw-command-bar**

Let's have a look at sample **commandPlacement**:

```
{
    "commandPlacements": {
        "homeCommandPlacement": {
            "id": "homeCommand",
            "uiAnchor": "aw_globalToolbar",
            "relativeTo": "Awp0ChangeTheme",
            "priority": -1
        },
    }
    ...
}
```

A command placement has the following properties:

| Attribute Name | Value Type | Is Required? | Description |
|---|---|---|---|
| **id** | String | Required | The command Id for which the placement is associated |
| **uiAnchor** | String | Required | A well known name for a given aw-command-bar element. Anchors could be created where needed by Solutions |
| **priority** | String | Required | The relative priority order to render commands in ascending order unless the command bar has reverse alignment |
| **relativeTo** | String | Optional | The command id of this command will be placed relative to. Priority as defined in the '*priority*' property will be applied relative to the 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 |
| **style** | String | Optional | This determines the style to be applied on the command - 'base' and 'primary' are the supported display styles. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| **showTitle** | String | Optional | This determines whether or not to show the command label - 'always' and 'never' are the supported values. If it is not defined then command label will honor the global label setting. |
| **hideOnSingleChoice** | Boolean | Optional | This provides a way to hide group/shuttle command if it contains single visible command. |

## Command Handlers

A command can have many **Command Handlers**. However at any given time, there may at most be only one *active* commandHandler for a command. A declarative **"activeWhen"** **condition** which 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. By being active, the command handler will determine:

1. When a command is visible and enabled.
2. What a command will do when executed is specified by the declarative **"action"** property for the handler.

A command handler has the following properties:

| Attribute Name | Value Type | Is Required? | Description |
|---|---|---|---|
| **id** | String | | The command Id for which the handler is associated |
| **action** | String | | The name of the declarative **action** (REST,SOA,Event or JSFunction) to be executed |
| **activeWhen** | Object | | The name of the declarative **condition** which will determine when this command handler is active for the associated command |
| **visibleWhen** | Object | | The name of the declarative condition which will determine when the associated command is visible. This condition is only evaluated if this command handler is active. When not defined default is true |
| **enableWhen** | Object | | The name of the declarative condition which will determine when the associated command will appear enabled or disabled. This condition is only evaluated if the command handler is active and command is visible. When not defined default is true |
| **selectWhen** | Object | | The name of the declarative condition which will determine when the associated command will be selected or deselected. This condition is only evaluated if the command handler is active and command is visible. When not defined default is false. |


Unpublished work. © 2025 Siemens

Here are some examples of commandHandlers:

```
    "commandHandlers": {
        "cmdShowObjectDevConfigCommandHandler": {
            "id": "cmdShowObjectDevConfig",
            "action": "activateShowObjectDevConfigNavPanel",
            "activeWhen": {
                "condition":
"conditions.cmdShowObjectDevConfigCommandActive"
            },
            "visibleWhen": {
                "condition":
"conditions.cmdShowObjectDevConfigCommandActive"
            }
        },
        "cmdQuickAccessHandler": {
            "id": "cmdQuickAccess",
            "action": "quickAccessAction",
            "activeWhen": true,
            "visibleWhen": {
                "condition": "conditions.quickAccessVisibility"
            }
        },
        "Awp0GoHomeCommandHandler": {
            "id": "Awp0GoHome",
            "action": "goHome",
            "activeWhen": true,
            "visibleWhen": true,
            "enableWhen": true,
            "selectWhen": true
        },
        "checkoutHandler": {
            "id": "Awp0Checkout",
            "action": "isDirty",
            "activeWhen": {
                "condition": "conditions.checkSelections"
            },
            "visibleWhen": {
                "condition": "conditions.checkoutVisibility"
            },
            "enableWhen": {
                "condition": "conditions.enableWhenCommandCondition"
            },
            "selectWhen": {
                "condition": "conditions.checkSelectionOfCommand"
            }
        }
    }
```

## Command Labels

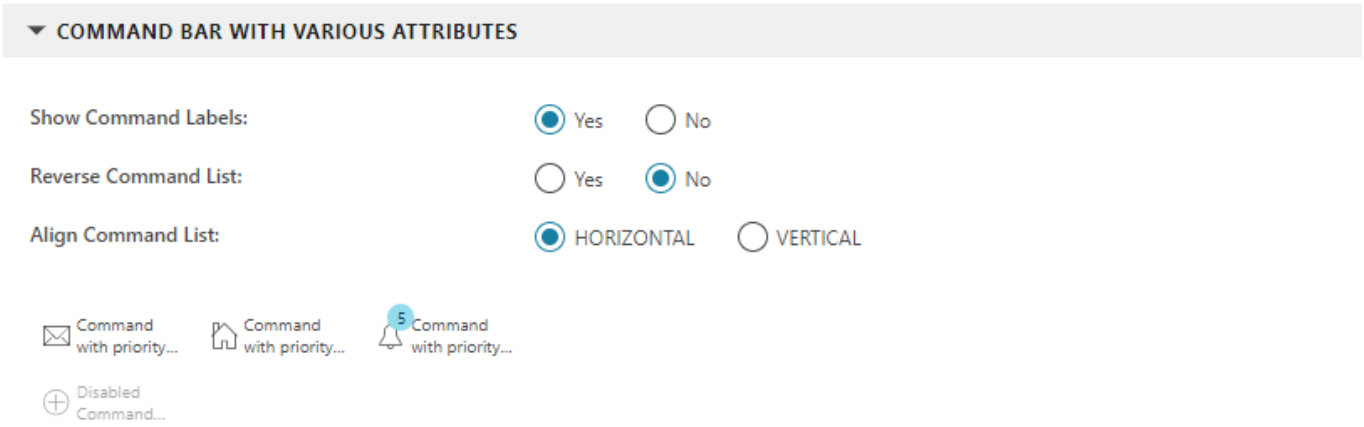
Command labels are configurable through command bars. A command bar can be configured to show labels for all active and visible commands in it at a time. All commands in a command bar can


Siemens Web Framework
Unpublished work. © 2025 Siemens

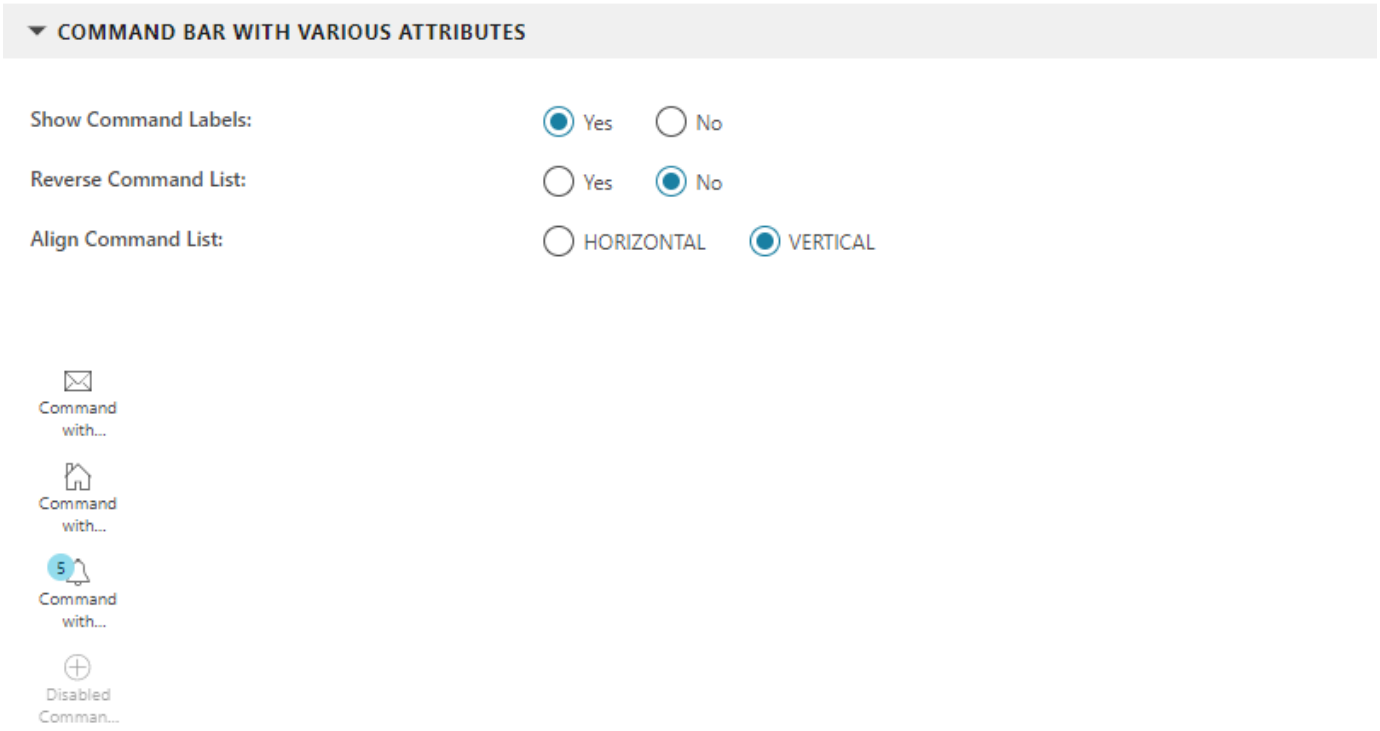
have labels turned off or on using the `show-command-labels` attribute on `aw-command-bar`. Refer to **Component Library->aw-command-bar->Basic Command->Command Bar with Various Attributes** example in Storybook to learn how to configure.

position of the label is not configurable. It appears on the side for horizontal and below for vertical commands.

Horizontal command bar with label:




Vertical command bar with label:

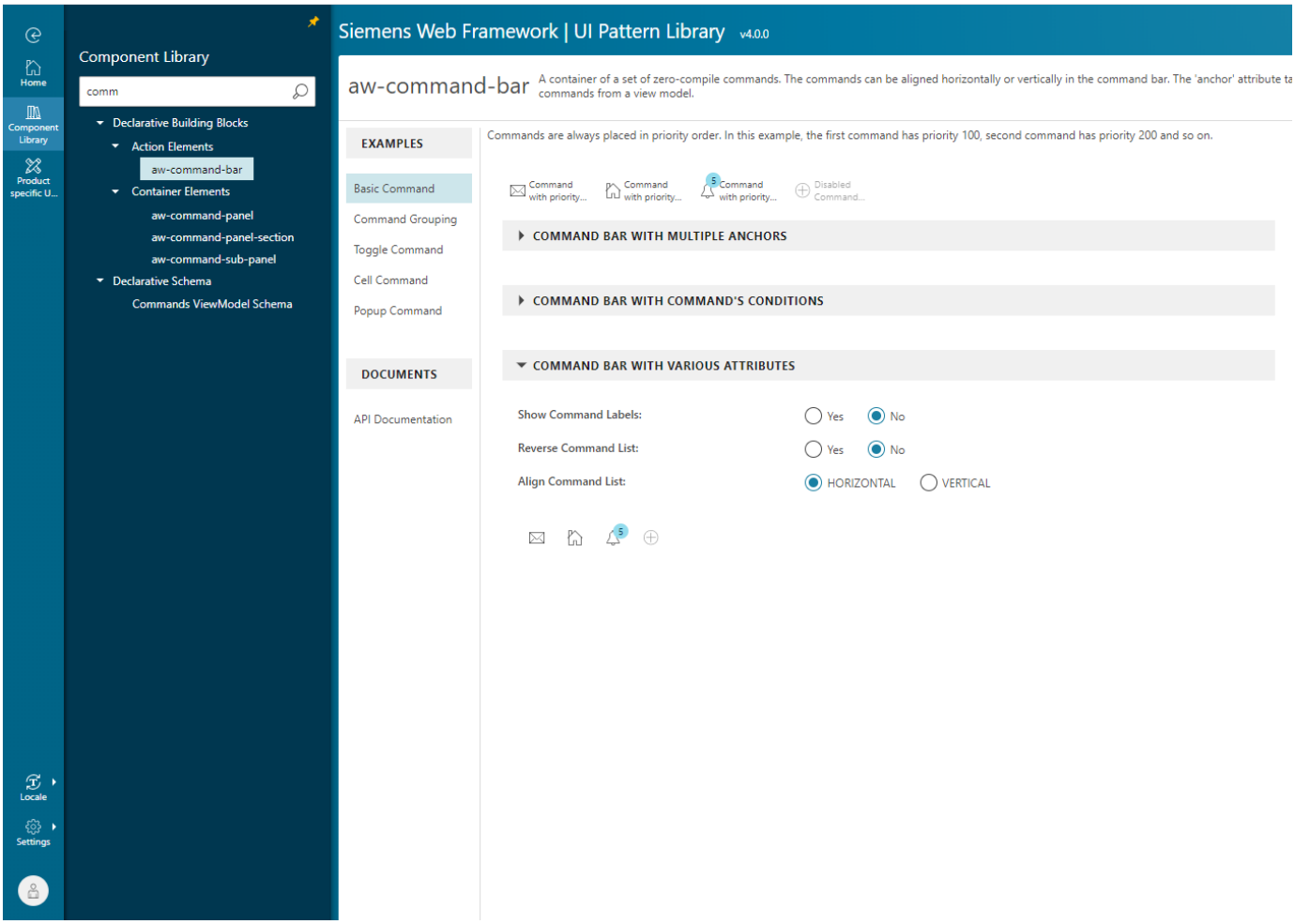



To add to this, the labels can be turned on or off globally as well. However, this needs to be done in the application body using code. To set it, add `aw-commands-showIconLabel` class on body, and all commands on page will show label.

```
<html>
    <head>...</head>
    <body class=".... aw-commands-showIconLabel">
        <!-- UI View -->
        <div>...</div>
    </body>
</html>
```

Global command bar with label, set particular command bar label off:


Siemens Web Framework
Unpublished work. © 2025 Siemens




Individual command bar setting for labels take a higher priority than the global setting. This implies that if a command-bar A has set the `show-command-labels` value ON/OFF, the global setting won't have any impact on the command-bar A. Only for the command-bars without any `show-comnand-labels` setting, the global setting will be applied.

As of now, labels are not turned off for child commands in a group/shuttle command, as icons are not supported there yet.

## Command Shortcuts

**There are few rules for command shortcuts to work**

1. Shortcut should be unique per command. E.g. "Alt+A" taken for any specific command, cannot be taken for any other command.
2. Browser reserved shortcuts cannot be used.
3. Single key shortcuts cannot be used.

**Special case when Command appears at multiple places at the same in an application**

If a command appears at multiple places at the same time (say, Primary Workarea and Secondary Workarea ), then on hitting the shortcut on keyboard, a command which belongs to "focused component" would be executed.

E.g. If Primary Workeara is active, then command from Primary Workarea toolbar would be executed.

Applications using custom views need to do 2 things:

1. Need to keep information of which component is active at "subPanelContext.pageContext.sublocationState.value.focusComponent".
2. Components hosting toolbars/commandbars need to keep info about themselves at subPanelContext.parentComponent.

## Command Context

Support for Cell Commands required the introduction of a new **context** property on the aw-command-bar.

```
<aw-command-bar anchor="{{firstAnchor}}" context="{vmo: item}"></aw-command-bar>
```

This **context** is any additional data that may be necessary to evaluate command visibility or execute commands. This allows you to have a single command anchor that is shared between multiple different views (meaning the same set of commands is in each view) while still allowing the commands to execute with information specific to where the command bar is used.

## Sample Use Case (Cell Commands)

## When is this used?

Commands within a cell are typically expected to act on the object that is displayed in that cell. For example the "Open" command is expected to open the the cell object, not the selected object. Therefore the cell must have some way to pass on that information to the command bar.

## How is this used?

aw-list UI element will create a **context** for each of cells it hosts. This context has a single "vmo" property which is the object displayed in the cell.

The object displayed in the cell is now available in the *commandsViewModel* as "commandContext.vmo" and can be used as documented on the **cell commands wiki**


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Actions and i18n

The command usages of **actions**, **conditions** and **i18n** in `commandsViewModel.json` are the same declarative **View Model** concepts used elsewhere for **declarative UI**.

Here is an **action** to call a SOA/REST Api to undertake a particular action which could be used in a command handler. As an example, we are performing a "checkout" on an object.

```
    "actions": {
        "performCheckout": {
            "actionType": "TcSoaService",
            "serviceName": "Core-2006-03-Reservation",
            "method": "checkout",
            "inputData": {
                "objects": "{{data.checkoutInput}}"
            },
            "outputData": {
                "checkOutPartialErrors": "partialErrors"
            },
            "actionMessages": {
                "failure": [{
                    "condition": "(ctx.mselected && ctx.mselected.length
=== 1)",
                    "message": "checkoutFailureSingle"
                }, {
                    "condition": "(ctx.mselected && ctx.mselected.length >
1)",
                    "message": "checkoutFailureMultiple"
                }]
            }
        }
    }
```

Another example that shows an **action** that reveals a slide out command panel for which there is a declarative **View** and **View Model** associated:

```
    "actions": {
        "quickLinkAction": {
            "actionType": "JSFunction",
            "method": "activateCommandPanel",
            "deps": "js/commandPanel.service",
            "inputData": {
                "commandId": "cmdQuickLinks",
                "location": "aw-navigation"
            }
        }
    }
```

Declarative **conditions** can be defined with the required expressions utilizing **application context**. Additional states can be added using the **application context** service.

```
    "conditions": {
        "haveSearchResultsAndEditNotActive": {
            "expression": "ctx.search.totalFound > 0 &&
ctx.someState.inEdit"
        },
        "alertItemRevisionCommandVisible": {
            "expression": "ctx.selected"
        },
        "alertItemRevisionCommandActive": {
            "expression": "ctx.selected.type === 'ItemRevision'"
        },
        "alertItemRevisionSpecialCommandActive": {
            "expression": "ctx.selected.type ==='ItemRevision' &&
ctx.selected.prop.object_name.dbValue === 'qwerty'"
        }
    }
```

## Types of Commands

In addition to regular commands, the following other types of commands can be created:

- **Command grouping** - List of commands, visible command icon doesn't change
- **Toggle Commands** - Same icon, background turns on or off
- **Cell Commands**
- **Commands in Command Panel & Panel Section**
- **Commands in Tables**
- **Context menu**

## Command decorators

It is possible to display additional information within a command using command decorators. The command decorator will be displayed to the right of the command icon.

In this example the command decorator is displaying the number of notifications in a blue circle.

Siemens Web Framework
Unpublished work. © 2025 Siemens

### Defining a command decorator

A command decorator is defined as part of the command definition

```
{
    "commands": {
         "notificationCommand": {
            "iconId": "cmdAlerts",
            "title": "Command with priority 300",
            "template": "<div>{{ctx.selectedNode.examples.length}}</div>"
        }
    }
}
```

The **template** property defines the contents of the decorator. The template supports data-binding and is able to bind to any information available in the application context.

### When to use scope on command definition vs visibleWhen/activeWhen on command handler?

If the "Scope" in the command definition is evaluates to false, the command engine will immediately stop processing any further command handlers. It won't even check if any handlers are active or visible. Use "Scope" to set conditions when the command isn't relevant, such as for different sub-locations or license levels.

On the other hand, "visibleWhen" and "activeWhen" in a command handler are specific only to that particular handler. These settings don't affect other handlers. Use them when you need to override the behavior of a specific command.

### Invoking a command programatically

A command can be executed programatically by using the action type `Command`; see Command for more information.

## Group command

Group command supports grouping of related actions under a single command. There are three flavors of group commands supported

| Type | How to | Description |
|---|---|---|
| Group command | Set `isGroup` to true | It provides a way to group related commands under a single command |

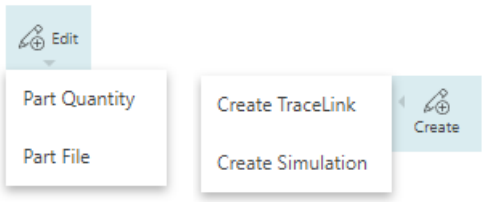
| | | |
|---|---|---|
| Shuttle command | Set `isShuttle` to true | It provides a way to group related commands under a single command. The *`icon and title of selected command`* from group is reflected at the group level. It overrides isGroup value |
| Ribbon command | Set `isRibbon` to true | It provides a way to group related commands under a single command. The *`visibility is independent of commands in the group`*. It overrides isGroup and isShuttle value |

## Group command

Say you want to provide `Edit Part Quantity` and `Edit Part File` actions, you can group them under `Edit` action and use drop down to list `Part Quantity` and `Part File` actions.

A group command shows triangular wedge indicator

## Example of Group command




## How to use Group command

1. Set the command's "*isGroup*" property to "true"
2. All of its child commands should set "*parentGroupId*" as the group command's id.

## Behavior of Group command

- If there is only one child command, the default action is to execute the child command's action
- If there are more than one active child commands, the default action is to show popup
- Ideally Group command itself never has an action - since clicking on a group reveals a popup with the command variants for the group
- Don't need to have a "*visibleWhen*" property for a group command to set it's visibility. Its visibility is dependent on its children
- If the group command has only one active child command, then the triangle icon will not be shown by default


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Example of a group command

```
"commands":
{
    "MyGroupCommand": {
        "iconId": "cmdWalk",
        "isGroup": true,
        "title": "{{i18n.MyGroupCommandTitle}}"
    },
    "MyCheckout": {
        "iconId": "cmdCheckout",
        "isGroup": false,
        "title": "{{i18n.MyCheckOutTitle}}"
    },
    "Awp0AlertUser": {
        "iconId": "cmdAlerts",
        "isGroup": false,
        "title": "Alert"
    },
```

```
"commandHandlers":
{
    "MyGroupCommandHandler":
    {
        "id": "MyGroupCommand",
        "activeWhen": true
    },

    "myCheckoutCommandHandler":
    {
        "id": "MyCheckout",
        "action": "checkItOut",
        "activeWhen": true,
        "visibleWhen": true
    },

    "itemCopyCommandHandler":
    {
        "id": "Awp0AlertUser",
        "action": "alertItemRevision",
        "activeWhen":
        {
            "condition": "conditions.alertItemRevisionCommandActive"
        },

        "visibleWhen": true
    },
```

```
"commandPlacements":
{
    "myGroupCommandOneStep": {
        "id": "MyGroupCommand",
        "uiAnchor": "aw_oneStep",
        "priority": 50
    },
    "myCheckoutOneStep": {
        "id": "MyCheckout",
        "uiAnchor": "aw_oneStep",
        "priority": 50,
        "parentGroupId":"MyGroupCommand"
    },

    "itemCopyOneStep": {
        "id": "Awp0AlertUser",
        "uiAnchor": "aw_oneStep",
        "priority": 100,
        "parentGroupId":"MyGroupCommand"
    },
```

## Showing selection state of children

To keep the selected highlighted state for the child commands, set the `"showGroupSelected":true`

```
"commandPlacements": {
        "cmdGroupCommandTAI":
        {
            "id": "TAIGroupCommand",
            "uiAnchor": "aw_toolsAndInfo",
            "priority": 50
        },

        "childQuickLinksTAI":
        {
            "id": "declarativexrtQuickLink",
            "uiAnchor": "aw_toolsAndInfo",
            "priority": 50,
            "parentGroupId": "TAIGroupCommand",
            "showGroupSelected":true
        },

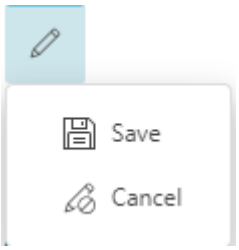
        "childAlertTAI":
        {
            "id": "declarativexrtWidget",
            "uiAnchor": "aw_toolsAndInfo",
            "priority": 50,
            "parentGroupId": "TAIGroupCommand",
            "showGroupSelected":true
        }
}
```

## Hiding the group command chevron

To hide the chevron included by default on all group commands, set the `"showChevron":false` on the group command placement

```
"commandPlacements": {
    "declCommand": {
        "id": "declCommandId",
        "uiAnchor": "aw_ZeroCompileCommandAnchor",
        "priority": 200,
        "showChevron": false
    },
}
...
```

With chevron:


Without chevron: 


## Hiding the group command if it contains single visible command

To hide the group command if it has single visible child command, set the `"hideOnSingleChoice":true` on the group command placement

```
"commandPlacements": {
    "declCommand": {
        "id": "declCommandId",
        "uiAnchor": "aw_ZeroCompileCommandAnchor",
        "priority": 200,
        "hideOnSingleChoice": true
    },
}
...
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Showing a popup in group command if it contains a single visible command

By default the group command will inherit the handler of the child command. To prevent this, set the `"showPopupOnSingleCommand":true` on the group command definition.

```
"commands": {
    "declCommand": {
        "iconId": "declCommandIcon",
        "isGroup": true,
        "title": "Title for declCommand",
        "showPopupOnSingleCommand": true
    },
}
...
```

## Different Display Types for Group Commands

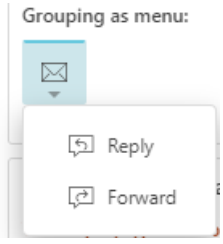
There are 3 types of displayTypes supported for Group Commands. This configuration can be specified on the command placement and it results into different display of the group commands.

### 1. DisplayType: menu

- This is the default display type.
- When specified on the command placement, it will display the group command as a popup menu.
- Here is the example code: Set `displayType: menu` on the command placement of the Group command

```
"respondCmdPlacement": {
    "id": "respondCmd",
    "uiAnchor": "anchorName",
    "displayType": "menu",
    "priority": 100
}
```

- Here is how the display of this Group Command with two child commands would look like:

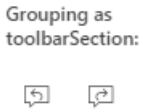


## 2. DisplayType: toolbarSection

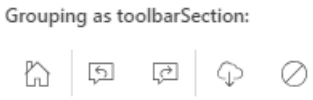
- This is the display type that helps in rendering commands as collections with both horizontal and vertical separators(as applicable).
- When specified on the command placement, it will display the group command spread directly on the toolbar.
- Here is the example code: Set `displayType: toolbarSection` on the command placement of the Group command

```
"respondCmdPlacement": {
    "id": "respondCmd",
    "uiAnchor": "anchorName",
    "displayType": "toolbarSection",
    "priority": 100
}
```

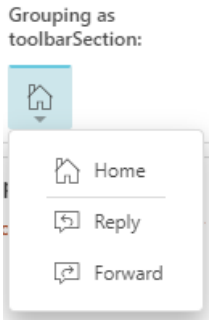
- Here is how the display of this Group Command with two child commands would look like:




- Here is how the display of one single command and two group commands with two child commands each would look like (vertical separators to demarcate the different command sections or collections):




- Here is how the display of one single command and one group command with two child commands would look like (horizontal separators to demarcate the different command sections or collections):




## 3. DisplayType: splitButton

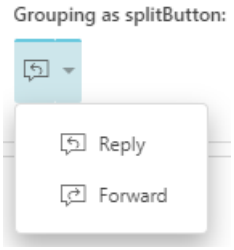
- This is the display type that helps in rendering commands as split commands.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- When specified on the command placement, it will display the first child command in the group as the head command and rest would be displayed as a popup menu.
- Here is the example code: Set `displayType: splitButton` on the command placement of the Group command

```
"respondCmdPlacement": {
    "id": "respondCmd",
    "uiAnchor": "anchorName",
    "displayType": "splitButton",
    "priority": 100
}
```

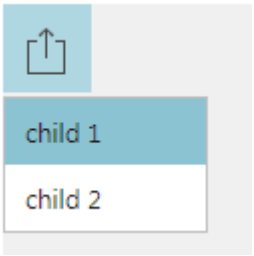
- Here is how the display of this Group Command with two child commands would look like (The first child command is rendered as the head command for quick access):



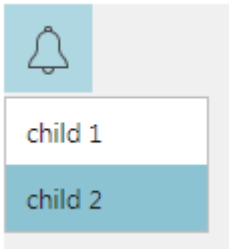

## Shuttle command

A *Shuttle command* can be used when you want to list of commands and change the visible command icon to selected command.

Shuttle command icon changes to 1st command from list:




Shuttle command icon changes to 2nd command from list:




## How to configure a shuttle command

```
commands:
{
        "shuttleCommand": {
            "isShuttle": true
        },
        "childShuttle1": {
            "iconId": "cmdCheckout",
            "title": "child 1"
        },
        "childShuttle2": {
            "iconId": "cmdAlerts",
            "title": "child 2"
        }
},
"commandHandlers": {
        "shuttleCommandHandler": {
            "id": "shuttleCommand",
            "activeWhen": true,
            "action": "shuttleCommandAction"
        },
        "childShuttle1CommandHandler": {
            "id": "childShuttle1",
            "action": "childShuttle1Action",
            "activeWhen": true,
            "visibleWhen": true,
            "selectWhen": {
                "condition": "conditions.cmdShuttle1SelectWhenInShowcase"
            }
        },
        "childShuttle2CommandHandler": {
            "id": "childShuttle2",
            "action": "childShuttle2Action",
            "activeWhen": true,
            "visibleWhen": true,
            "selectWhen": {
                "condition": "conditions.cmdShuttle2SelectWhenInShowcase"
            }
        }

},
"commandPlacements": {
        "declShuttleCommand": {
            "id": "shuttleCommand",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "uiAnchor": "aw_ZeroCompileShuttleCommandAnchor",
            "priority": 200,
            "showGroupSelected": false
        },
        "childShuttle1OneStep": {
            "id": "childShuttle1",
            "uiAnchor": "aw_ZeroCompileShuttleCommandAnchor",
            "priority": 200,
            "parentGroupId": "shuttleCommand"
        },
        "childShuttle2OneStep": {
            "id": "childShuttle2",
            "uiAnchor": "aw_ZeroCompileShuttleCommandAnchor",
            "priority": 200,
            "parentGroupId": "shuttleCommand"
        }
}
```

### Different Display Types for Shuttle Commands

The same displayTypes which are available for **Group Commands** are supported for Shuttle Commands as well. But there is a difference in the splitButton display with respect to the head command.

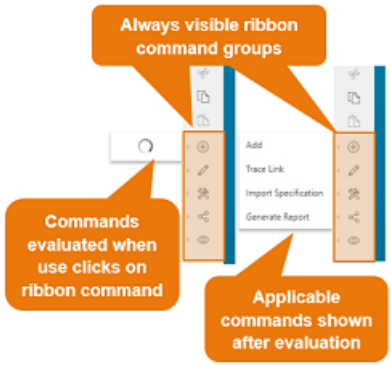
When displayType `splitButton` is specified on the command placement of a Shuttle Command, it will display the last selected child command in the group as the head command and rest would be displayed as a popup menu. Basically, the head command would keep changing just like what is expected from a shuttle command.

## Ribbon command

The primary driver for the new group type was to provide a command group that remains visible and in a consistent position in the user interface. This type of command group differentiates itself from standard command groups in that their visibility is determined irrespective of the visibility of the commands in the group. The command group can have its own visible when and enable when conditions that act independently of its content. The ribbon command groups shown here have both their visibility and enablement attributes set to true. You should also note that they otherwise appear the same as any other command group including the little triangle in the bottom-left corner that indicates that the command is a group of other commands.

A side benefit of ribbon commands is that the evaluation of the content of the group is delayed until the user selects on the command group. This improves the overall performance by delaying the evaluation of those commands until needed. There could be several thousand commands in the system and some of them could have fairly complex visibility logic. By reducing the number of commands to process on every selection change, we have noticeably improved the performance.

### Example of Ribbon command




### Behavior of Ribbon command

- Ribbon command groups remain visible at all times to retain consistent positioning
- Differs from standard command groups in that ribbon command groups visibility is determined irrespective of the visibility of the commands in the group
- Displays the same as standard command groups with the triangle indicating the stack
- Delays evaluation of the visibility of the commands in the group until user clicks for better performance

## Toggle command

A **Toggle command** can be used when we want to show the state of the **command** as selected/unselected (or on/off).

Toggle command in **On** state:

Toggle command in **Off** state:

### How to use Toggle command

- In the **command handler** add the "*selectWhen*" property.
- "`selectWhen`" **condition** determines whether a **command** will be in selected or unselected state.
- "`selectWhen`" property is mandatory for a toggle command.
- If you want a different *icon* and *title* for when selected/unselected you can provide the "`selected`" property in the command definition.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- "iconId" and "title" from "selected" will be used when toggle command background is highlighted i.e selected.
- "*selected*" is an optional property. If not provided then the icon and title will remain the same for selected and unselected states.

Example of toggle **command**:

```
  "Awp0FilterColor": {
            "iconId": "cmdDisplayColor",
            "isGroup": false,
            "title": "Hide Color Filtering",
            "selected": {
              "iconId":"cmdWalk",
              "title": "Quick Links"
            }
        }
```

Example of toggle `commandHandler`:

```
"Awp0FilterColorHandler":{
        "id": "Awp0FilterColor",
            "action": "doMethodInXRT",
             "activeWhen":{
               "condition": "conditions.showInDeclaretivexrt"
               },
            "visibleWhen": {
                "condition": "conditions.showInDeclaretivexrt"
            },
            "selectWhen":{
            "condition":"conditions.cmdWalkerSelectWhenInXRT"
            }
        }
```

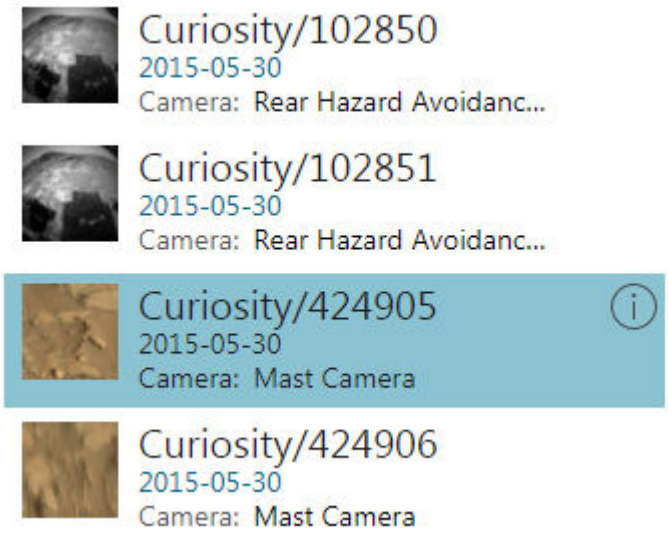
As a result different icons and command titles are presented when in unselected and selected states:

### Alternative command handlers

Note - If you want to have a different handler for an existing toggle **command** then you should add your handler in your respective `commandsViewModel.json` with the `selectWhen` property.

## Cell command

**Commands** can be used in **cells**, such as in a list (aw-list).




## How to use cell commands

### Step 1

Specify the commands anchor that your data provider will use. This commands anchor will determine the set of commands displayed in each cell of the list, table, or tree. This anchor is defined by specifying a "commandsAnchor" directly within the data provider definition.

```
"schemaVersion": "1.0.0",
"actions": {
    "search": {
        "actionType": "RESTService",
        "method": "GET",
        "inputData": {
            "request": {
                "method": "GET",
                "url": "https://api.nasa.gov/mars-photos/api/v1/rovers/
curiosity/photos?sol=1000&page={{(data.dataProviders.mars.startIndex / 25)
+ 1}}&api_key={{ctx.nasaApiKey}}",
                "withCredentials": false
            }
        },
        "outputData": {
            "searchResults": "data.photos"
        }
    }
},
"dataProviders":
{
    "mars": {
        "action": "search",
        "response": "{{data.searchResults}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "totalFound": 200,
        "commandsAnchor": "nasa_cell_commands"
    }
}
```

### Step 2

Similar to any other type of **command**, declare a command in `commandsviewmodel.json` file of your module. The command has the regular **command**, `commandHandler`, and `commandPlacement`. Take note of the `uiAnchor` value of the `commandPlacement`.

```
"commands": {
    "cmdShowPictureDate": {
        "iconId": "cmdShowInfoPanel",
        "title": "Show Date"
    }
},
"commandHandlers": {
    "cmdShowPictureDateHandler": {
        "id": "cmdShowPictureDate",
        "action": "alertDate",
        "activeWhen": true,
        "visibleWhen": {
            "condition": "conditions.mastCameraPicture"
        }
    }
},
"commandPlacements": {
    "cmdShowPictureDatePlacement": {
        "id": "cmdShowPictureDate",
        "uiAnchor": "nasa_cell_commands",
        "priority": 1,
        "cellDisplay": {
            "hover": true,
            "position": "TOP_RIGHT",
            "selected": true
        }
    }
},
"actions": {
    "alertDate": {
        "actionType": "JSFunction",
        "method": "showInfo",
        "inputData": {
            "0": "This picture was taken on
{{commandContext.vmo.earth_date}}"
        },
        "deps": "js/NotyModule"
    }
},
"conditions": {
    "mastCameraPicture": {
        "expression": "commandContext.vmo.camera.name === 'MAST'"
```

```
    }
}
```

Note the usage of "commandContext.vmo". The command context of a command bar displayed within a cell has a single property "vmo" which is the object displayed in that specific cell.

Also note the usage of "cellDisplay" within the command placement. This is additional placement information specific to commands displayed within a cell.

```
"cmdShowPictureDatePlacement": {
    "id": "cmdShowPictureDate",
    "uiAnchor": "nasa_cell_commands",
    "priority": 1,
    "cellDisplay": {
        "hover": true,
        "position": "TOP_RIGHT",
        "selected": true
    }
}
```

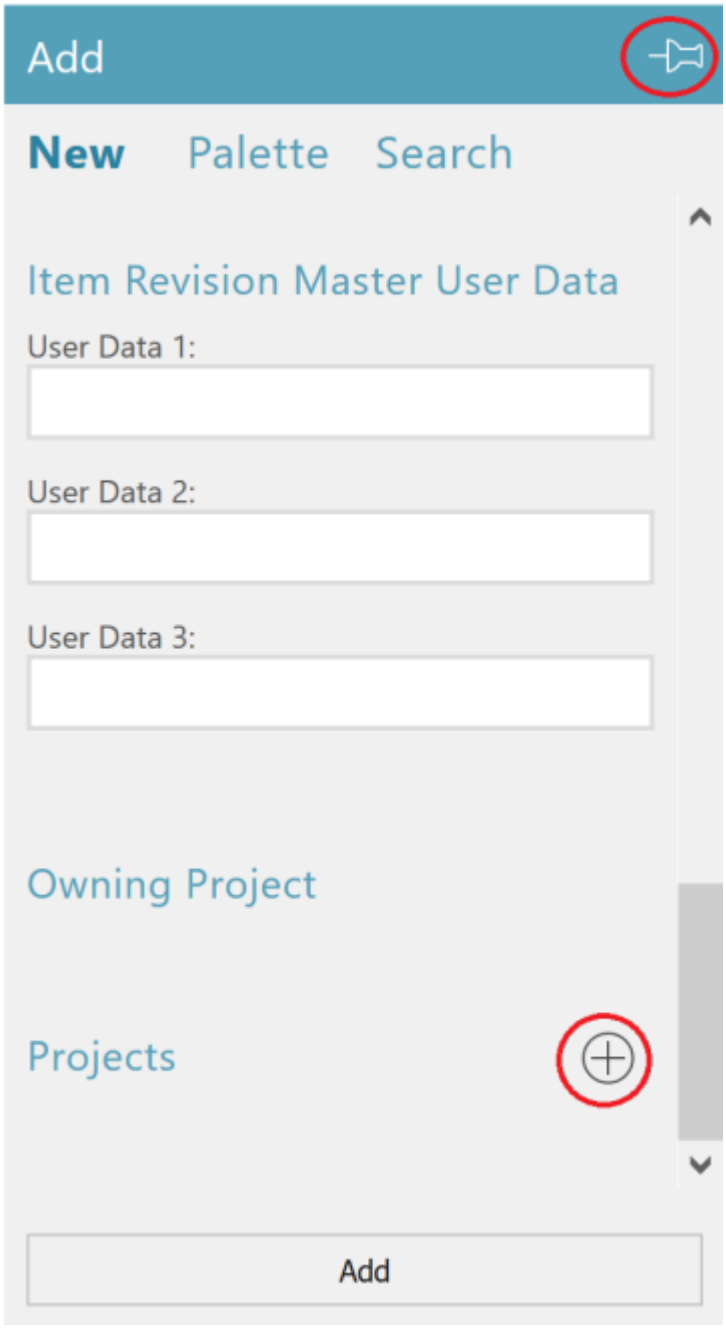
This "cellDisplay" adds the ability to add user interaction visibility on top of any visibleWhen condition that may exist.

- "hover" makes the command display only when the cell is hovered or selected
- "selected" makes the command display only when the cell is selected
- "position" defines a more specific for the placement for the command inside of the cell.
  - Commands may specify the "TOP_RIGHT", "MIDDLE_RIGHT", and "BOTTOM_RIGHT" as their placement.
  - The "RIGHT" part of the placement is for legacy usage, there is no ability for any sort of "LEFT" placement.

## Commands in command panel and panel section

**Commands** can be added to a **Command Panel** header and **Panel Section**.


Siemens Web Framework
Unpublished work. © 2025 Siemens




How to add commands to `<aw-command-panel>` and `<aw-command-panel-section>`.

### Step 1

Similar to any other type of **command**, define a command in `commandsviewmodel.json` file of your module. The command has the regular `command`, `commandHandler`, and `commandPlacement`. Take note of the `uiAnchor` value of the commandplacement.

in `commandsViewModel.json`:

```
    "panelWalkCommandPlacement": {
        "id": "commandPanelWalkCommand",
        "uiAnchor": "commandPanelZCCAnchor",
        "priority": 100
    }
```

### Step 2

In your usage of aw-command-panel or aw-command-panel-section, provide the *anchor* property to associate commands with that placement.

For example:

```
<aw-command-panel caption="i18n.commandPanelDemo"
anchor="data.projectsAnchor">
```

```
<aw-command-panel-section caption="Projects Command Panel Section"
anchor="data.projectsAnchor" >
</aw-command-panel-section>
```

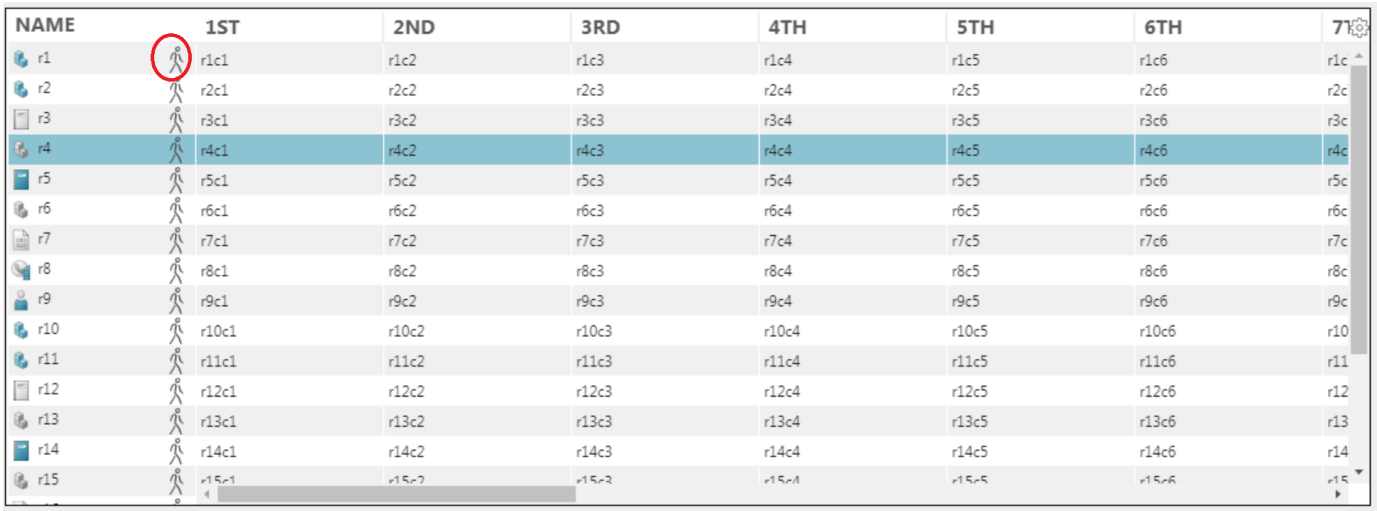
In the **viewmodel** file:

```
    "data": {
        "projectsAnchor":"commandPanelZCCAnchor"
    }
```

## Table commands

**Commands** can be added to a table or tree table.

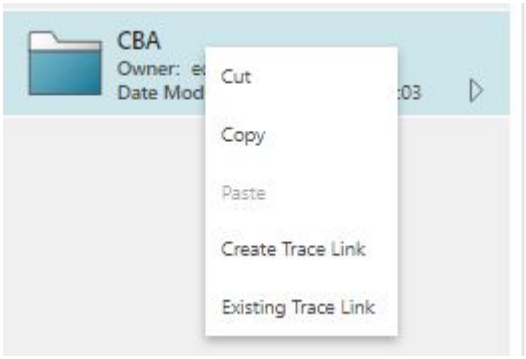

Siemens Web Framework
Unpublished work. © 2025 Siemens




How to add commands to `<aw-splm-table>` is exactly the same as how we **add commands in any cells** such as a list.

## Context menu command

Context menu is a popup with command list. It will open when you right click on list/table items provided that you have enabled it on your list/table.




Any existing or new command can be contributed to a list's/table's context menu. It evaluates all the commands contributed to context menu command anchor and displays them.

### Defining a context menu

You can enable context menu on your list/table by two changes :

1. From viewModel, define context menu command anchor as part of the data provider definition of aw-list/aw-splm-table.

```
{
    "dataProviders": {
         "performSearch": {
            "action": "search",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "contextMenuCommandsAnchor": "aw_listContextMenu"
        }
    }
}
```

The `contextMenuCommandsAnchor` property defines the command anchor name for the context menu. All the commands placed on "aw_listContextMenu" anchor will be shown on command popup opened on list/table item right click.

1. From view, set show-context-menu attribute of aw-list/aw-splm-table to true.

```
<aw-list dataprovider="data.dataProviders.performSearch" show-context-
menu="true">
```

Please refer to *Context Menu* Example in `aw-splm-table` and `aw-ist` in **Siemens Web Framework Storybook**

## Command overflow

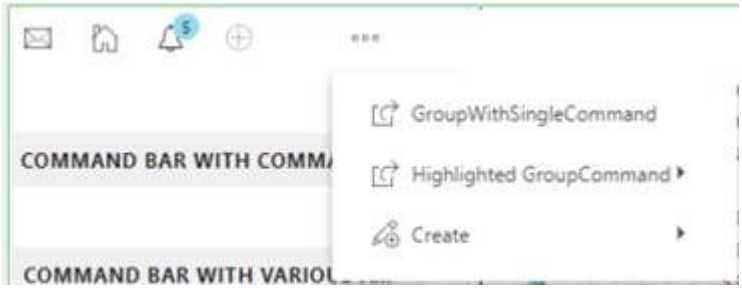
**Note**

Available from **Siemens Web Framework v4.1.0** onwards.

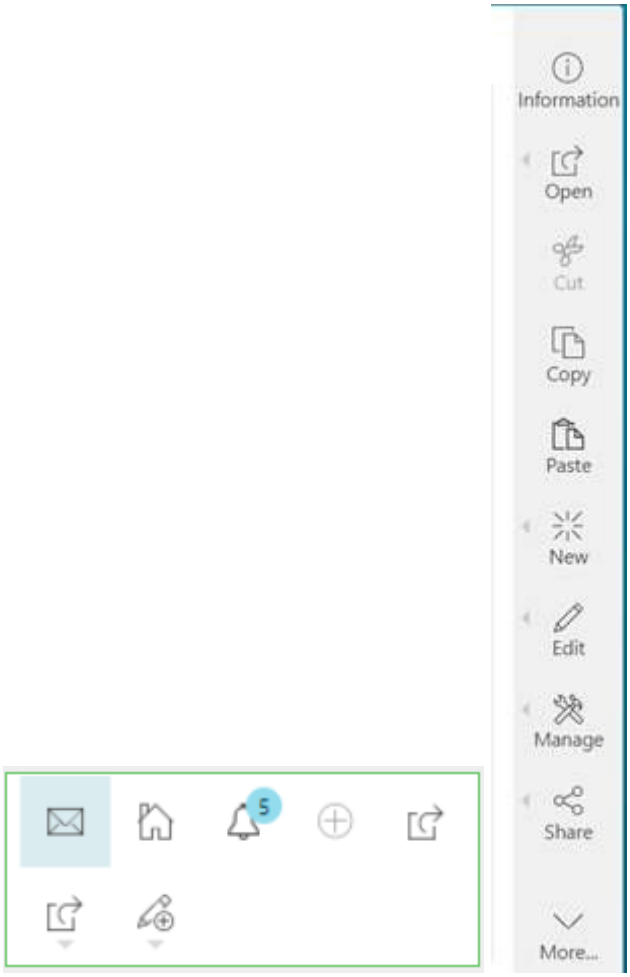
Users have two configurable ways to arrange overflowing commands in command bars and toolbars.

1. `Overflow` - Commands overflow into a "More Commands" button at the right/bottom of the command bars/toolbars. This "More Commands" show up only if commands have overflown due to less space. Clicking it will open a popup which holds all overflowing commands. It works similarly for commandbar and toolbars horizontally or vertically placed. Toolbars containing one or more command bars, by default, would overflow in this way.

Siemens Web Framework
Unpublished work. © 2025 Siemens




1. `Wrap` - Wrap in multiple lines for horizontal command bars. For vertical command bar, a 'More' button is available to open an overlay for overflowed commands. A standalone command bar, would wrap by default.

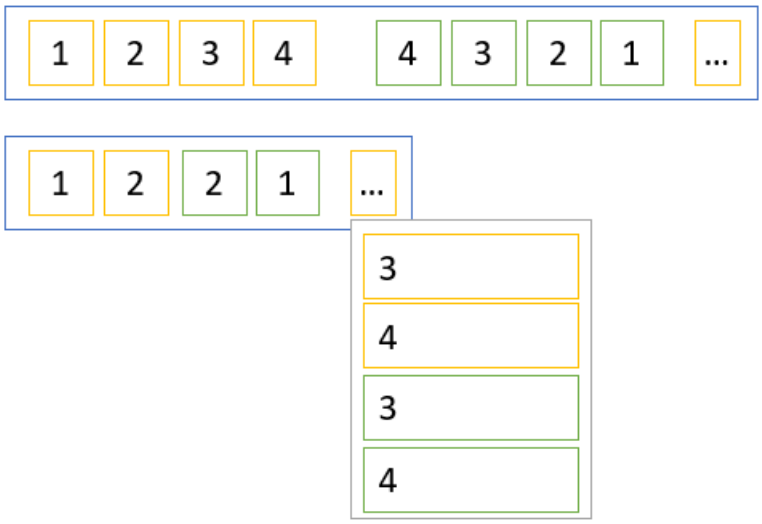



## Containers that are support "Overfow"

- aw-command-bar
- aw-toolbar

## How does "Overflow" work in aw-toolbar:

`aw-toolbar` containes two command bars, one on its left and the other on its right. When tool bar space is limited, the commands start dropping into the overflow from the middle. So, the most significant commands should reside on the left or right side so that they go into the overflow last.




## How to configure overflow

If you want to change the default behavior, or configure it based on the use case, you may follow the steps mentioned below.

## If you are using aw-command-bar:

Command bars by default `"wrap"`. To `"overflow"` add attribute `overflow="true"`.

```
<aw-command-bar anchor="aw_ZeroCompileCommandAnchor" alignment="HORIZONTAL"
overflow='true'></aw-command-bar>
```

## If you are using aw-toolbar:

Toolbars `"overflow"` by default. To `"wrap"` add attribute `overflow="false"`.

```
<aw-toolbar first-anchor="leftSlot" second-anchor="rightSlot"
overflow="false"></aw-toolbar>
```

## Chrome and modern browsers

When "overflow" behavior is enabled on a command bar/toolbar, “More Commands” would show up only when commands overflow into it.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### IE Browser behavior

When "overflow" behavior is enabled on a command bar/toolbar, “More Commands” would always shows up. A popup with “No Commands Available” would be displayed when there are no overflowing commands.

### Default behaviors Siemens Web Framework's command bars.

In Common Frame, the Left Navigation Bar has "overflow" behavior, while the right wall has a overlapping tray behavior.

## Application and component-level State

There are two basic types of state in an application:

- *Application-Level State* This applies to the entire application a.k.a global data or singleton.
- *Component-Level State* The scope of the state is bound to a given component instance.

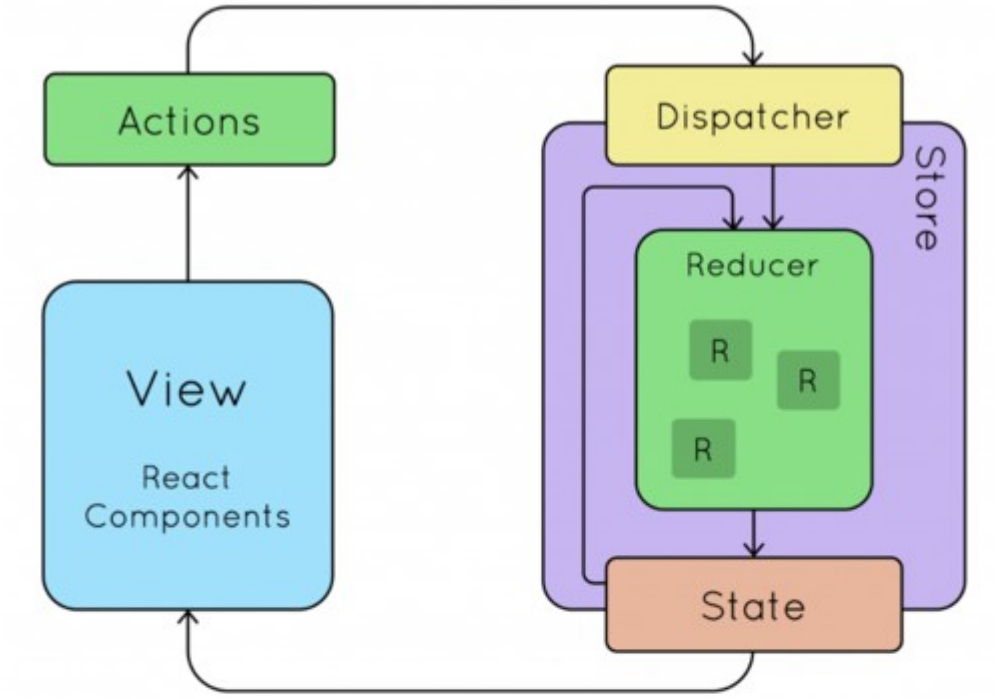
You should always try to minimize the usage of global data and therefore application state. It is an anti-pattern to store component state in the application context thus making it a singleton. You should always assume your component should require being able to be instantiated multiple times on a single page.

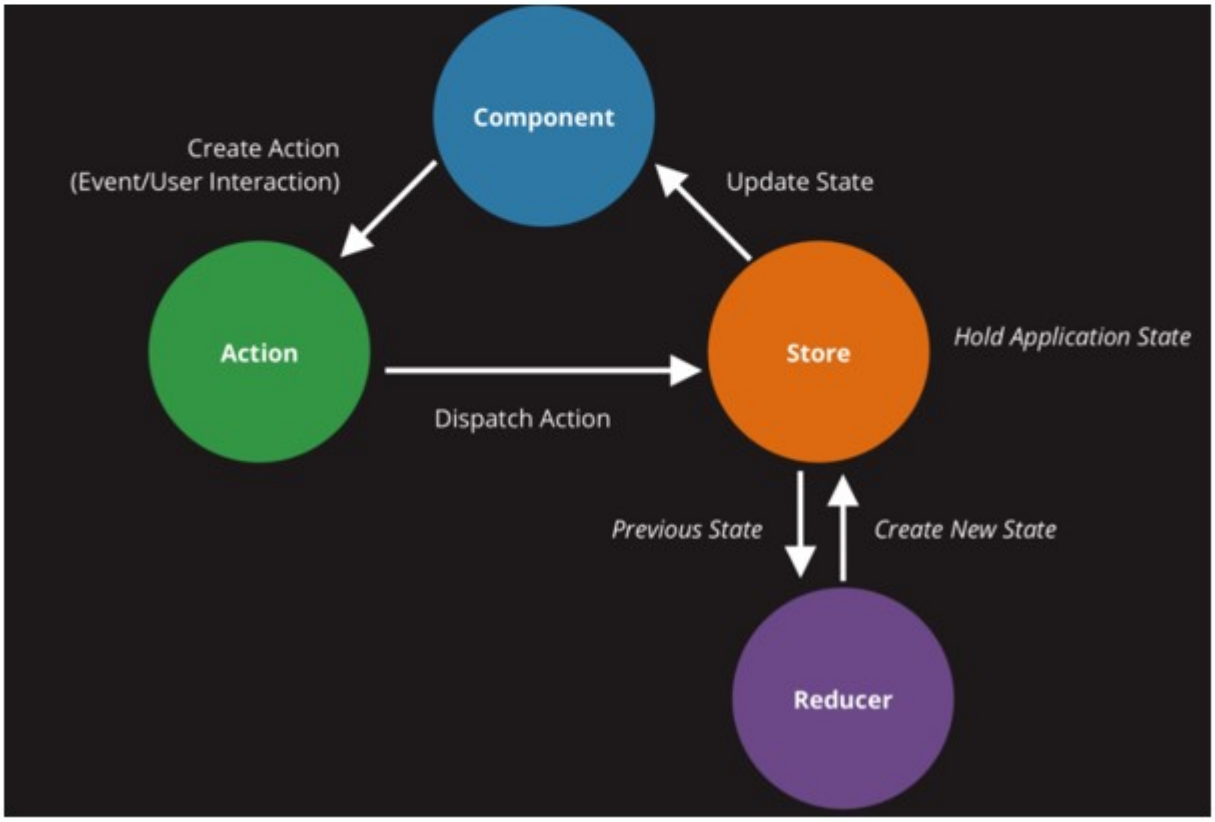
Based on the ideas of the Flux architecture pattern for managing global application-level state:

- the whole global state of your application is stored in an object tree inside a single store
- the only way to change the state tree is to create an action, an object describing what happened, and dispatch it to the store state gets updated by a pure reducer function that calculates a new state based on the old state and the action

This approach has the following advantages:

- It is possible to trace every mutation to the action that caused it. You can record user sessions and reproduce them just by replaying every action.
- The state is always predictable.
- It makes it easy to debug an application. By logging actions and state, it is easy to understand coding errors.
- It is easy to test applications since functions used to change the state are pure functions.





Siemens Web Framework
Unpublished work. © 2025 Siemens

# Application context

Live binding to application values is achieved through application context. Code and **view models** can register, bind data to, update, and unregister application context data.

The application context is simply a singleton JavaScript object that can hold state. It will announce changes performed through its API using the `appCtx.register` and `appCtx.update` event bus events. Also, you can use the `registerPartialCtx` , `updatePartialCtx` ,ethods to register and update some part of a context, and they will publish the `appCtx.register` and `appCtx.update` events respectively.

Application context is data that is made available for multiple **view models** to access in their **conditions**, **actions** *inputData* or *outputData* mapping, etc. It is a representation of the entire **application-level state** available throughout the application.

**Tip**

Some best practices to consider when working with application context:

- Unless you own a given property in the context beware of setting its value for unintended consequences: application context is a singleton.
- Each `ctx.some.path.obj` should be registered first, and then updated.

## When to use application context or events

Application context is useful for situations where multiple view models need to data bind to common information. Since application context is a singleton, it must be used judiciously and have a defined lifecycle, i.e. when it is *registered* and *unregistered*.

Events should be used for communication between two or more view models on a page. This ensures proper data encapsulation.

## Using application context

## Usage in view and view models

Application context can be used in **views** and view models using standard declarative data binding syntax:

```
"{{ctx.somePath.value}}"
```

Values are namespaced by the choice of path name. These should be semantically meaningful and logically constructed to provide appropriate context. For example, values logically relating to *search* that are made available for consumers could be placed within `{{ctx.search}}`, such as `{{ctx.search.searchString}}` and `{{ctx.search.filterString}}`.

Consider the following examples of application context usage within view models:


Unpublished work. © 2025 Siemens

- **Conditions** referring to application context:

```
"conditions": {
    "haveSearchResults": {
        "expression": "ctx.search.totalFound > 0"
    }
}
```

- **Data** referring to application context:

```
"data": {
    "searchString": {
        "displayName": "{{i18n.searchString}}",
        "type": "STRING",
        "isRequired": "false",
        "isEditable": "false",
        "dbValue": "{{ctx.search.searchString}}",
        "dispValue": "{{ctx.search.searchString}}"
    }
}
```

- **Actions** referring to context:

```
"actions": {
    "save": {
        "actionType": "SOAService",
        "serviceName": "Internal-AWS2-2015-10-FullTextSearch",
        "method": "createFullTextSavedSearch",
        "inputData": {
            "inputs": [ {
                    "pinSearch": "{{function:getPinSearchValue}}",
                    "savedSearchName": "{{data.searchName.dbValue}}",
                    "searchString": "{{data.searchString.dbValue}}",
                    "override": false,
                    "receiveNotification": 0,
                    "shareSavedSearch": 0,
                    "searchFilterMap": "{{ctx.search.searchFilterMap}}"
            } ]
        }
    }
}
```

## Usage in render functions

If your component uses a **render function** instead of a view, application context usages will need to be manually configured. Consider the following example view model with a render function:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "render": {
            "method": "awListRenderFunction",
            "deps": "js/AwListService"
        }
    },
    "lifecycleHooks": {
        "render": "render"
    }
}
```

```
export const awListRenderFunction = ( { ctx } ) => {
    return <div>Total found is {ctx.search.totalFound}</div>;
};
```

In this case, an explicit `ctx` entry for `search.totalFound` must be added:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "render": {
            "method": "awListRenderFunction",
            "deps": "js/AwListService"
        }
    },
    "ctx": {
        "search.totalFound": null
    },
    "lifecycleHooks": {
        "render": "render"
    }
}
```

## Declaratively registering application context

Values can be registered and updated in the application context declaratively. This is achieved in the same way we write values into **data** using the `outputData` property of an action:

```
"actions": {
    "myAction": {
        "actionType": "RESTService",
        "method": "GET",
        "inputData": {
            "name" : "something"
        },
        "outputData": {
             "ctx.somePath.someValue" : "newValue"
        }
    }
}
```

## Codeful usage

Values can be registered, updated, and unregistered from the application context codefully using the `js/appCtxService` JavaScript service.

## js/appCtxService APIs

The published JavaScript APIs for mutating the application context values are:

- `registerCtx`
- `registerPartialCtx`
- `updateCtx`
- `updatePartialCtx`.

These APIs are internally implemented as dispatching actions to a **Redux** store. This is why it is critical that you only update or mutate the application context through one of these APIs.

**Tip**

If you need assistance in inspecting or debugging the current application context you can use the **Redux DevTools** browser extension.

When you register interest in a particular path stored in the application context in your view model using the `ctx` object, your component will get re-rendered whenever the value of that path gets modified through one of the published APIs.

Any direct application context assignments in the code must be avoided. If your application code has any direct context assignment, you should:

- use one of the published APIs when mutating context
- group your changes to application context whenever possible, as every invocation of the application context APIs will trigger an explicit re-render which may lead to poor performance
- eventually, aim to not use application context at all by using **props and state** instead

This is an example of invalid direct assignment:

```
appCtxSvc.ctx.taskAssignmentCtx.enableModifyButton = true;
```

This is the corresponding example using the published APIs:

```
const taskAssignmentCtx = appCtx.getCtx( 'taskAssignmentCtx' );
let newTaskAssignmentCtx = { ...taskAssignmentCtx };
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
newTaskAssignmentCtx.enableModifyButton = true;
appCtxSvc.updateCtx( 'taskAssignmentCtx', newTaskAssignmentCtx );
```

### Context usage dependent on props

If your component has a property that determines what application context is being used, you need to use **derived state** to retrieve the necessary data. A simple example of this would be a view such as the following:

```
<aw-column>
  {{ctx[props.keyToUse]}}
</aw-column>
```

### Storing referenced objects in application context

The expectation that an object reference can be stored on different application context objects and updating one of the references is going to update the other is no longer valid.

Consider the following example:

```
let obj1 = {
    boo: {
         someString: 'something'
    }
 };
appCtxSvc.registerCtx( 'obj1', obj1 );

let obj2 = {
    refToBoo: ctx.getCtx( 'obj1.boo' )
};
appCtxSvc.registerCtx( 'obj2', obj2 );

console.log( appCtxSvc.getCtx( 'obj1.boo.someString' ) ); ---> This will
print 'something'
console.log( appCtxSvc.getCtx( 'obj2.refToBoo.someString' ) ); ---> This
will print 'something'

appCtxSvc.updatePartialCtx( 'obj1.boo', { someString: 'modified' } );

console.log( appCtxSvc.getCtx( 'obj1.boo.someString' ) ); ---> This will
print 'modified'
console.log( appCtxSvc.getCtx( 'obj2.refToBoo.someString' ) ); ---> This
will still print 'something'
```

For these patterns, you need to actually de-link the reference of the objects created previously. Since application context is an immutable object now, the references will not be preserved: you will end up creating a copy and hence updating previous references will not reflect in the copy.

# Services

Services are nothing but standard JavaScript modules exposing public methods which are responsible to perform only specific tasks. This makes them individual entities that can be maintained and tested.

You can use Service when:

- you want to create a new **UI Element** by defining a new **Component**. In this case, your Service will contain the methods referenced in the Component ViewModel **Actions**.
- you want to execute a particular JavaScript function via an Action defined in a ViewModel or other configuration file (for example **commandsViewModel.json**).
- you want to execute a particular JavaScript function via a **function** used in a ViewModel.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# 3. Tutorials

**Warning**

This document refers to the set of tutorials that are compatible with the latest version of Siemens Web Framework. Older versions of the tutorials are included within the .zip file of older versions Siemens Web Framework that can be downloaded from **Support Center**. They are available in PDF format within the `tutorials` directory, after unzipping the Siemens Web Framework .zip file of the desired version.

If you are new to Siemens Web Framework, the best way to learn and get started is to follow the step-by-step tutorial provided in the following section, in the following order:

- **Your first Hello World application**
- **Configuring the common frame**
- **URL routing and page layout**
- **List with data providers**
- **Selection summary**
- **Working with commands**
- **Property widgets and validation**
- **Edit in a summary view**
- **Localization**
- **Working with tables**
- **Editing data in a table**
- **Selection summary with table**
- **Local and global selection**
- **Selection master-details pattern**

**Tip**

The source code for all the tutorials can be found in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

**Note**

Older versions of the tutorials and their source code are included in each Siemens Web Framework release ZIP file, that can be downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Getting Started

Follow the **instructions** to install prerequisite software and create a new Siemens Web Framework project, and then start the **Your First Hello World Application** tutorial.

> **Note**
>
> It is recommended that you follow the tutorials in the specified sequence, as later tutorials rely on the code created in the previous ones. If you want to skip some tutorials, copy the code from the tutorial that precedes the one you want to try in your local project directory.

# Your first *Hello World* application

> **Note**
>
> This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

Let's create your first web application using Siemens Web Framework, this shows how to create a new **solution** with a single **module** in it. The war built from this solution is our web application.

## Objective

You will learn how to:

- Create a solution using SWF CLI (Command Line Interface).
- Use the Siemens Web Framework build tooling.

## Prerequisites

See **Prerequisites**.

## Step 1: Creating the Solution

1. Choose the directory you wish to create the application in. Ensure that the directory is empty.
2. Using the terminal of your choice, `cd` into this directory and run:

```
aw quickstart
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

So while running 'aw quickstart' command, you will be asked

```
Do you want to install SWF packages from a local folder?
Yes
No
```

If you are a Consumer outside Siemens network then select 'Yes' and follow below steps (NOTE: Consumers in Siemens network select 'No' and skip these steps) :

Assuming that:

- The Customer have access to Support Center
- The Customer has downloaded a particular version of SWF from Support Center, e.g. SWF 7.1.0
- The Customer has unzipped the compressed archive containing SWF packages and other files
- The Customer has a folder on their disk called "packages" containing all public packages necessary to install and build an SWF project

```
Specify the path to the folder containing .tgz SWF packages ../packages
```

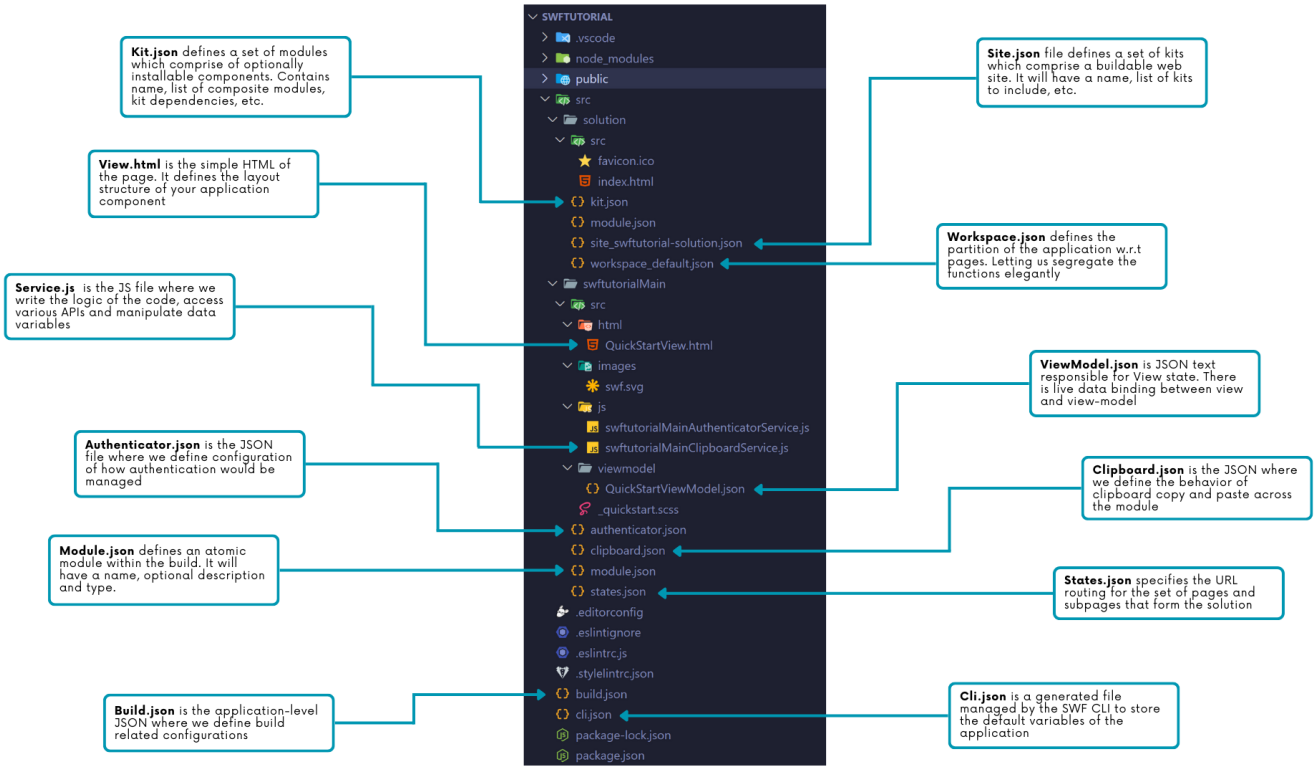
Provide the path of packages folder, command will use the tgz packages present in the specified folder instead of attempting to downloading them from Artifactory.

```
This command would install all the necessary packages, generate a starter
code and start the development server.
```

1. Launch Visual Studio Code with a different instance of the server.

```
code .
```

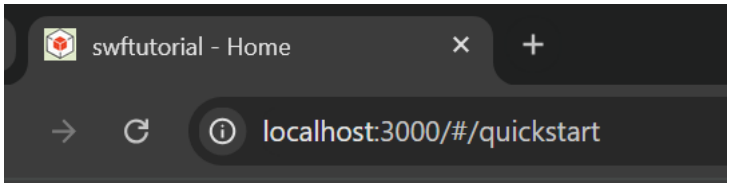

Unpublished work. © 2025 Siemens




## Step 2: Branding (skip this step in @swf/core)

1. Add favicon to your application

   A favicon (short for favorite icon), also known as a shortcut icon, website icon, tab icon, URL icon, or bookmark icon, is a file containing small icon associated with a particular website or web page.




   a. Create your own *favicon.ico* file.

   b. Replace this file with favicon.ico under src => solution => src folder as a peer to *index.html*.

2. Add browser title

   Update browserTitle key inside src/solution/kit.json under solutionDef

```
{
    "browserTitle": "tutorial"
}
```

3. Restart the Development Server


Siemens Web Framework
Unpublished work. © 2025 Siemens

Restart the dev-server using the command:

```
aw start
```

**Note**

If you are not able to see the favicon, then clear the browser cache and retry. If that doesn't work either, close the server, run `aw clean` and restart the dev server.

## Step 3: Build the Solution

A build is a deployable unit created for the SWF application. SWF builds are site specific. Notice that our code has only 1 site (defined at `site_swftutorial-solution.json`). The name of that SITE is "swftutorial-solution". If there are multiple sites available, we must first set the site as:

```
set SITE=\<site_name\>
```

Then proceed to build the application.

1. Run the following **build command** to build the solution.

   ```
   aw build
   ```

   This command creates the `site` to deploy for the solution defined by *site_\<swftutorial-solution>\.json* definition inside your solution module. Resulting `site` is placed in `/out/site/swftutorial-solution`.

2. Using any local server package, deploy this site to localserver. If you use the 'serve' package, run the following command:

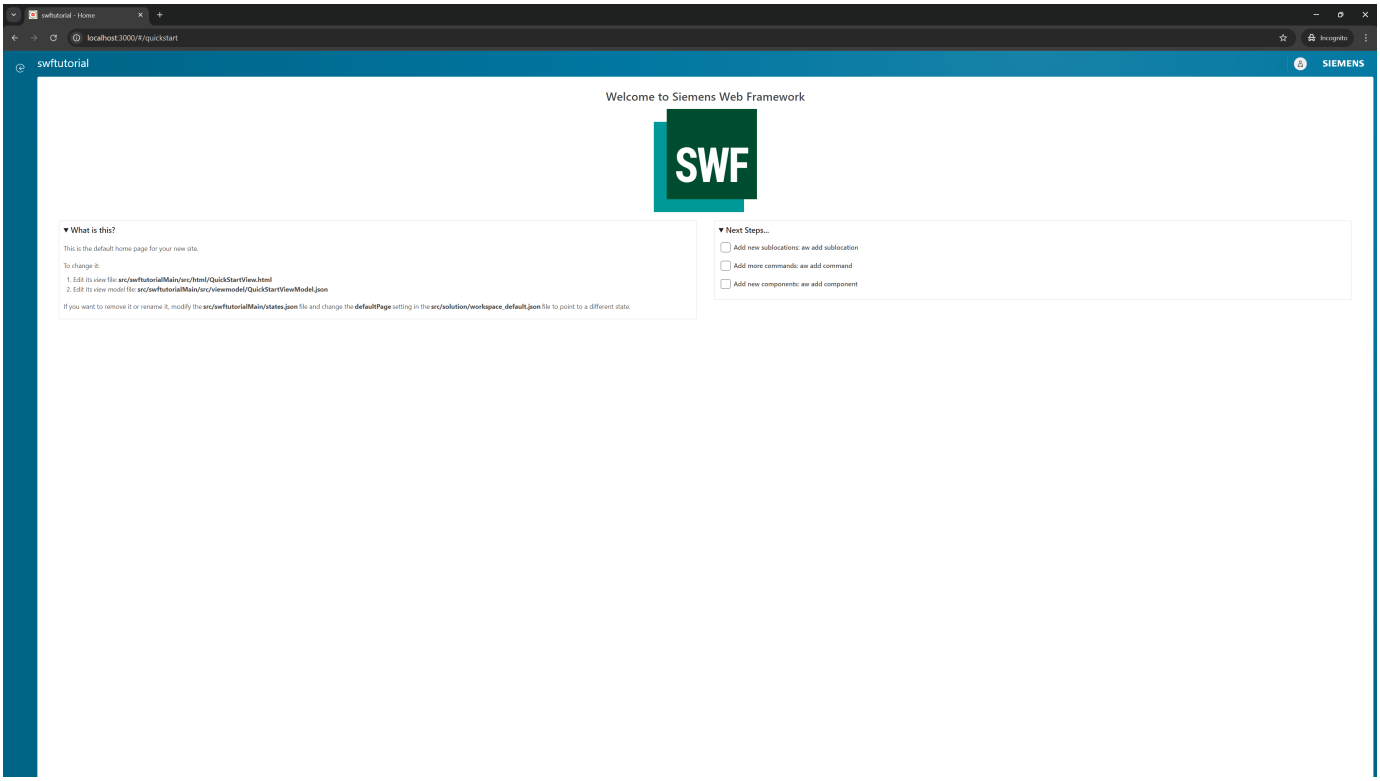
   ```
   serve .\out\site\swftutorial-solution
   ```

   This command deploys the `site` to local server for testing. You can then open a browser to exercise your solution at localhost.

## Summary

In this tutorial, you learned how to create a Solution using the SWF CLI.


Unpublished work. © 2025 Siemens




## References

- **Solution**
- **Module**
- **Tools**

# Configuring the common frame

> **Note**
>
> This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

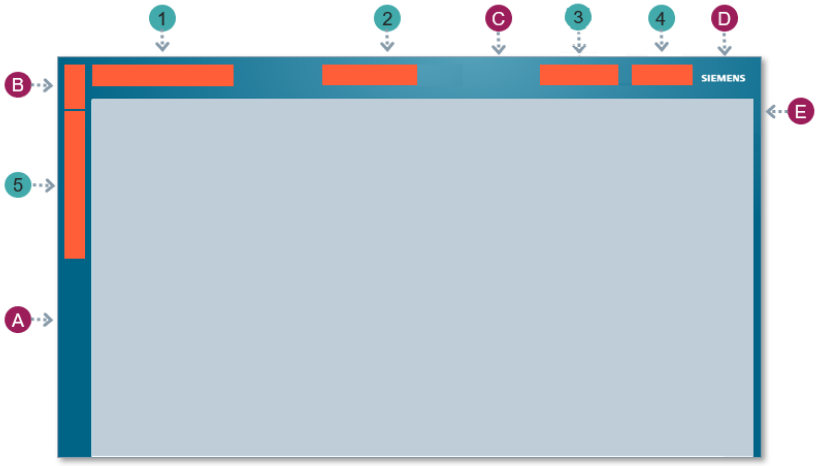
This tutorial shows how to add the `Common Application Frame` to an existing solution.

Common Application Frame consists of header and side navigation and it provides single, recognizable web presence across all PL Web offerings. It is recommended to all the applications developed using Siemens Web Framework to have consistent layout.


Unpublished work. © 2025 Siemens

| | |
|---|---|
| A | Left Navigation area, fixed width, constant blue background*** |
| B | Global Navigation controls and app logo, common to all uses |
| C | Header area, gradient blue background, variable height from minimum value |
| D | Siemens Logo area, fixed, common to all |
| E | Right hand bar, fixed width, smaller than left (WIP) |

| | |
|---|---|
| 1 | Header left custom area* |
| 2 | Header centre custom area* |
| 3 | Header right custom area* |
| 4 | Main Navigation profile and settings controls |
| 5 | Main Navigation custom area** |




## Objective

Learn how to add an outer frame to your application.

| Frame Type | Description | Internal /External Usage | Note |
|---|---|---|---|
| Common Frame with Slot Layout | Framework provides approved version of outer frame, also called common frame at framework level. This ensures any application using common frame is guaranteed to look consistent. Additionally there are slots provided to configure contents inside the common frame based on application needs. | Consumers inside and outside Siemens network may use it. | Any future changes to common frame shall get reflected to your application automatically. It's mandatory for applications built internally to use common frame. |

This is the basic frame which is sufficient to continue with Siemens Web Framework tutorials.

**Note**

The alternative mechanism previously available to generate common frame known as 'Common Frame using the Generator' is no longer supported starting Siemens web framework v5.0 or higher.

## Pre-requisites

**Recommended:**

Complete Your First Hello World Application before proceeding.

**Otherwise:**

Download the source code for the **hello_world_application** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Use Common Frame with Slot Layout

**Use the common frame in state definition**

1. Define an abstract state called `commonRoot` with view as `commonRoot`, this will be root view for our application.
2. Define new abstract state called `helloFrame` with `commonRoot` as it's parent
3. Set `commonLocation` as view for `helloFrame` state
4. Update default state `hello` and set `helloFrame` as it's parent

Your `states.json` should look like this:

```
{
    "commonRoot": {
        "abstract": true,
        "view": "commonRoot"
    },
    "helloFrame": {
        "abstract": true,
        "data": {
            "headerTitle": "!!! Hello World !!!",
            "browserSubTitle": "Common Frame"
        },
        "view": "commonLocation",
        "parent": "commonRoot"
    },
    "hello": {
        "url": "/hello",
        "data": {},
        "view": "hello",
        "parent": "helloFrame"
    }
}
```

The attributes *headerTitle* and *browserSubTitle* are optional and may be omitted but when they are mentioned, Common Frame uses them.

Add view/viewModel for 'hello' component.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Content of `helloView.html`. Add this file under html folder in your module.

```
<aw-flex-row height="fill" offset="3">
    <aw-flex-column width="fill" offset="4">
       <aw-label prop="data.helloText"></aw-label>
    </aw-flex-column>
</aw-flex-row>
```

Content of `helloViewModel.json`. Add this file under viewmodel folder in your module.

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "helloText": {
            "type": "STRING",
            "uiValue": "{{i18n.helloText}}"
        }
    },
    "i18n": {
        "helloText": [
            "helloModuleMessages"
        ]
    }
}
```

Content of `helloModuleMessages.json`. Create "i18n" folder under "src" folder and keep this file over there.

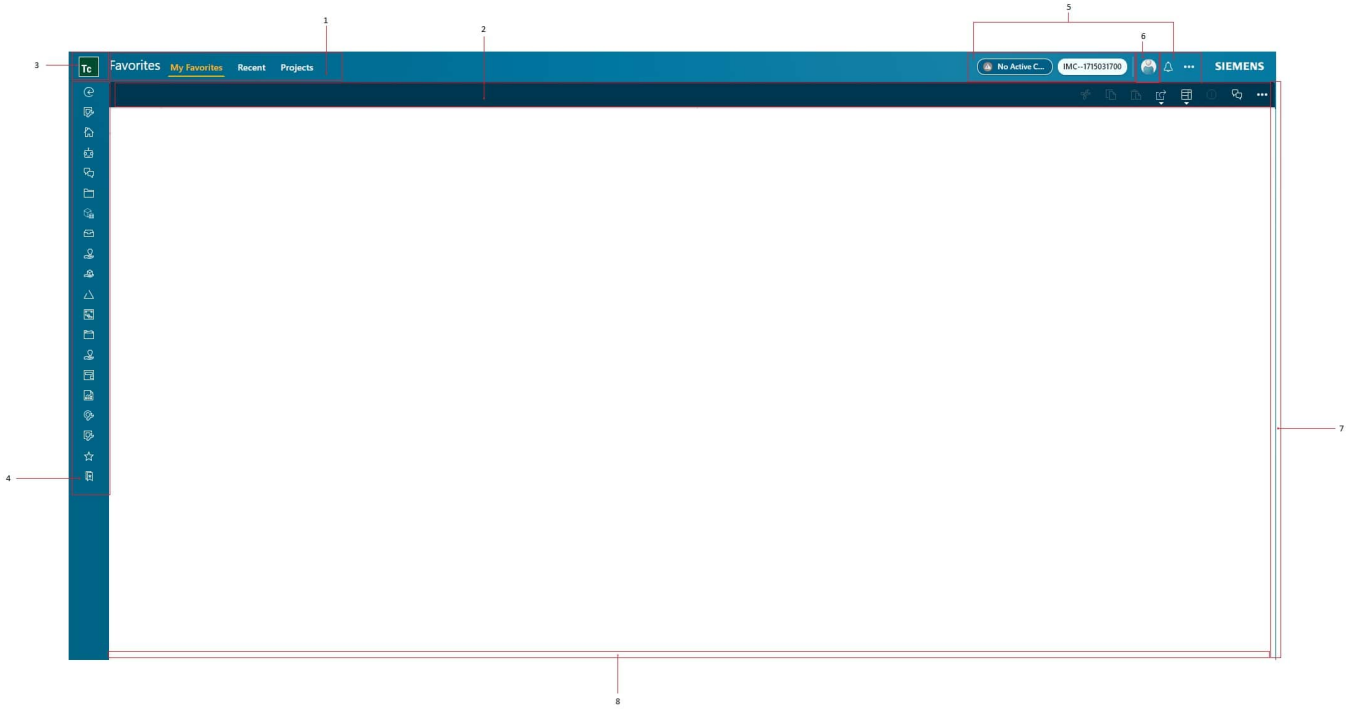
```
{
    "helloText": "!! Hello World !!"
}
```

In case user is not willing to change the UI provided by the framework, Common Frame already has its own default view/viewmodel. If you don't want custom views for frame, You can skip **Override a slot** step and directly jump to **Build the Solution** step.

**Override a slot**

This Common Frame is providing various slots as shown in the image and consumer can override using respective slot name.


Unpublished work. © 2025 Siemens




1. **Header Slot**

   *aw_headerTitleWithTabs* - This slot needs to override if the consumer is using the frame with Header Tabs. *aw_headerTitle* - This slot needs to override if the consumer is using the frame without Header Tabs (Where tabs are visible below taskbar).

2. **Taskbar Slot**

   *aw_taskBar* - Consumer can override this Slot to create their own taskbar

3. **App Logo Slot**

   *aw_appLogo* - Consumer can override this Slot to show/hide their app logo. Default app logo slot is not visible in the application.

4. **Global Navigation Bar Slot**

   *aw_globalNavigationBar* - Consumer can contribute the commands to this Slot using *aw_globalNavigationbar* anchor. Consumer can also override this slot and can create their own view instead of just contributing the commands.

5. **User Session Bar Slot**

   *aw_userSessionBar* - Consumer can contribute the session specific commands to this Slot using *aw_userSessionbar* anchor. Consumer can also override this slot and can create their own view instead of just contributing the commands.

6. **Avatar Slot**

   *aw_avatar* - Basic avatar with side-nav is available on click of avatar.

7. **Right Edge Slot**


Siemens Web Framework
Unpublished work. © 2025 Siemens

*aw_rightEdge* - Right Edge Empty Slot is made available with this frame. Consumer can render their own view/viewmodel on right edge of application.

8. **Footer Slot**

   *aw_footer* - Empty Footer Slot is also made available with this frame where consumer can render their own view/viewmodel.

To override any of the slots, user need to configure their own view and viewmodel through *layoutSlots.json*

**Steps to override the slot**

1. Create *layoutSlots.json* in *swfTutorialMain* directory, parallel to *module.json* and configure it to override the slot

```
{
    "slots": {
        "tutorialHeaderSlot": {
            "name": "aw_headerTitle",
            "view": "tutorialHeaderSlot",
            "activeWhen": {
                "condition": "conditions.tutorialHeaderSlot"
            }
        }
    },
    "conditions": {
        "tutorialHeaderSlot": {

"expression":"ctx.locationContext['ActiveWorkspace:Location'] ===
'helloFrame'"
        }
    }
}
```

2. Create respective *tutorialHeaderSlotView.html* and *tutorialHeaderSlotViewModel.json* files and put in html and viewmodel folders in src directory of *swfTutorialMain*.

3. Add the following code snippets for files: *tutorialHeaderSlotView.html*

```
<!-- Left (HEADER) Area -->
<aw-flex-column class="awHeader">
    <aw-flex-row align-content="center" class="aw-layout-
headerTitleContainer">
        <aw-header-title headerVMO="subPanelContext.headerViewModel"
headerTitle="subPanelContext.headerTitle"
headerProperties="subPanelContext.headerProperties"></aw-header-title>
    </aw-flex-row>
</aw-flex-column>
```

*tutorialHeaderSlotViewModel.json*

```
{
    "schemaVersion": "1.0.0",
    "data": {},
    "actions": {},
    "dataProviders": {},
    "functions": {},
    "onEvent": [],
    "i18n": {},
    "messages": {},
    "conditions": {}
}
```

## Common Steps for all the Frames: Build the Solution

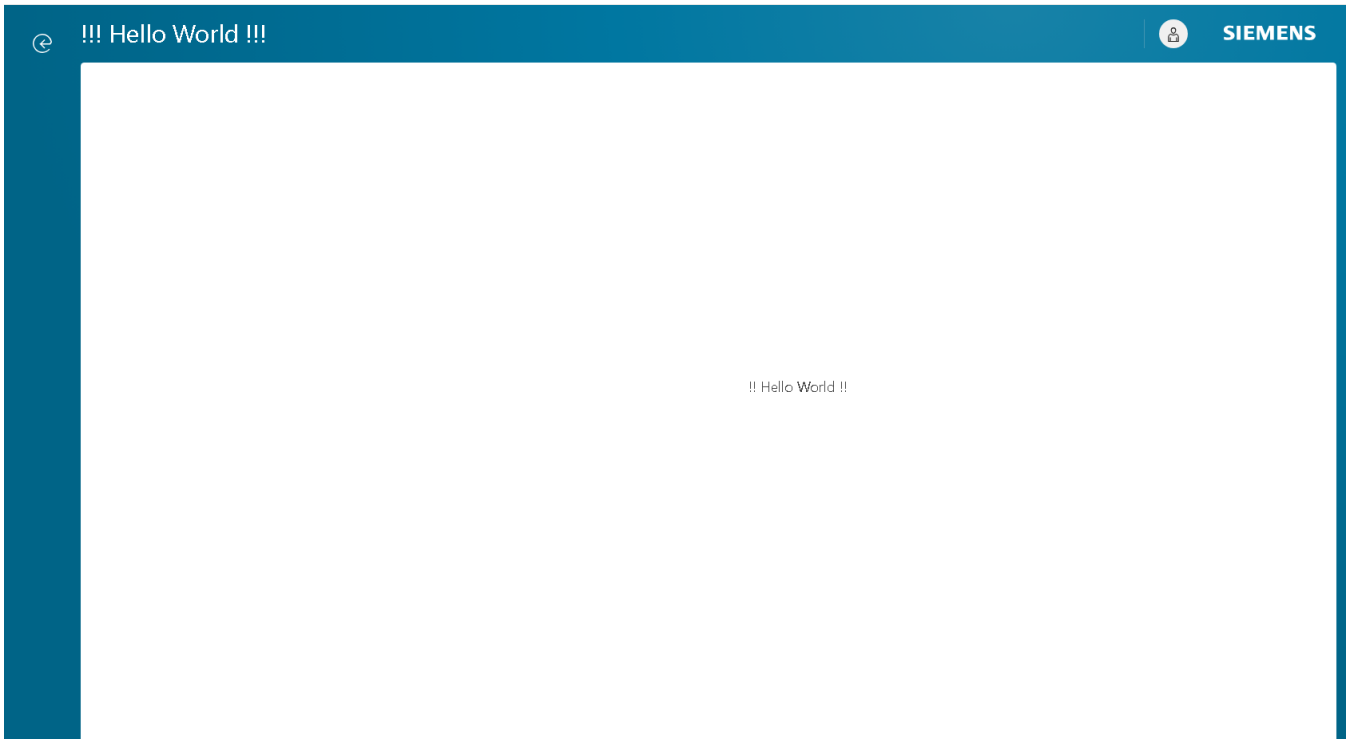
Run the build commands to build and deploy the site.

```
aw clean && aw start
```

## Summary

In this tutorial, you learned how to:

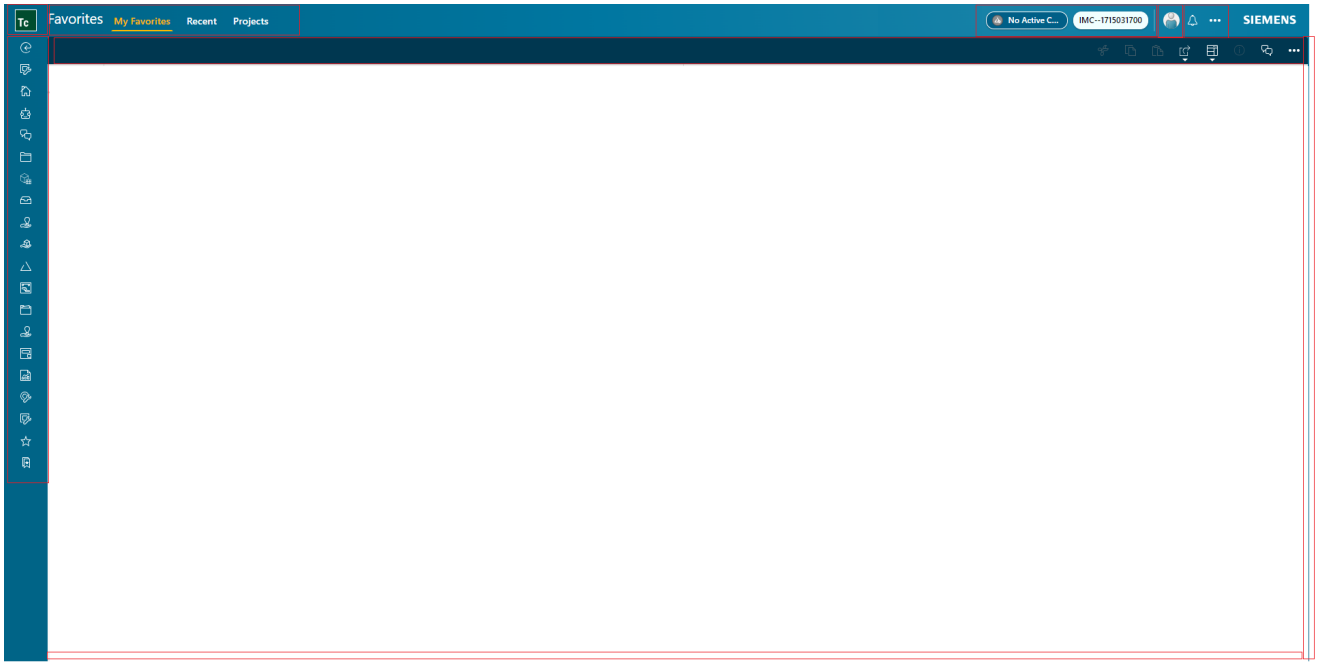
- How to create a Common Frame.





Siemens Web Framework
Unpublished work. © 2025 Siemens

- How to use Common Frame with Slot Layout.




# URL routing and page layout

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

Let's add a new page to our web application using URL routing and add suitable layout elements to build the page.

## Objective

You will learn how to:

- How to add new pages to your `Single Page Application` created using Siemens Web Framework
- How URL routing works?
- Use Location/Sublocation template to render your application UI

## Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **Consuming Common Frame** before proceeding.

**Otherwise:**

Download the source code for the **consuming_common_app_frame** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Add new pages

Update `states.json` file in the `swfTutorialMain` folder generated from last tutorial. The states.json file **defines the states** for this module. The Siemens Web Framework build tool collects all the state definitions for the **solution** from each of its constituent modules when building the site.

The content of this file contains the `state definition` for a single state/page. It configures the *URL* fragment, and the **declarative** *view* for this particular page (or subpage).

1. Define `Location` - Movies

   Define a new state `movies` inside `states.json`. Set its parent as commonLocation (so that it uses common application frame)

```
{
    "movies": {
        "abstract": true,
        "data": {
            "headerTitle": "!!! Movies !!!"
        },
        "view": "commonLocation",
        "parent": "commonRoot"
    }
}
```

2. Define `Sublocation` - Favorites

   Define a new state `myFavs` with its parent as "movies" and view as "AwStandardSublocation"

```
{
    "myFavs": {
        "url": "/myFavs",
        "data": {
            "label": "Favorites",
            "nameToken":
"com.siemens.splm.client.favorites:favoritesSubLocation",
            "clientScopeURI": "",
            "viewBase": "favorites"
        },
        "parent": "movies",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "view": "AwStandardSublocation"
    }
}
```

:pushpin: Make note of `view` attribute and its value `AwStandardSublocation`. Also notice the `viewBase` attribute

3. Define another `Sublocation` - Recent

   Define a new state `myRecent` with its parent as "movies" and view as "AwStandardSublocation"

```
{
   "myRecent": {
        "url": "/myRecent",
        "data": {
            "label": "Recent",
            "nameToken":
"com.siemens.splm.client.favorites:myRecentSubLocation",
            "clientScopeURI": "",
            "viewBase": "favorites"
        },
        "parent": "movies",
        "view": "AwStandardSublocation"
    }
}
```

:pushpin: Make note of `view` attribute and its value `AwStandardSublocation`

## Step 2: Set Movies page as the default launch page

Update `defaultPage` entry inside `workspace_default.json` to point to myFavs instead of hello

```
{
    "defaultPage": "myFavs"
}
```

`Warning`: This requires you to perform clean build (i.e. aw clean)

```
aw clean
```

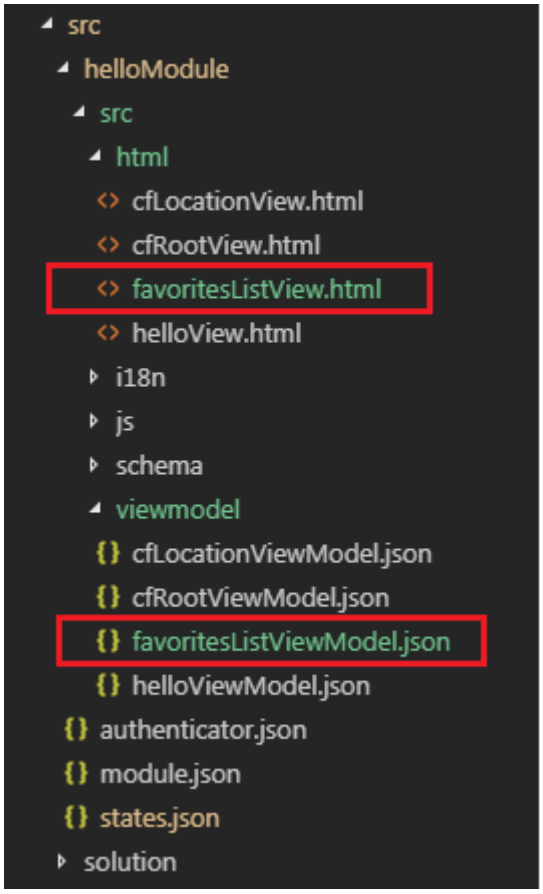
## Step 3: Configure view for myFavs and myRecent pages (i.e. subLocations)

Note the `viewBase` entry in states.json for `myFavs` and `myRecent` states

```
{
    "myFavs": {
        "data": {
```

```
            "viewBase": "favorites"
        }
    }
}
```

Mentioning the `viewBase` provides ways to configure ready made view modes, like, *Lists*, *Table*, etc. The value in `viewBase` is used to configure the views. For now lets add two files named `favoritesListView.html` and `favoritesListViewModel.json` and put in `html` and `viewmodel` folders in `src` directory of `swfTutorialMain`.




Update `favoritesListView.html` with following code snippet:

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


Siemens Web Framework
Unpublished work. © 2025 Siemens

Update `favoritesListViewModel.json` with following code snippet:

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "data": {
        "label": {
            "displayName": "Label",
            "type": "STRING",
            "dispValue": "List View"
        }
    }
}
```

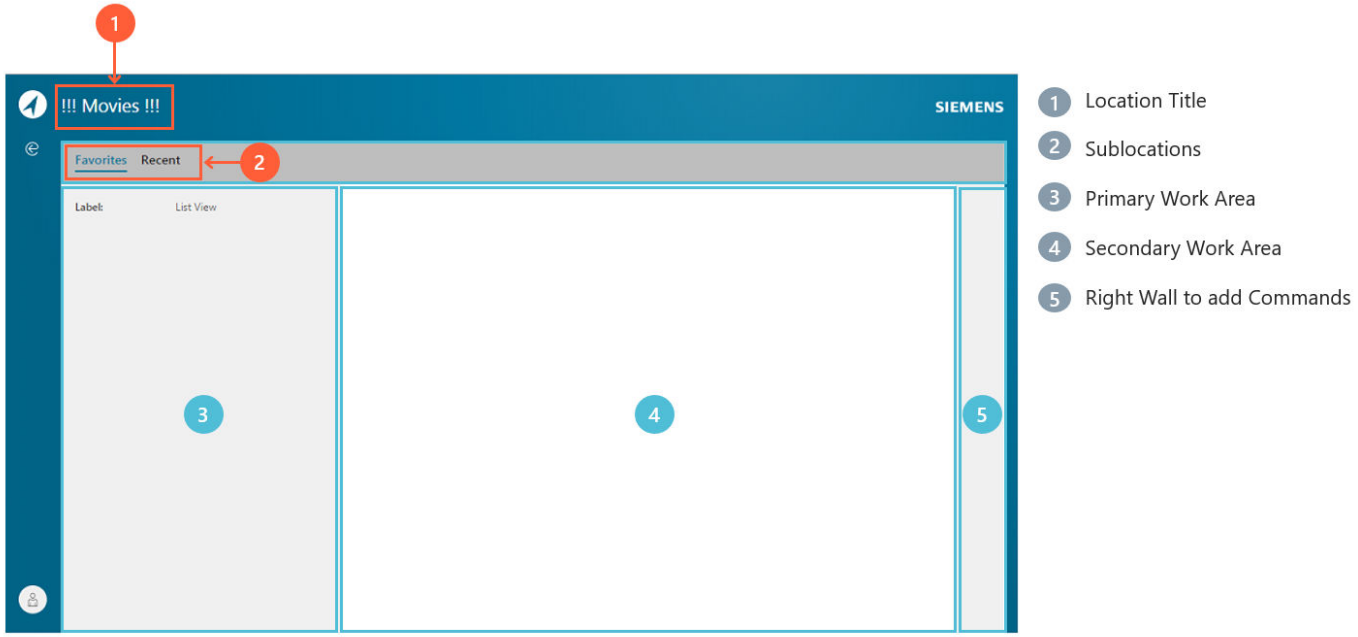
## Step 4: Build the Solution and test various pages

Run the following build commands to build and deploy the site.

```
aw clean && aw start
```

## Summary

In this tutorial, you learned how to add a new page to an existing application and design the page with various layout and container elements.



### References

- Solutions
- States
- Declarative UI
- View
- View model
- Audit

# Lists with data provider and selection

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from Support Center, as explained in the Tutorials Overview.

## Objective

You will learn how to configure a data provider to display data in a list in Primary Work Area.

## Pre-requisites

You must have already gone through pre-requisite software by now.

**Recommended:**

Complete URL Routing and Page Layout before proceeding.

**Otherwise:**

Download the source code for the **url_routing_and_page_layout** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see Pre-requisite software for information on how to download Siemens Web Framework).

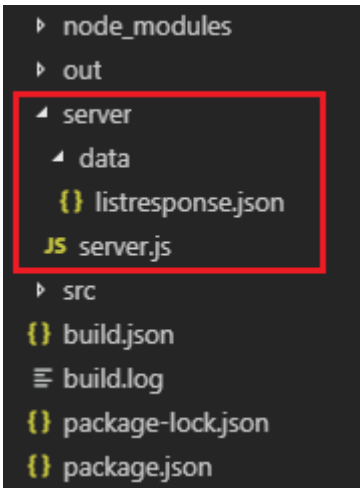

Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 1: Setup the backend server that serves data to the Movie web application.

Download the source code for the **list_with_data_provider** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

> **Tip**
> Note that this folder contains the source code of the *completed* tutorial, so unzip it outside your current project folder if you are planning to follow along with the rest of the tutorial.

Inside the **list_with_data_provider** folder, copy the **server** folder to the root folder of your project.




## Step 2: Update package.json to add the `startServer` command to start the node server to serve the data.

In the `scripts` section of package.json, update the "start" command as :

```
{
    "scripts": {
        "startServer": "node server/server.js"
    }
}
```

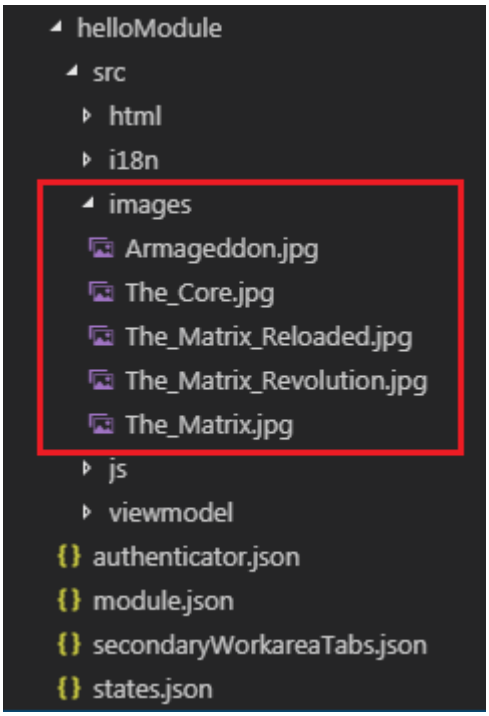
Start the server by running below command in command prompt.

```
npm run startServer
```

## Step 3: Add folder to store images

Create a new **images** folder in the **src** directory under **helloModule**.

From the downloaded tutorial code (see Step 1), copy the images included in **list_with_data_provider/src/helloModule/src/images** into your images folder.




## Step 4: Use `aw-list` UI element to display the list of movies in View.

Update `favoritesListView.html` with the following code snippet:

```
<aw-panel>
    <aw-list dataprovider="data.dataProviders.performSearch">
        <aw-default-cell vmo="item"></aw-default-cell>
    </aw-list>
</aw-panel>
```

`aw-list` can be configured with a **data provider** which is responsible to get the data. Let us display each movie in the list referred to as `item` using `aw-default-cell`.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 5: Configure data provider in View Model for the list.

Add `dataProviders` section for configuring the data provider for list which refers to `action` - `getMovies`.

```
{
    "dataProviders": {
        "performSearch": {
            "action": "getMovies",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            }
        }
    }
}
```

Add **lifecycle hooks** section to initialize the list on mount of the component.

```
{
    "lifecycleHooks": {
        "onMount": "initializeList"
    }
}
```

Add **actions**section in the view model. `initializeList` action is responsible to initialize the list and `getMovies` actions uses a data parser `movieParser` which maps the rest call out to the right set of properties in our view model.

```
{
    "actions": {
        "initializeList": {
            "actionType": "dataProvider",
            "method": "performSearch"
        },
        "getMovies": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/movies",
                    "params": {
                        "startIndex":
"{{data.dataProviders.performSearch.startIndex}}",
                        "pageSize": "15"
                    }
```


Unpublished work. © 2025 Siemens

```
                }
            },
            "dataParsers": [{
                "id": "movieParser",
                "responseObjs": "data.search"
            }],
            "outputData": {
                "totalFound": "data.totalResults",
                "searchResults": "data.search"
            }
        }
    }
}
```

Add the **data parser** definition for `movieParser`.

```
{
    "dataParseDefinitions": {
        "movieParser": {
            "identifier": "{{response.id}}",
            "cellHeader1": "{{response.Title}}",
            "typeIconURL": "{{response.Poster}}",
            "displayName": "{{response.Title}}",
            "cellProperties": {
                "Year": {
                    "key": "Year",
                    "displayName": "{{i18n.movieYear}}",
                    "value": "{{response.Year}}",
                    "displayValue": "{{response.Year}}",
                    "type": "String"
                },
                "Type": {
                    "key": "Type",
                    "displayName": "{{i18n.movieType}}",
                    "value": "{{response.Type}}",
                    "displayValue": "{{response.Type}}",
                    "type": "String"
                },
                "Starring": {
                    "key": "Starring",
                    "displayName": "{{i18n.movieCast}}",
                    "value": "{{response.Starring}}",
                    "displayValue": "{{response.Starring}}",
                    "type": "String"
                }
            },
            "props": {
                "object_string":{
                    "displayName": "{{response.Title}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                }
            }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        }
    }
}
```

Add the **i18n** section as the last section in view model to support localization.

```
{
    "i18n": {
        "movieTitle": ["helloModuleMessages"],
        "movieType": ["helloModuleMessages"],
        "movieFormat": ["helloModuleMessages"],
        "Description": ["helloModuleMessages"],
        "movieYear": ["helloModuleMessages"],
        "movieCast": ["helloModuleMessages"]
    }
}
```

Add "proxy" : "**http://localhost:8181**" in package.json file

```
{
    "proxy": "http://localhost:8181"
}
```

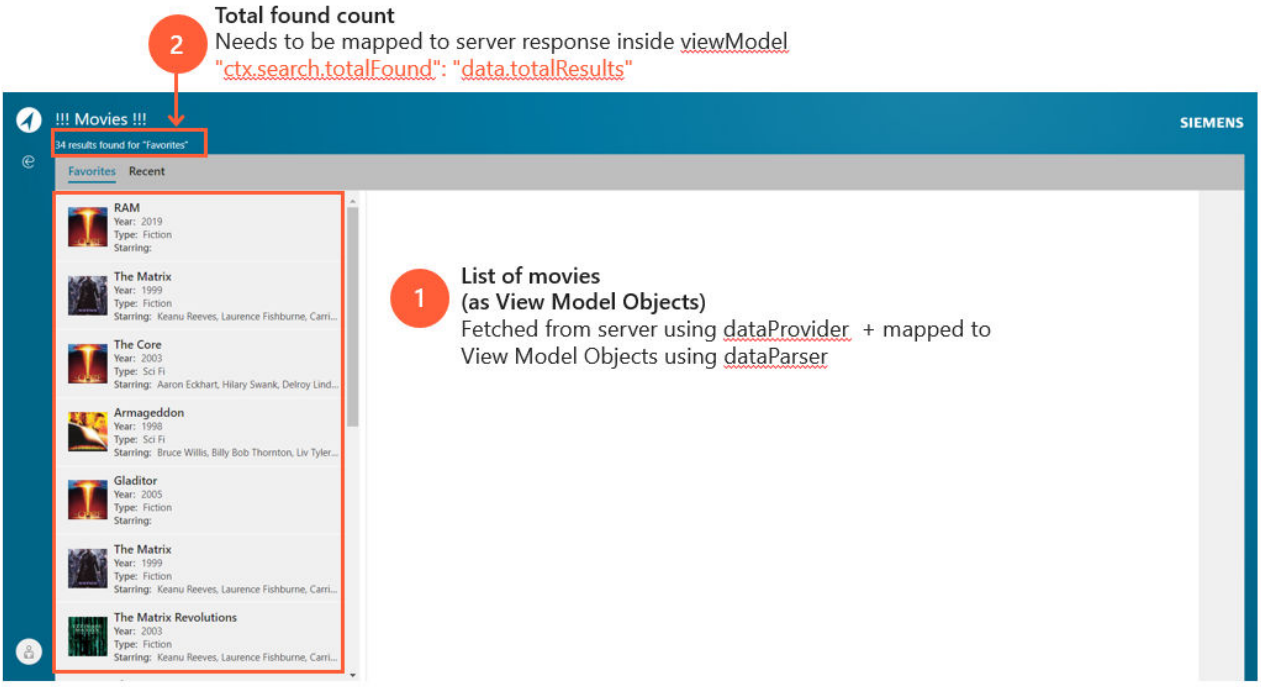
Let's build your solution and see how far we have reached.

```
aw clean && aw start
```


Unpublished work. © 2025 Siemens

**Note**

Search BreadCrumb (Circle 2) currently not supported.




## Summary

In this tutorial, you learned how to:

- Use a list to display data
- Configure data provider in view model

## References

- **Data provider**
- **Data parser**
- **Actions**
- **Localization**


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Selection summary

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

## Objective

You will learn how to configure selection summary to display in Secondary Work Area.

## Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **List with Dataprovider** before proceeding.

**Otherwise:**

Download the source code for the **list_with_dataprovider** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Configure selection in list and display a secondary work area on selection.

Add a file named `secondaryWorkareaTabs.json` in `swfTutorialMain` directory, parallel to module.json file. This configuration is used to configure a tab `Overview` for secondary work area and it's corresponding view `favSelectionDetails`.

```
{
    "schema": "1.0.0.0",
    "tabs": {
        "favSelectionDetails": {
            "name": "Overview",
            "priority": 50,
            "view": "favSelectionDetails",
            "visibleWhen": "selection.length === 1"
        }
    }
}
```


Unpublished work. © 2025 Siemens

Now lets add two files named `favSelectionDetailsView.html` and `favSelectionDetailsViewModel.json` and put in `html` and `viewmodel` folders in `src` directory of `swfTutorialMain`.

Add the following code snippets for each file:

`favSelectionDetailsView.html` - Using property widgets to display movie data like Title, Description, Year of Release, Id, etc. These are bound with properties defined in `favSelectionDetailsViewModel.json`.

```
<aw-flex-column width="12">
    <aw-flex-row height="fill">
        <aw-flex-column width="4">
            <aw-widget prop="data.selectedObj.props.Title"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Id"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Description"></aw-
widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Type"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Year"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Starring"></aw-widget>
        </aw-flex-column>
        <aw-flex-column width="fill" justify="center">
            <aw-image source="data.selectedObj.props.Poster.dbValue"></aw-
image>
        </aw-flex-column>
    </aw-flex-row>
</aw-flex-column>
```

`favSelectionDetailsViewModel.json` - On selecting an object in list, action "getSelectedObject" is performed.

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "actions": {
        "getSelectedObject": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/movies/
{{props.subPanelContext.selection[0].uid}}"
                }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            },
            "dataParsers": [{
                "id": "selectedMovieParser",
                "responseObjs": "data.search"
            }],
            "outputData": {
                "selectedObj": "data.search"
            }
        }
    }
}
```

Action "getSelectedObject" uses data parser "selectedMovieParser" which has the definition for each property, configured in view, bound with a property in view model.

```
{
    "dataParseDefinitions": {
        "selectedMovieParser": {
            "identifier": "{{response.id.value}}",
            "props": {
                "Title": {
                    "name": "Title",
                    "displayName": "Title",
                    "value": "{{response.Title.value}}",
                    "displayValue": "{{response.Title.value}}",
                    "isEditable": "{{response.Title.isEditable}}",
                    "type": "String"
                },
                "Description": {
                    "name": "Description",
                    "displayName": "Description",
                    "value": "{{response.Description.value}}",
                    "displayValue": "{{response.Description.value}}",
                    "isEditable": "{{response.Description.isEditable}}",
                    "type": "String"
                },
                "Year": {
                    "name": "Year",
                    "displayName": "Year Of Release",
                    "value": "{{response.Year.value}}",
                    "displayValue": "{{response.Year.value}}",
                    "isEditable": "{{response.Year.isEditable}}",
                    "type": "String"
                },
                "Type": {
                    "name": "Type",
                    "displayName": "Movie Type",
                    "value": "{{response.Type.value}}",
                    "displayValue": "{{response.Type.value}}",
                    "isEditable": "{{response.Type.isEditable}}",
                    "type": "String"
                },
                "Id": {
                    "name": "Id",
```

```
                    "displayName": "Movie ID",
                    "value": "{{response.id.value}}",
                    "displayValue": "{{response.id.value}}",
                    "isEditable": "{{response.id.isEditable}}",
                    "type": "String"
                },
                "Starring": {
                    "name": "Starring",
                    "displayName": "Starring",
                    "value": "{{response.Starring.value}}",
                    "displayValue": "{{response.Starring.value}}",
                    "isEditable": "{{response.Starring.isEditable}}",
                    "type": "String"
                },
                "Poster": {
                    "name": "Poster",
                    "dbValue": "{{response.Poster.value}}",
                    "value": "{{response.Poster.value}}",
                    "isEditable": "{{response.Poster.isEditable}}",
                    "type": "String"
                }
            }
        }
    }
}
```

Whenever the selection received in props is changed, the action "getSelectedObject" needs to be invoked.

```
{
    "lifecycleHooks": {
        "onUpdate": [ {
            "action": "getSelectedObject",
            "observers": [ "props.subPanelContext.selection.uid",
"props.subPanelContext.selection[0].uid",
"props.subPanelContext.selection.length" ]
        }]
    }
}
```

## Step 2: Add an empty view for the cases where nothing is selected.(optional)

Update `secondaryWorkareaTabs.json` in `swfTutorialMain` directory, parallel to module.json file; to configure a default empty view when nothing is selected in the Primary workarea.

```
{
    "schema": "1.0.0.0",
    "tabs": {
        "noSkeleton": {
            "name": "",
            "priority": 40,
            "view": "AwSecondaryWorkareaNoSkeleton",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "visibleWhen": "selection.length === 0"
        }
    }
}
```

Now lets add two files named `AwSecondaryWorkareaNoSkeletonView.html` and `AwSecondaryWorkareaNoSkeletonViewModel.json` and put in `html` and `viewmodel` folders in `src` directory of `swfTutorialMain`.

Add the following code snippets for each file:

`AwSecondaryWorkareaNoSkeletonView.html` -

```
<div></div>
```

`AwSecondaryWorkareaNoSkeletonViewModel.json` -

```
{
    "schemaVersion": "1.0.0"
}
```

## Step 3: Update URL with selected object.(optional)

Update sublocation state definition to pass selected uid to update URL with selected object. `states.json` file under `swfTutorialMain` folder add following code snippet in `myFavs` section as shown below.

```
"params": {
    "s_uid": null
}
```


Unpublished work. © 2025 Siemens

```
"myFavs": {
    "url": "/myFavs",
    "data": {
        "label": "Favorites",
        "nameToken": "com.siemens.splm.client.movies:favoritesSu
        "clientScopeURI": "",
        "viewBase": "favorites"
    },
    "params": {
        "s_uid": null
    },
    "parent": "movies",
    "type": "subLocation"
```

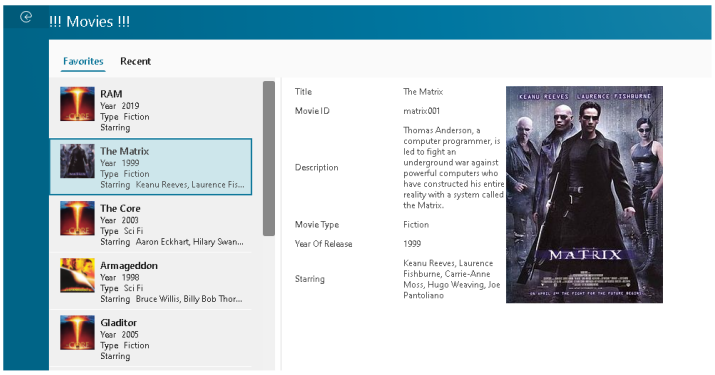
## Step 4: Build and Test.

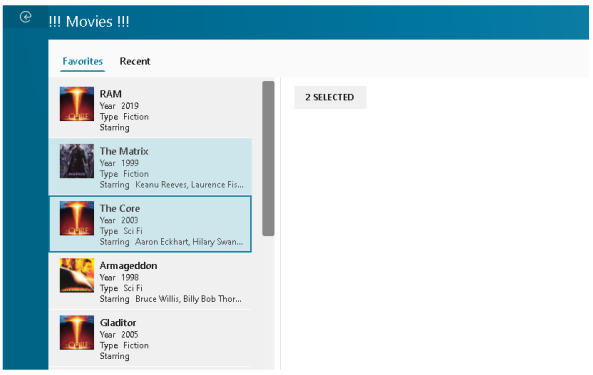
1. Run the following build commands to build and deploy the site.

```
aw clean && aw start
```

2. Launch your application at following port (Port number is defined in server.js ).

```
localhost:8181
```




## Summary

In this tutorial, you learned how to provide selections to each item in a list and display contents in secondary work area


Siemens Web Framework
Unpublished work. © 2025 Siemens

## References

- Data provider
- Data parser
- Actions
- Localization

# Working with commands

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from Support Center, as explained in the Tutorials Overview.

This tutorial demonstrates using commands in various ways in a web application based on Siemens Web Framework. In last tutorial you learned to display a list of movies in your page.

## Objective

You will learn how to:

- Add a command to add a new movie.
- Add a command to remove an existing movie.
- Add the command definitions for the same.
- Add the command placements.
- Add the command handlers.

## Pre-requisites

You must have already gone through pre-requisite software by now.

**Recommended:**

Complete Selection Summary before proceeding.

**Otherwise:**

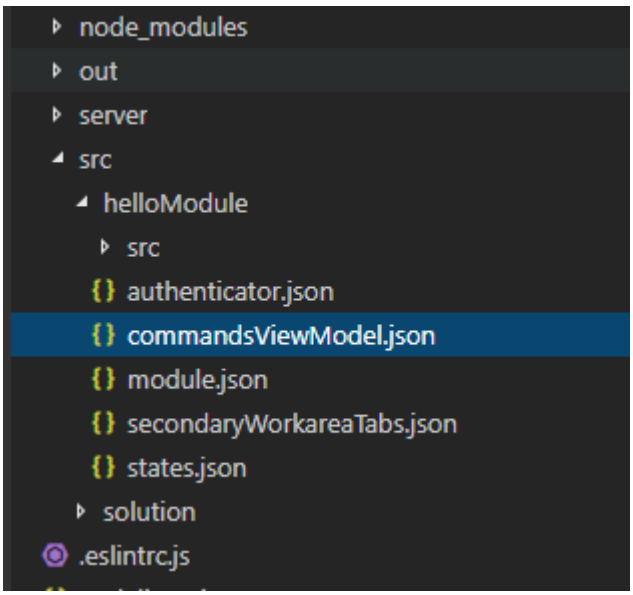
Download the source code for the **selection_summary** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Add commandsViewModel.json for your module

Add a new file `commandsViewModel.json` file in the `swfTutorialMain` folder as a sibling to 'states.json' or 'module.json'. Each module can contain a single 'commandsViewModel.json' which has the definition of all commands definitions, its placement and handlers for that module. When the web application is built, all these files(from each module) roll up to one single file. Thus, there should not be any naming conflicts.

To know more, refer **Declarative Commands**.

```
{
    "commands": {
    },
    "commandHandlers": {
    },
    "commandPlacements": {
    },
    "conditions": {
    },
    "actions": {
    },
    "onEvent": [],
    "messages": {
    },
    "i18n": {
    }
}
```





Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 2: Add a command "Add Movie" to the right wall to add a new movie to the list of movies.

1. Add the "AddMovieCmd" command definitions, its placement.

Add the following code snippet to define the `AddMovieCmd` with title and an icon id to show an add icon on it.

```
{
    "commands": {
        "AddMovieCmd": {
            "iconId": "cmdAdd",
            "title": "Add Movie"
        }
    }
}
```

The `title` attribute signifies the label to be shown for the command. This again is displayed as a tooltip on the command.

You can also have a tooltip with title and decription using the below configuration.

```
{
    "commands": {
        "AddMovieCmd": {
            "iconId": "cmdAdd",
            "title": "Add Movie",
            "description": "Allows to add movies to the list."
        }
    }
}
```

You may also use your own view by defining extended tooltip configuration as below and display any custom message. Try that as an assignment.

```
{
    "commands": {
        "AddMovieCmd": {
            "iconId": "cmdAdd",
            "title": "Add Movie",
            "extendedTooltip": {
                "view": "extendedTooltipForAddmovie"
            }
        }
    }
}
```

1. Place the command in right wall

Add the following code snippet to place the command on the right wall of your page. This is done by using the pre-defined right wall identifier `aw_workarea_toolbar`, referred to as UI anchor, for a page. Multiple commands can be contibuted to a UI anchor. Sequence of the commands can be handled through `priority`.

```
{
    "commandPlacements": {
        "AddMovieCmdPlacement": {
            "id": "AddMovieCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "priority": 20
        }
    }
}
```

1. Attach its handler that performs an action

Add the following code snippet to attach a handler to `AddMovieCmd` in `commandHandlers` section. Here `addMovieCommandHandler` would perform an action `addMovie` when nothing is selected in the list movies. This is achieved through "conditions".

```
{
    "commandHandlers": {
        "addMovieCommandHandler": {
            "id": "AddMovieCmd",
            "action": "addMovie",
            "activeWhen": {
                "condition": "conditions.noSelection"
            },
            "visibleWhen": {
                "condition": "conditions.noSelection"
            }
        }

    }
}
```

1. Define the appropriate condition to show the command only when nothing is selected in list.

Add a condition `noSelection` which uses application context ( `ctx` ) to determine if there are selections.

```
{
    "conditions": {
        "noSelection": {
            "expression": "ctx.mselected.length === 0"
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. Define the action that needs to be performed when "Add Movie" is clicked.

As done in previous tutorial, you may define `actions` in `commandsViewModel.json` as it is also considered as a view model by the declarative framework. Here, on the click of "Add Movie", a panel should open on the right side (identified by `aw_toolsAndInfo`) of the page.

```
{
    "actions": {
        "addMovie": {
            "actionType": "JSFunction",
            "method": "activateCommandPanel",
            "deps": "js/commandPanel.service",
            "inputData": {
                "commandId": "AddMovieCmd",
                "location": "aw_toolsAndInfo"
            }
        }
    }
}
```

1. Associate a view and a view model to the panel.

The view is named using the "commandId" mentioned while activating command panel. The framework internally uses "aw-include" to embed contents of the view `AddMovieCmdView.html` if provided. Thus let's define a new file with this name in "html" folder.

```
<aw-command-panel caption="i18n.commandPanelCaption">
    <aw-panel-body>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button><aw-i18n>i18n.addMovie</aw-i18n></aw-button>
    </aw-panel-footer>
</aw-command-panel>
```

This panel uses "aw-command-panel" which is recommended to be used for panels which gets displayed on commands. It has a body and a footer. The contents of the panel go in body while the action buttons go in the footer. Here we have added "Add" button.

Add a file `AddMovieCmdViewModel.json` in viewmodel folder with following content.

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "data": {},
    "actions": {
    },
    "dataProviders": {},
    "onEvent": [],
    "i18n": {
```

```
        "commandPanelCaption": ["helloModuleMessages"],
        "addMovie": ["helloModuleMessages"]
    },
    "messages": {},
    "conditions": {}
}
```

Update `helloModuleMessages.json` in i18n folder with following content.

```
{
    "deleteMovieConfirmation": "Do you want to delete movie \"{0}\"?",
    "cancel": "Cancel",
    "delete": "Delete",
    "commandPanelCaption": "Add Movie",
    "addMovie": "Add",
    "addMovieCmdDesc": "Allows to add a movie to the list"
}
```

## Step 3: Build the Solution

Run the following build commands to build and deploy the site.

```
aw clean && aw start
```

Use the url **http://localhost:8181/** to load the application in browser.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Try hovering on the `Add Movie` command and then click on it.

You might want to try some hand at localization after learning how to do it.

## Step 4: Add a command "Delete Movie" to the primary workarea toolbar delete/ remove an existing movie from the list.

1. Add the `DeleteMovieCmd` command definitions, its placement.

   Add the following code snippet to define the `DeleteMovieCmd` with title and an icon id to show an delete icon on it.

```
{
    "commands": {
        "DeleteMovieCmd": {
            "iconId": "cmdDelete",
            "title": "Delete Movie"
        }
    }
}
```

2. Place the command on primary workarea toolbar.

Add the following code snippet to place the command on the primary workarea toolbar. This is done by using the pre-defined UI anchor `aw_primaryWorkArea`.

```
{
    "commandPlacements": {
        "deleteMovieCmdPlacement": {
            "id": "DeleteMovieCmd",
            "uiAnchor": "aw_primaryWorkArea",
            "priority": 100
        }
    }
}
```

3. Attach its handler that performs an action

   Add the following code snippet to attach a handler to `DeleteMovieCmd` in `commandHandlers` section. Here `deleteMovieCommandHandler` would perform an action `deleteMovieConfirmation` unconditionally. The "Delete Movie" command should be enabled only when there is one selection using condition `SingleSelection`.

```
{
    "commandHandlers": {
        "deleteMovieCommandHandler": {
            "id": "DeleteMovieCmd",
            "action": "deleteMovieConfirmation",
            "activeWhen": true,
            "visibleWhen": true,
            "enableWhen": {
                "condition": "conditions.SingleSelection"
            }
        }
    }
}
```

4. Define the appropriate condition to show the command only when only one movie is selected in list.

   Add the following code snippet add a condition `SingleSelection` which uses application context ( `ctx` ) to determine if there are selections.

```
{
    "conditions": {
        "SingleSelection": {
            "expression": "ctx.mselected.length === 1"
        }
    }
}
```

5. Define the action that needs to be performed when "Delete Movie" is clicked.


Siemens Web Framework
Unpublished work. © 2025 Siemens

On the click of "Delete Movie", it should ask the user for a confirmation. Here the action is firing an event `confirmDeleteMovieEvent` to display notification message for user confirmation.

```
{
    "actions": {
        "deleteMovieConfirmation": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [{
                    "name": "confirmDeleteMovieEvent"
                }]
            }
        }
    }
}
```

6. Show a notification message on the delete action.

   This can be displayed by defining a message `deleteMovieConfirmPopup` which will be processed when `confirmDeleteMovieEvent` is fired.

```
{
    "onEvent": [{
        "eventId": "confirmDeleteMovieEvent",
        "message": "deleteMovieConfirmPopup"
    }],
    "messages": {
        "deleteMovieConfirmPopup": {
            "messageType": "WARNING",
            "messageText": "Do you want to delete movie \"{0}\"? ",
            "navigationOptions": [{
                    "option": "cancel",
                    "text": "Cancel",
                    "action": ""
                },
                {
                    "option": "delete",
                    "text": "Delete",
                    "action": "deleteMovie"
                }
            ],
            "messageTextParams": [
                "{{ctx.selected.cellHeader1}}"
            ]
        }
    }
}
```

On clicking the "Delete" button on notification, the action `deleteMovie` finally makes a REST call to the node server to remove the movie. On success of the REST call, the primary workarea needs to be updated to show the removal of the movie.

```
{
    "actions": {
        "deleteMovie": {
            "actionType": "RESTService",
            "method": "DELETE",
            "inputData": {
                "request": {
                    "method": "DELETE",
                    "url": "/afx/movies/{{ctx.selected.uid}}"
                }
            },
            "events": {
                "success": [{
                    "name": "aw.refreshPrimaryWorkArea"
                }]
            }
        }
    }
}
```

`Important` - Update `favoritesListViewModel.json` to listen to the event `aw.refreshPrimaryWorkArea` and refresh the PWA. This would also be utilized when a movie is added to the list.

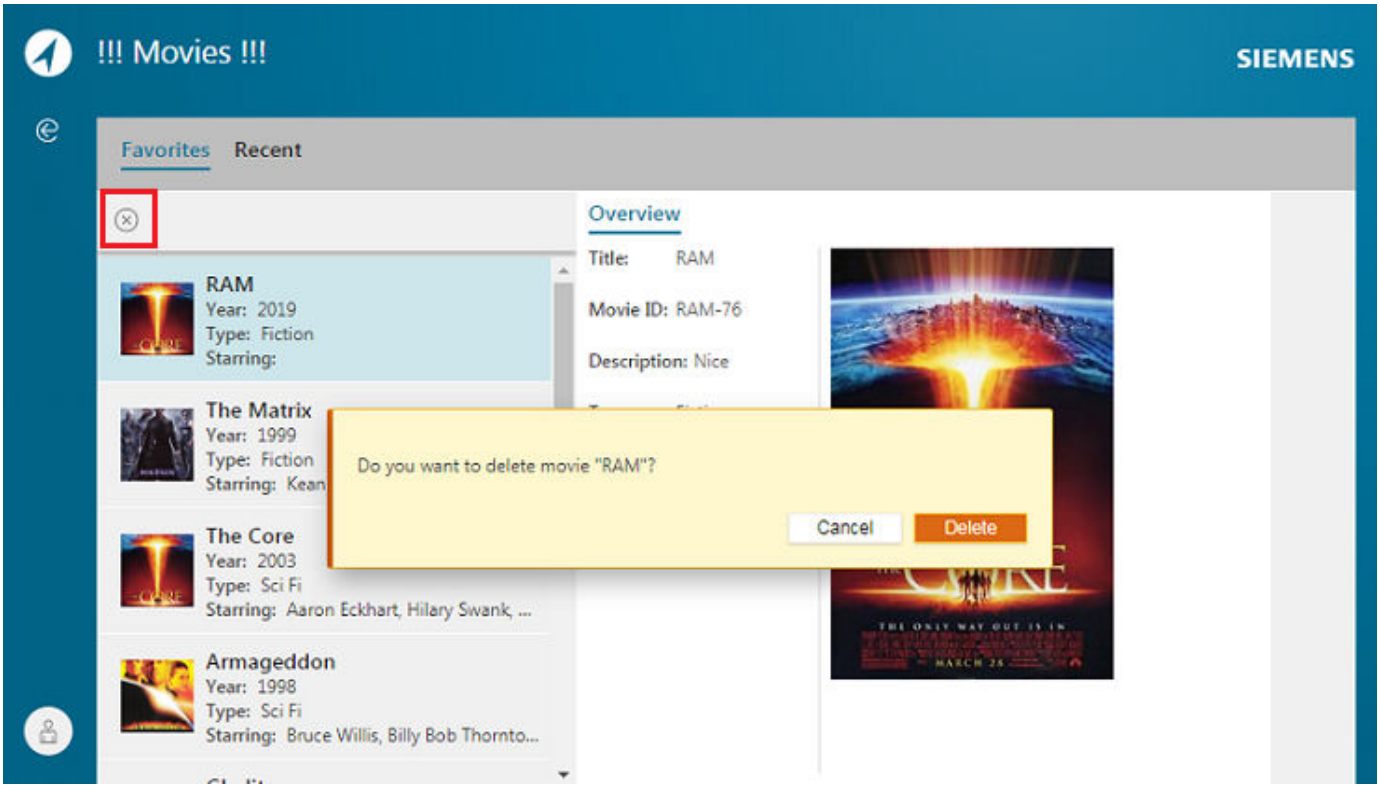
```
{
    "actions": {
        "refreshDataProvider": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [{
                    "name": "primaryWorkarea.reset"
                }]
            }
        }
    }
    "onEvent": [{
        "eventId": "aw.refreshPrimaryWorkArea",
        "action": "refreshDataProvider"
    }]
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 5: Build the Solution

Run the following build commands to build and deploy the site.

```
aw clean && aw start
```




## Step 6: Configuring display commands to switch between views

Display commands will be there in primary workarea toolbar according to default **viewModes**. One can specify the required `viewModes` by configuring `pwaViewModes`, `secondaryWorkarea` and `supportedLayouts` in module's `state.json` file at sublocation.

```
{
    "myFavs": {
        "data": {
            "pwaViewModes":["list","table"],
            "secondaryWorkarea":true,
            "supportedLayouts": [ "Awp0SinglePaneLayout",
"Awp0TopBottomLayout", "Awp0LeftRightLayout"]
        }
    }
}
```

## Step 7: Add view file for table display mode which would be reused for Table with Left-Right view also.

As mentioned in tutorial **URL Routing and Page Layout**, the `viewBase` provides ways to configure ready made view modes, like, *Lists*, *Table*, etc. The way `favoritesListView.html` and `favoritesListViewModel.json` was added, add new files `favoritesTableView.html` and `favoritesTableViewModel.json` and put in `html` and `viewmodel` folders in `src` directory of `swfTutorialMain`.

Update `favoritesTableView.html` with following code snippet:

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

Update `favoritesTableViewModel.json` with following code snippet:

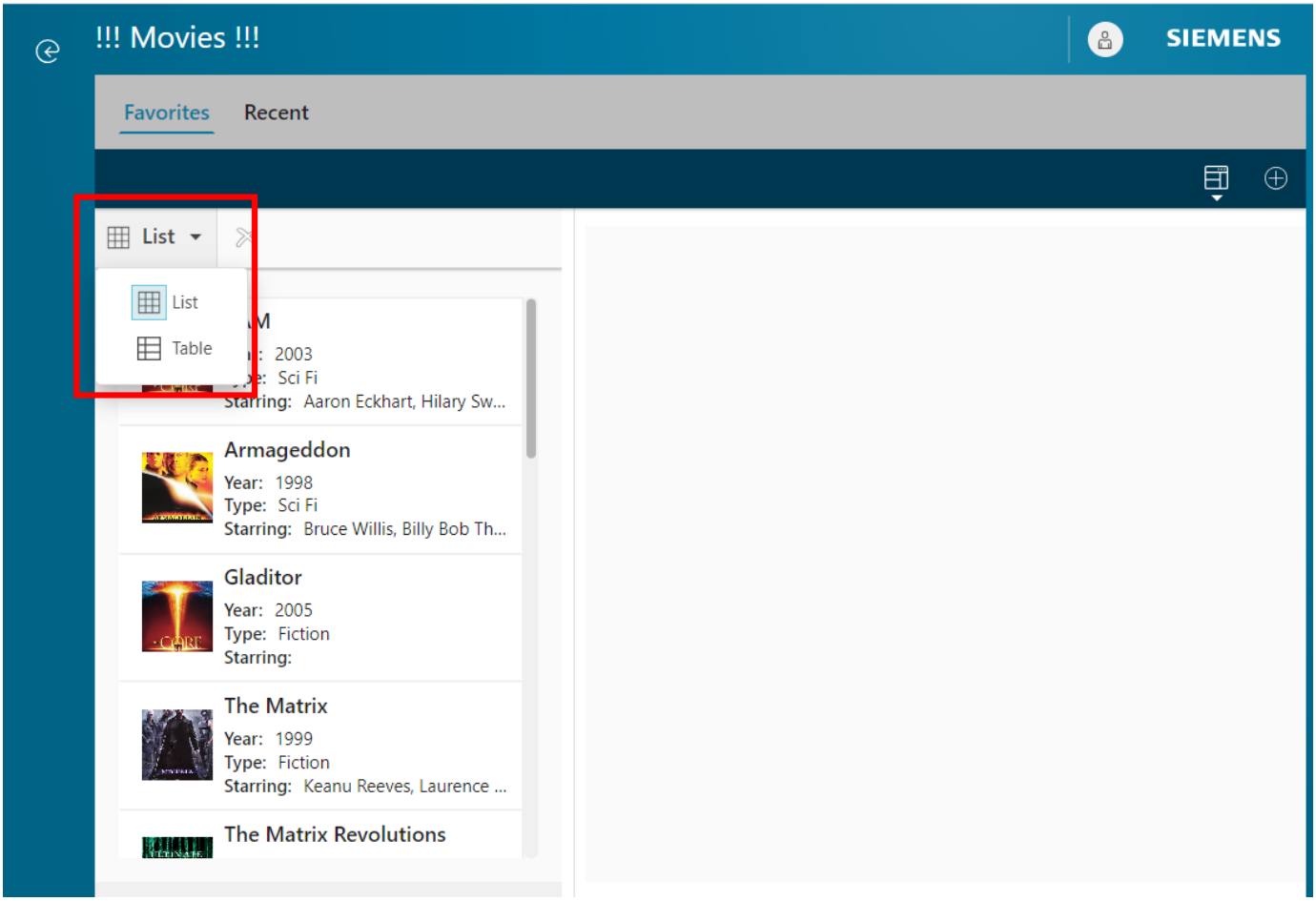
```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "data": {
        "label": {
            "displayName": "Table",
            "type": "STRING",
            "dispValue": "To be added in subsequent Tutorials"
        }
    }
}
```

## Step 8: Build the Solution

Run the following build commands to build and deploy the site.

```
aw clean && aw start
```


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Summary

In this tutorial, you learned how to:

- Add command and associate action with it
- Design the page with various layout and container elements.

## References

- **Solutions**
- **States**
- **Declarative UI**
- **View**
- **View model**
- **Audit**

# Property widgets and validation

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

This tutorial demonstrates use of different types of property widgets such as, textarea, radio button etc.

## Objective

You will learn how to add different types of property widgets to "Add Movie" panel.

## Pre-requisites

You must have already gone through **Pre-requisite software** by now.

**Recommended:**

Complete **Working with Commands** before proceeding.

**Otherwise:**

Download the source code for the **working_with_commands** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Add *aw-textbox* in `AddMovieCmdView.html`.

1. Add the following content in the `AddMovieCmdView.html` file.

To use the ui element/property widgets "aw-textbox" to allow user to enter the name of the movie. The "Add Movie" panel consists of panel body and panel footer. The property widgets are added in panel body.

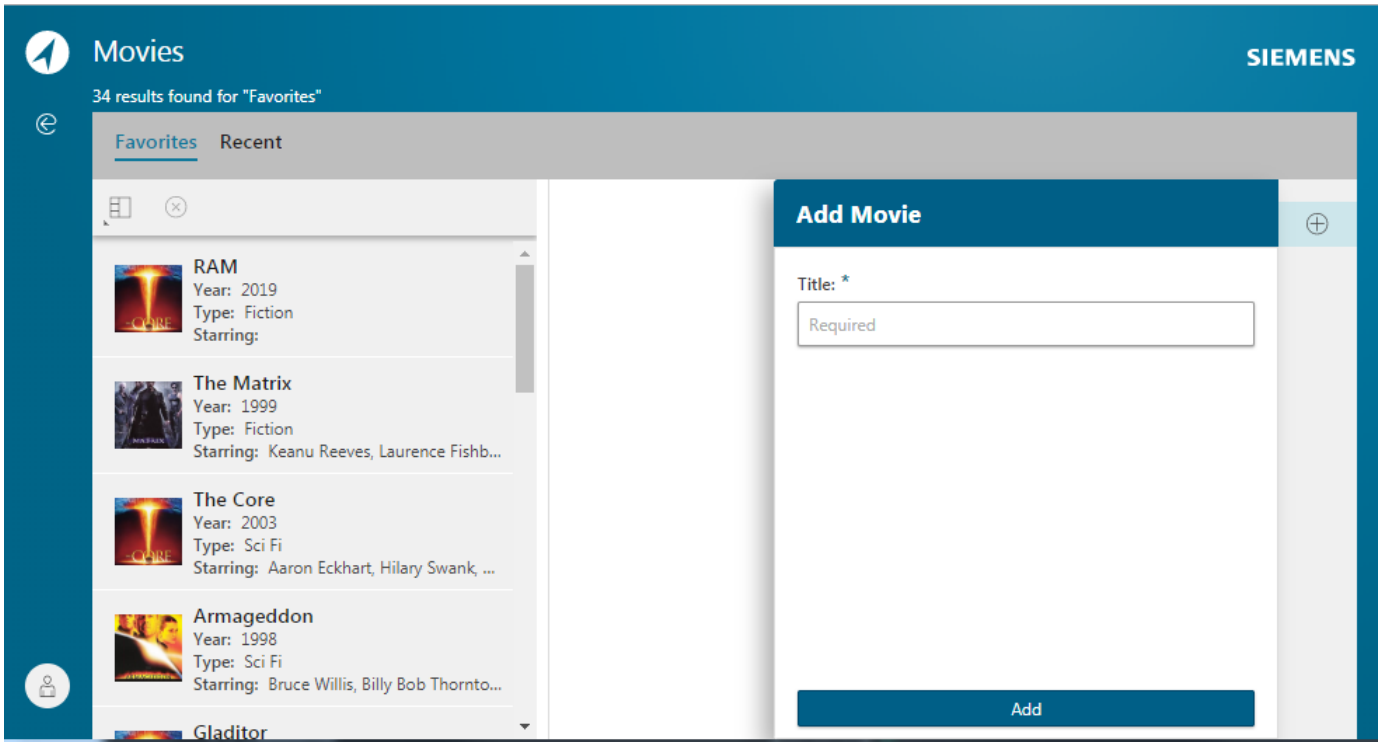
```
<aw-panel-body>
    <aw-textbox prop="data.Title"></aw-textbox>
</aw-panel-body>
```

1. Add the following content in the `AddMovieCmdViewModel.json` file.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Here the "Title" property defined in view model is being bound with the "aw-textbox" widget in view. Thus any update to the widget would update the view model property as well. Similarly, if the view model property is updated, the widget would display the same.

```
{
    "data": {
        "Title": {
            "displayName": "Title",
            "type": "STRING",
            "isRequired": true,
            "isEditable": "true",
            "dbValue": "",
            "dispValue": ""
        }
    }
}
```




## Step 2: Similarly add all other type of properties in `AddMovieCmdView.html`.

1. Add the following content in the `AddMovieCmdView.html` file to allow user to add the year of the movie.

```
...
<aw-panel-body>
    <aw-numeric prop="data.Year"></aw-numeric>
    <aw-textbox prop="data.Type"></aw-textbox>
```

```
    <aw-radiobutton prop="data.format"></aw-radiobutton>
    <aw-textarea prop="data.Description"></aw-textarea>
    <aw-widget prop="data.Starring"></aw-widget>
</aw-panel-body>
...
```

1. Add the following content in the `AddMovieCmdViewModel.json` file.

```
{
    "data": {
        "Year": {
            "displayName": "Year",
            "type": "INTEGER",
            "isRequired": true,
            "isEditable": "true",
            "dbValue": "",
            "dispValue": ""
        },
        "Type": {
            "displayName": "Genre",
            "type": "STRING",
            "isRequired": true,
            "isEditable": "true",
            "dbValue": "",
            "dispValue": "",
            "dataProvider": "movieGenreProvider"
        },
        "format": {
            "displayName": "Format",
            "type": "BOOLEAN",
            "isRequired": "false",
            "isEditable": "true",
            "dbValue": true,
            "labelPosition": "PROPERTY_LABEL_AT_SIDE",
            "propertyRadioTrueText": "2D",
            "propertyRadioFalseText": "3D"
        },
        "Description": {
            "displayName": "Plot",
            "type": "STRING",
            "isRequired": "false",
            "isEditable": "true",
            "dbValue": "",
            "dispValue": ""
        },
        "Starring": {
            "displayName": "Cast",
            "type": "STRING",
            "isRequired": false,
            "isArray": "true"
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Note**

The addition of a data provider "movieGenreProvider" to "Genre" property. This would be later converted to a list of values(LOV).

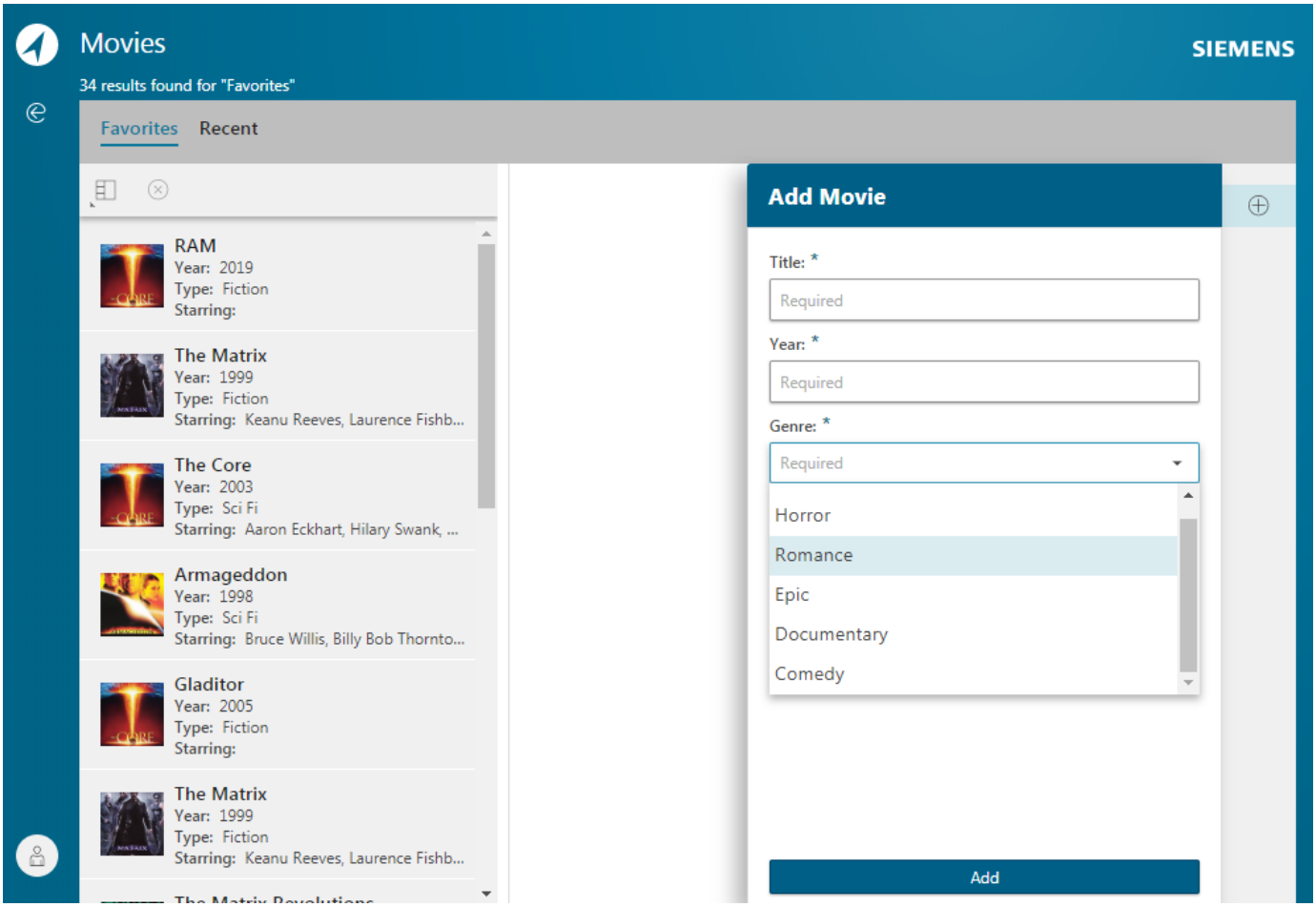
1. To display a list of movie genres(commonly known as LOV), add the following content in `AddMovieCmdViewModel.json` file under `data` section.

```
{
    "data": {
        "movieGenres": [{
                "propDisplayValue": "Horror",
                "propInternalValue": "horror"
            },
            {
                "propDisplayValue": "Romance",
                "propInternalValue": "romance"
            },
            {
                "propDisplayValue": "Epic",
                "propInternalValue": "epic"
            },
            {
                "propDisplayValue": "Documentary",
                "propInternalValue": "documentary"
            },
            {
                "propDisplayValue": "Comedy",
                "propInternalValue": "Comedy"
            }
        ]
    }
}
```

1. Add the following content `AddMovieCmdViewModel.json` file.

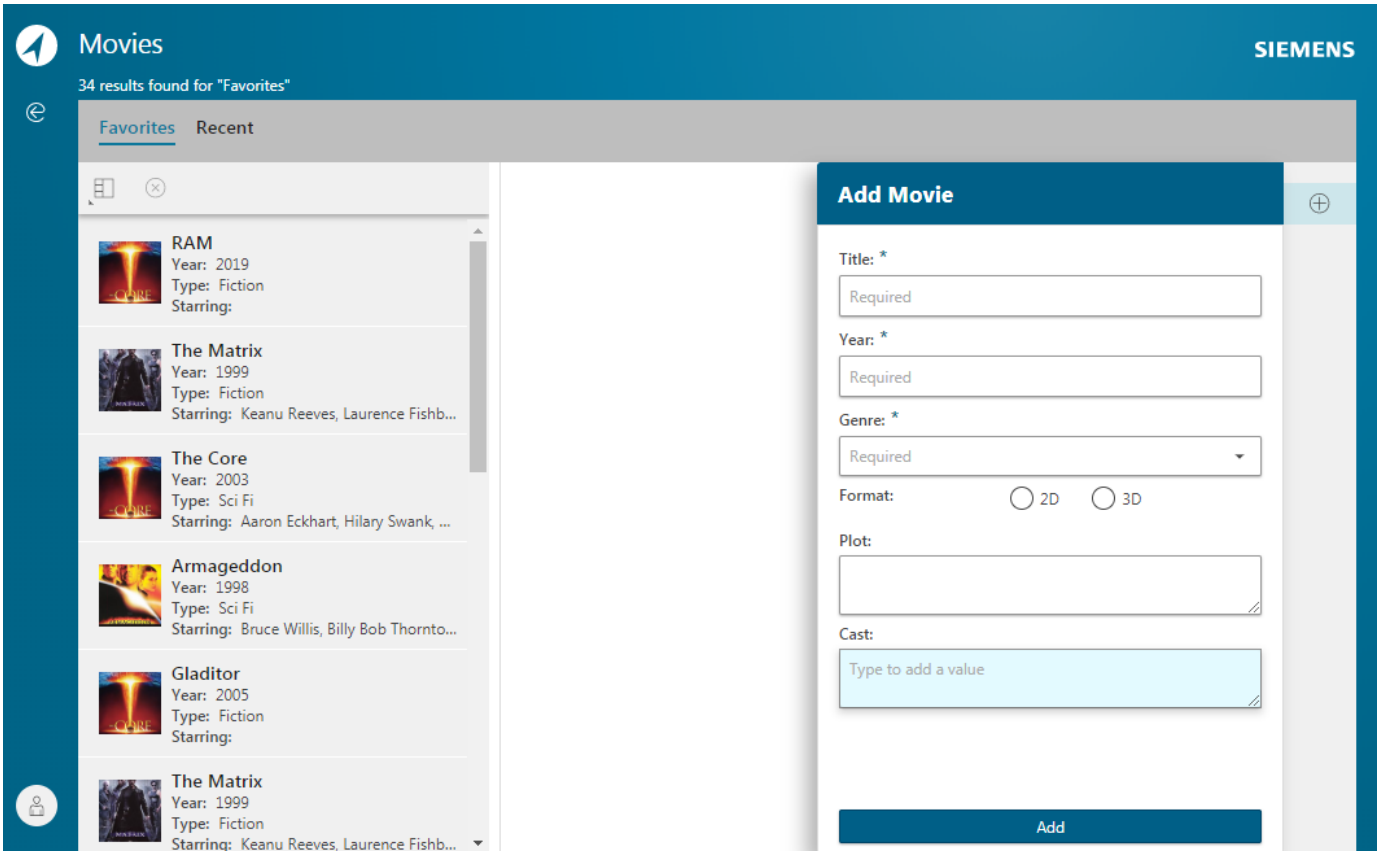
```
{
    "dataProviders": {
        "movieGenreProvider": {
            "response": "{{data.movieGenres}}"
        }
    }
}
```

Here the LOV is provided with data using the data provider "movieGenreProvider".




Finally the "Add Movie" should look like below:


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Step 3: Perform action to add a movie.

Update `AddMovieCmdView.html` to add action on the click of the `Add` button. This would fire a REST call to add an entry to the movie database.

```
...
<aw-panel-footer>
    <aw-button action="addMovie"><aw-i18n>i18n.addMovie</aw-i18n></aw-
button>
</aw-panel-footer>
...
```

Add the following content in `AddMovieCmdViewModel.json` file to perform action on the `Add` button to add a movie.

```
{
    "actions": {
        "addMovie": {
            "actionType": "RESTService",
            "inputData": {
                "request": {
                    "method": "POST",
                    "url": "/afx/movies",
```

```
                    "data": {
                        "type": "{{data.Type.dbValue}}",
                        "title": "{{data.Title.dbValue}}",
                        "year": "{{data.Year.dbValue}}",
                        "desc": "{{data.Description.dbValue}}",
                        "starring": "{{data.Starring.dbValue}}"
                    }
                }
            },
            "outputData": {}
        }
    }
}
```

:triangular_flag_on_post: Once the `addMovie` response returns successfully, we need to fire two events

1. The 'Add Movie' panel should be closed - For this fire event `complete` with source as toolsAndInfoPanel.
2. The Primary workarea needs to be refreshed to display the new Movie in the list - For this fire `aw.refreshPrimaryWorkArea` event. Please note that this event is already listened in `favoritesListVieModel.json`.

```
{
    "actions": {
        "addMovie": {
            "events": {
                "success": [{
                        "name": "aw.refreshPrimaryWorkArea"
                    },
                    {
                        "name": "complete",
                        "eventData": {
                            "source": "toolAndInfoPanel"
                        }
                    }

                ]
            }
        }
    }
}
```

## Step 4: Build the Solution

Run the following build commands to build and deploy the site.

```
set SITE=site_tutorial
aw build site_tutorial
aw start
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 5: Add a validation criteria to "Year of Release" property.

You may add validation criteria to a property.

```
{
    "data": {
        "Year": {
            "validationCriteria": [ {
                "condition": "conditions.invalidYear",
                "value": "Year should not be greater than 2021"
            } ]
        }
    }
}
```

Add the relevant condition

```
{
    "conditions": {
        "invalidYear": {
            "expression": {
                "$source": "data.Year.dbValue",
                "$query": {
                    "$gt": 2021
                }
            }
        }
    }
}
```

Note that since "Genre", "Title" and "Year of Release" has been marked "Required" in view model. To disable the "Add" button untill these values are fields are filled we can update `AddMovieCmdView.html` with `enable-when`.

```
<aw-panel-footer>
    <aw-button
action="addMovie" enable-when="formProp.attributes.valid"><aw-
i18n>i18n.addMovie</aw-i18n></aw-button>
</aw-panel-footer>
```

## Step 6: Build the Solution

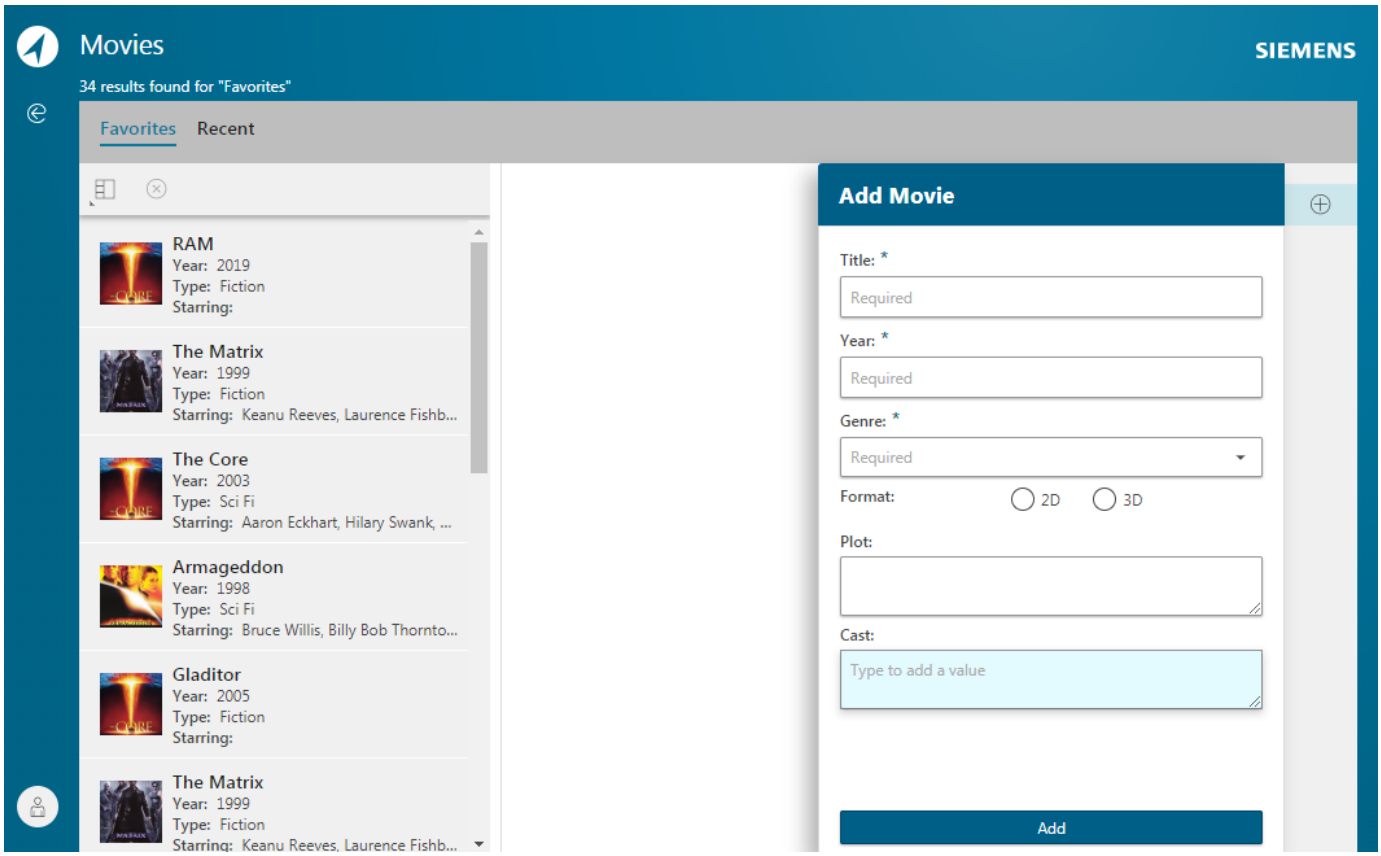
Run the following build commands to build and deploy the site.

```
aw build site_tutorial
set SITE=site_tutorial
```

```
aw start
```

While adding a movie, try entering an Year value > 2021.

## Summary




In this tutorial, you learned how to:

- Add different types of property widgets in a panel and use data binding to bind properties in view model.
- Different kind of form validations.

# Editing in a summary view

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. A version of this tutorial compatible with Siemens Web Framework v4.x can be found **here**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Let's add editing capabilities to update properties of a movie in a form, ie. in Overview or Summary.

## Objective

You will learn how to:

- Add "Edit" group command with its child commands "Start Edit", "Save Edits" and "Cancel Edits".
- Add respective handlers.
- Show how edit related events are fired and editing capabilities are turned on in Overview.
- Show how conditionally visibility of commands can be handled.

## Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **Working with Commands** before proceeding.

**Otherwise:**

Download the source code for the **working_with_commands** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Add a group command `EditMovieGroupCmd` to support edit in Overview.

To show an "Edit" command on right wall, a group command `EditMovieGroupCmd` would be a defined. The "Edit" command displays a set of child commands "Start Edit", "Save Edits" and "Cancel Edits". Similar definitions of adding a command, its placement, handler and action shoould be defined.

Update `commandViewModel.json` file in the `swfTutorialMain` folder for adding the command definitions.

1. Add the edit command definitions. Please note that `EditMovieGroupCmd`is a group command while the other commands are normal commands.

```
{
    "commands": {
        "EditMovieGroupCmd": {
            "iconId": "cmdEdit",
            "isGroup": true,
            "title": "Edit"
```


Unpublished work. © 2025 Siemens

```
        },
        "StartEditMovieCmd": {
            "iconId": "cmdEdit",
            "title": "Start Edit"
        },
        "SaveEditMovieCmd": {
            "iconId": "cmdSave",
            "title": "Save Edit"
        },
        "CancelEditMovieCmd": {
            "iconId": "cmdSave",
            "title": "Cancel Edit"
        }
    }
}
```

2. Place the "Edit" command on right wall

   Add the following code snippet to place the command on the right wall of your page using the pre-defined right wall identifier `aw_workarea_toolbar`. The child commands are added as to the right wall with parent command as `EditMovieGroupCmd`.

```
{
    "commandPlacements": {
        "editMovieGroupCmdPlacement": {
            "id": "EditMovieGroupCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "priority": 25
        },
        "startEditMovieCmdPlacement": {
            "id": "StartEditMovieCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "parentGroupId": "EditMovieGroupCmd",
            "priority": 1
        },
        "saveEditMovieCmdPlacement": {
            "id": "SaveEditMovieCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "parentGroupId": "EditMovieGroupCmd",
            "priority": 2
        },
        "cancelEditMovieCmdPlacement": {
            "id": "CancelEditMovieCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "parentGroupId": "EditMovieGroupCmd",
            "priority": 3
        }
    }
}
```

3. Attach its handler that performs an action


Siemens Web Framework
Unpublished work. © 2025 Siemens

Add the following code snippet to attach a handler to the commands in `commandHandlers` section. Here `startEditMovieCommandHandler` would perform an action `startEditMovie` when only one movie is selected in the list of movies.

```
{
    "commandHandlers": {
        "startEditMovieCommandHandler": {
            "id": "StartEditMovieCmd",
            "action": "startEditMovie",
            "activeWhen": true,
            "visibleWhen": true,
            "enableWhen": {
                "condition": "conditions.EnableStartEdit"
            }
        },
        "saveEditMovieCommandHandler": {
            "id": "SaveEditMovieCmd",
            "action": "saveEditMovie",
            "activeWhen": true,
            "visibleWhen": true,
            "enableWhen": {
                "condition": "conditions.EnableSaveCancelEdit"
            }
        },
        "cancelEditMovieCommandHandler": {
            "id": "CancelEditMovieCmd",
            "action": "cancelEditMovie",
            "activeWhen": true,
            "visibleWhen": true,
            "enableWhen": {
                "condition": "conditions.EnableSaveCancelEdit"
            }
        }
    }
}
```

4. Define the appropriate condition to show the command at the right time.

   "Start Edit" command should not be enabled when edit is in progress. However, the "Save Edits" and "Cancel Edits" should show up only when edit is in progress. Again we use the application context here to determine if edit is in progress.

```
{
    "conditions": {
        "EnableStartEdit": {
            "expression": "ctx.mselected.length === 1 &&
ctx.editInProgress === false"
        },
        "EnableSaveCancelEdit": {
            "expression": "ctx.editInProgress === true"
        }
    }
}
```

5. Define the action that needs to be performed when "Add Movie" is clicked.

   As done in previous tutorial, you may define `actions` in `commandsViewModel.json` as it is also considered as a view model by the declarative framework. Here, on the click of "Start Edit", the movie details in the Secondary Work Area go into edit mode based on the props that are set as editable.

```
{
    "actions": {
        "startEditMovie": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [
                    {
                        "name": "movie.startEdit",
                        "excludeLocalDataCtx": true
                    }
                ]
            }
        },
        "saveEditMovie": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [
                    {
                        "name": "movie.saveEdit",
                        "excludeLocalDataCtx": true
                    }
                ]
            }
        },
        "cancelEditMovie": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [
                    {
                        "name": "movie.cancelEdit",
                        "excludeLocalDataCtx": true
                    }
                ]
            }
        }
    }
}
```

   Please note that the events fired from the command needs to be listened to by "Overview" tab


Siemens Web Framework
Unpublished work. © 2025 Siemens

6. To add actions for *saveEditMovie* and *startEditMovie*, add the following code in `favSelectionDetailsViewModel.json` under `actions` section.

```
{
    "actions":{
        "startEditMovie": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/movies/{{ctx.selected.uid}}",
                    "params": {
                        "operation": "startEdit"
                    }
                }
            },
            "dataParsers": [{
                "id": "selectedMovieParser",
                "responseObjs": "data.search"
            }],
            "outputData": {
                "selectedObj": "data.search",
                "ctx.editInProgress": true
            }
        },
        "saveEditMovie": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "PUT",
            "inputData": {
                "request": {
                    "method": "PUT",
                    "url": "/afx/movies/{{ctx.selected.uid}}",
                    "data": {
                        "updates": {
                            "Description":
"{{data.selectedObj.props.Description.newValue}}"
                        }
                    }
                }
            },
            "dataParsers": [{
                "id": "selectedMovieParser",
                "responseObjs": "data.search"
            }],
            "outputData": {
                "selectedObj": "data.search",
                "ctx.editInProgress": false
            }
        }
    }
}
```

7. To call these actions, add the following code in in `favSelectionDetailsViewModel.json` under `onEvent` section.

```
{
    "onEvent": [
        {
            "eventId": "movie.startEdit",
            "action": "startEditMovie"
        },
        {
            "eventId": "movie.saveEdit",
            "action": "saveEditMovie"
        },
        {
            "eventId": "movie.cancelEdit",
            "action": "getSelectedObject"
        }
    ]
}
```

8. Update the `outputData` property of the `getSelectedObject` action in `favSelectionDetailsViewModel.json`.

```
{
    "actions:": {
        "getSelectedObject": {
            "outputData": {
                "ctx.editInProgress": false
            }
        }
    }
}
```

## Step 3: Build the Solution

Run the following build commands to build and deploy the site.

```
set SITE=site_tutorial
aw build site_tutorial
aw start
```

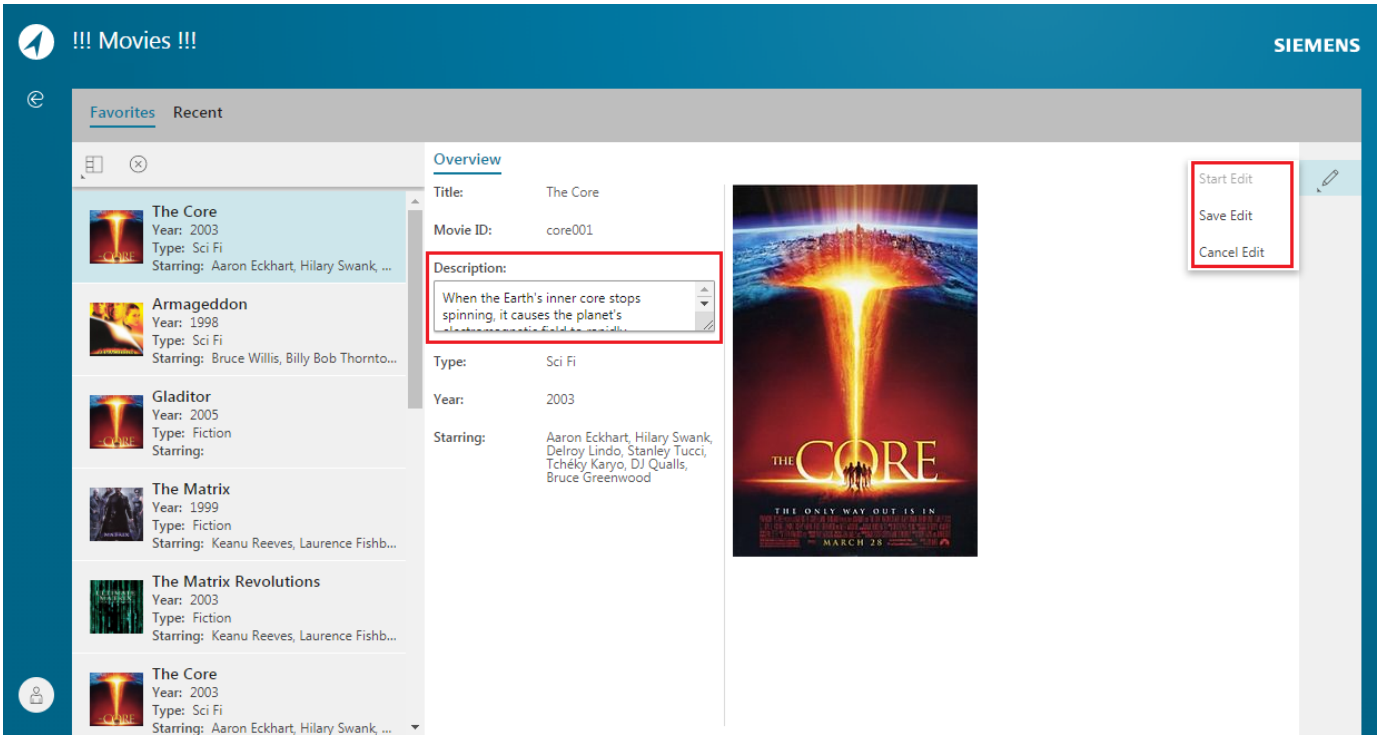
## Summary

In this tutorial you learned how to:

- Add "Edit" group command with its child commands "Start Edit", "Save Edits" and "Cancel Edits".
- Add respective handlers.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- Show how edit related events are fired and editing capabilities are turned on in Overview.
- Show how conditionally visibility of commands can be handled.




# Localization

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

This tutorial shows how Siemens Web Framework provides a way to localize a web client solution.

## Objective.

You will learn:

- How support for localization (L10N) is provided with Siemens Web Framework.
- How to change the language manually by providing different URLs to our users or updating the URL.
- How to provide a command for the user to change the language.

## Pre-requisites

**Recommended:**

Complete **Edit in summary view** before proceeding.

**Otherwise:**

Download the source code for the **edit_in_summary_view** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Localizing Command text

**Commands** are defined in `commandsViewModel.json`. The *title* field in the `commands` provides the user-facing display text and tooltip for a command as shown in the following image.

To show this text in different languages (for example, English, German etc.), we need to add this in `helloModuleMessage.json` under `i18n` folder as shown in the following image.


Unpublished work. © 2025 Siemens

Update `helloModuleMessage.json` under `i18n` folder with the following content.

```
{
    "addMovie": "Add Movie",
    "newWorkspaceName": "New Workspace",
    "helloText": "!! Hello World !!",
    "commandPanelCaption": "Add Movie",

    "movieTitle": "Title",
    "movieType": "Genre",
    "movieFormat": "Format",
    "movieCast": "Stars",
    "movieYear": "Year",
    "Description": "Plot",

    "horror": "Horror",
    "romance": "Romance",
    "epic": "Epic",
    "documentary": "Documentary",
    "comedy": "Comedy",

    "deleteMovieTooltip": "Delete Movie",
    "Awp0ShowImageViewTitle": "Images",
    "Awp0ShowListViewTitle": "List",
    "Awp0ShowSummaryViewTitle": "List with Summary",
    "Awp0ShowTableSummaryViewTitle": "Table with Summary",
    "Awp0ShowTableViewTitle": "Table",
    "editTooltip": "Edit",
    "startEditTooltip": "Start Edit",
    "saveEditTooltip": "Save Edit",
    "cancelEditTooltip": "Cancel Edit",
    "deleteMovieConfirmation": "Do you want to delete movie \"{0}\"?",
    "cancel": "Cancel",
    "delete": "Delete",
    "changeLocale": "Change Locale",
    "englishLocale": "English",
    "germanLocale": "German"
}
```

Update `commandsViewModel.json` with the following content to display localized text.

```
{
    "commands": {
        "AddMovieCmd": {
            "iconId": "cmdAdd",
            "title": "{{i18n.addMovie}}"
        },
        "DeleteMovieCmd": {
            "iconId": "cmdDelete",
            "title": "{{i18n.deleteMovieTooltip}}"
        },
        "Awp0ModelObjListDisplayToggles": {
            "isShuttle": true
        },
        "Awp0ShowListView": {
            "iconId": "cmdListView",
            "title": "{{i18n.Awp0ShowListViewTitle}}"
        },
        "Awp0ShowSummaryView": {
            "iconId": "cmdListSummaryView",
            "title": "{{i18n.Awp0ShowSummaryViewTitle}}"
        },
        "Awp0ShowTableSummaryView": {
            "iconId": "cmdTableSummaryView",
            "title": "{{i18n.Awp0ShowTableSummaryViewTitle}}"
        },
        "Awp0ShowTableView": {
            "iconId": "cmdTableView",
            "title": "{{i18n.Awp0ShowTableViewTitle}}"
        },
        "EditMovieGroupCmd": {
            "iconId": "cmdEdit",
            "isGroup": true,
            "title": "{{i18n.editTooltip}}"
        },
        "StartEditMovieCmd": {
            "iconId": "cmdEdit",
            "title": "{{i18n.startEditTooltip}}"
        },
        "SaveEditMovieCmd": {
            "iconId": "cmdSave",
            "title": "{{i18n.saveEditTooltip}}"
        },
        "CancelEditMovieCmd": {
            "iconId": "cmdSave",
            "title": "{{i18n.cancelEditTooltip}}"
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

Update `i18n` in `commandsViewModel.json`.

```
{
    "addMovie": [
        "helloModuleMessages"
    ],
    "deleteMovieTooltip": [
        "helloModuleMessages"
    ],
    "Awp0ShowListViewTitle": [
        "helloModuleMessages"
    ],
    "Awp0ShowSummaryViewTitle": [
        "helloModuleMessages"
    ],
    "Awp0ShowTableSummaryViewTitle": [
        "helloModuleMessages"
    ],
    "Awp0ShowTableViewTitle": [
        "helloModuleMessages"
    ],
    "editTooltip": [
        "helloModuleMessages"
    ],
    "startEditTooltip": [
        "helloModuleMessages"
    ],
    "saveEditTooltip": [
        "helloModuleMessages"
    ],
    "cancelEditTooltip": [
        "helloModuleMessages"
    ]
}
```

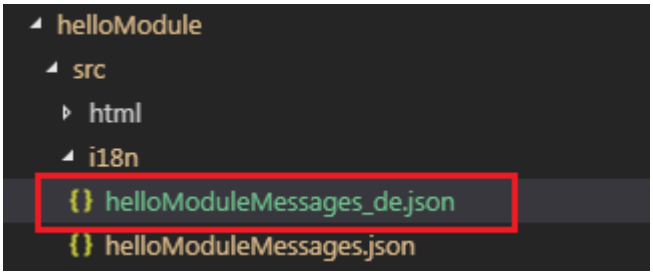
Here we **data bind** to the localized value provided in the **view model**.

## Step 2: Adding Language Resources

User-facing text for the UI is specified in JSON files in the component's `i18n` folder (for example, src\i18n).

There is one JSON file for each language you support (for example, `src\i18n\<myComponent>Messages_<locale>.json`) and a master file, which is used as the default (for example, `src\i18n\<myComponent>Messages.json`).

1. Add a set of localization files for different locales for your component in the i18n folder.




2. Add the following content in `helloModuleMessages_de.json`.

```
    {
        "addMovie": "Film hinzufügen",
        "newWorkspaceName": "New Workspace",
        "helloText": "!! Hallo Welt !!",
        "commandPanelCaption": "Film hinzufügen",

        "movieTitle": "Titel",
        "movieType": "Typ",
        "movieCast": "Filmstars",
        "movieFormat": "Format",
        "movieYear": "Jahr",
        "Description": "Handlung",

        "horror": "Grusel",
        "romance": "Romantik",
        "epic": "Epos",
        "documentary": "Dokumentarfilm",
        "comedy": "Komödie",

        "deleteMovieTooltip": "Film löschen",
        "Awp0ShowImageViewTitle": "Bilder",
        "Awp0ShowListViewTitle": "Liste",
        "Awp0ShowSummaryViewTitle": "Liste mit Zusammenfassung",
        "Awp0ShowTableSummaryViewTitle": "Tabelle mit Zusammenfassung",
        "Awp0ShowTableViewTitle": "Tabelle",
        "editTooltip": "Bearbeiten",
        "startEditTooltip": "Starten Sie Bearbeiten",
        "saveEditTooltip": "Speichern Bearbeiten",
        "cancelEditTooltip": "Bearbeiten abbrechen",
        "deleteMovieConfirmation": "Möchten Sie den Film löschen
\"{0}\"?",
        "cancel": "Abbrechen",
        "delete": "Löschen",
        "changeLocale": "Gebietsschema ändern",
        "englishLocale": "Englisch",
        "germanLocale": "Deutsch"
    }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 3: Adding a Command to Switch between Languages

Provide a command to switch between languages.

## Step 1: Add the Locale Options

Add `LoginLocale.json` in the i18n folder peer to `helloModuleMessages.json` and add the following content.

```
    {
        "cs_CZ": "Czech",
        "de_DE": "German",
        "de": "German",
        "en_US": "English",
        "en": "English",
        "es_ES": "Spanish",
        "es": "Spanish",
        "fr_FR": "French",
        "fr": "French",
        "it_IT": "Italian",
        "it": "Italian",
        "ja_JP": "Japanese",
        "ko_KR": "Korean",
        "pt_BR": "Portuguese",
        "ru_RU": "Russian",
        "zh_CN": "Chinese-China",
        "zh_TW": "Chinese-Taiwan"
    }
```

## Step 2: Add a Command to Change the Language

Add **command**, commandHandler, and commandPlacement in `commandsViewModel.json` for the user to switch language.

Add the following code in `commandsViewModel.json` under `commands` section.

```
{
    "commands" : {
        "ChangeLocaleCmd": {
            "iconId": "cmdFlatToScreen",
            "isGroup": true,
            "title": "{{i18n.changeLocale}}"
        },
        "EnglishLocaleCmd": {
            "iconId": "cmdChanges",
            "title": "{{i18n.englishLocale}}"
        },
        "GermanLocaleCmd": {
            "iconId": "cmdFreehand",
```

```
            "title": "{{i18n.germanLocale}}"
        }
    }
}
```

Add the following code in `commandsViewModel.json` under `commandHandlers` section.

```
{
    "commandHandlers": {
        "changeLocaleCommandHandler": {
            "id": "ChangeLocaleCmd",
            "activeWhen": true,
            "visibleWhen": true
        },
        "englishLocaleCommandHandler": {
            "id": "EnglishLocaleCmd",
            "action": "changeToEnglishLocale",
            "activeWhen": true,
            "visibleWhen": true
        },
        "germanLocaleCommandHandler": {
            "id": "GermanLocaleCmd",
            "action": "changeToGermanLocale",
            "activeWhen": true,
            "visibleWhen": true
        }
    }
}
```

Add the following code in `commandsViewModel.json` under `commandPlacements` section.

```
{
    "commandPlacements": {
        "changeLocaleCmdPlacement": {
            "id": "ChangeLocaleCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "priority": 30
        },
        "EnglishLocaleCmdPlacement": {
            "id": "EnglishLocaleCmd",
            "uiAnchor": "",
            "parentGroupId": "ChangeLocaleCmd"
        },
        "GermanLocaleCmdPlacement": {
            "id": "GermanLocaleCmd",
            "uiAnchor": "",
            "parentGroupId": "ChangeLocaleCmd"
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

Add the following code in `commandsViewModel.json` under `i18n` section.

```
"i18n": {
    "changeLocale": [
        "helloModuleMessages"
    ],
    "englishLocale": [
        "helloModuleMessages"
    ],
    "germanLocale": [
        "helloModuleMessages"
    ]
}
```

**Note**: how we bind the command `title` with *i18n*, which can be changed between languages.

## Step 3: Add the Command Action

Add **action** to call JSFunction to automatically update the browser URL.

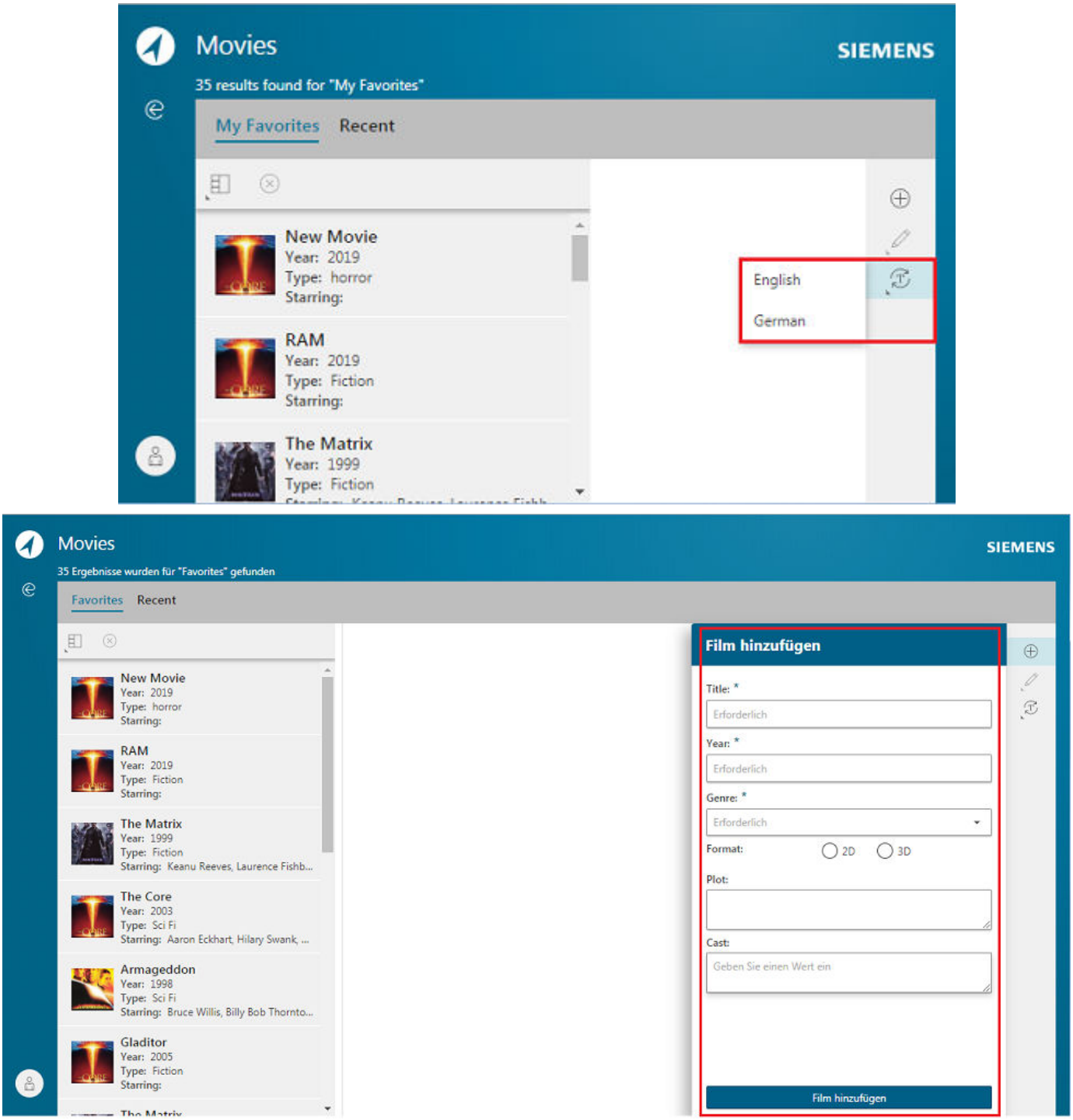
Add the following code in `commandsViewModel.json` under `actions` section.

```
"actions": {
   "changeToEnglishLocale": {
        "actionType": "JSFunction",
        "method": "updateBrowserUrl",
        "inputData": {
            "locale": "en_EN"
        },
        "deps": "js/browserUtils"
    },
    "changeToGermanLocale": {
        "actionType": "JSFunction",
        "method": "updateBrowserUrl",
        "inputData": {
            "locale": "de_DE"
        },
        "deps": "js/browserUtils"
    }
}
```

The locale is the language that you want to show.

**Note**

This will add localized text to `Add Movie` section.




## Step 4: Add localized text to location/sublocation.

In order to add localized text for location/sublocation, add two new files `locationAndSublocationMessages.json` and `locationAndSublocationMessages_de.json` under `i18n` folder peer to `helloModuleMessages.json`.

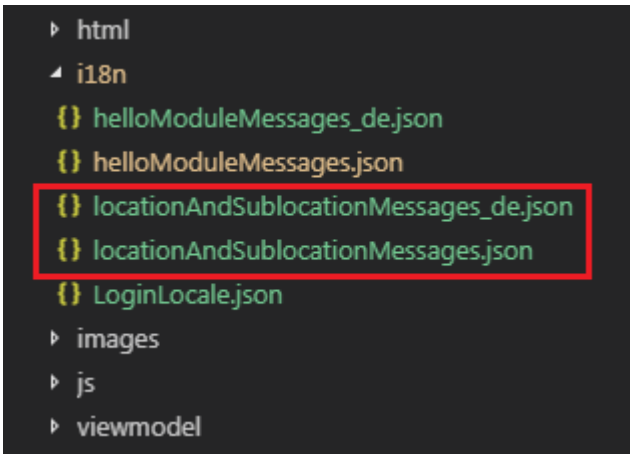

Siemens Web Framework
Unpublished work. © 2025 Siemens

Add the following content in `locationAndSublocationMessages.json`.

```
{
    "favoritesLocationTitle": "Movies",
    "favoritesSubLocationTitle": "My Favorites",
    "recentSubLocationTitle": "Recent"
}
```

Add the following content in `locationAndSublocationMessages_de.json`.

```
{
    "favoritesLocationTitle": "Filme",
    "favoritesSubLocationTitle": "Meine Favoriten",
    "recentSubLocationTitle": "Kürzlich"
}
```




To consume localized text, update `states.json` file under `swfTutorialMain` folder.

Update `movies` section of `states.json`.

```
{
    "data": {
        "headerTitle": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "favoritesLocationTitle"
        }
    }
}
```

```
"movies": {
    "abstract": true,
    "data": {
        "headerTitle": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "favoritesLocationTitle"
        }
    },
    "view": "commonLocation",
    "parent": "commonRoot"
},
```

Update `myFavs` section of `states.json`.

```
{
    "data": {
        "label": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "favoritesSubLocationTitle"
        }
    }
}
```

```
"myFavs": {
    "url": "/myFavs",
    "data": {
        "label": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "favoritesSubLocationTitle"
        },
        "nameToken": "com.siemens.splm.client.favorites:favoritesSubLocation",
        "clientScopeURI": "",
        "viewBase": "favorites"
    },
    "params": {
        "d_uids": null,
        "s_uid": null
    },
    "parent": "movies",
    "type": "subLocation"
```

Update `myRecent` section of `states.json`.

```
{
    "data": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "label": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "recentSubLocationTitle"
        }
    }
}
```

## Step 4: Build the Solution

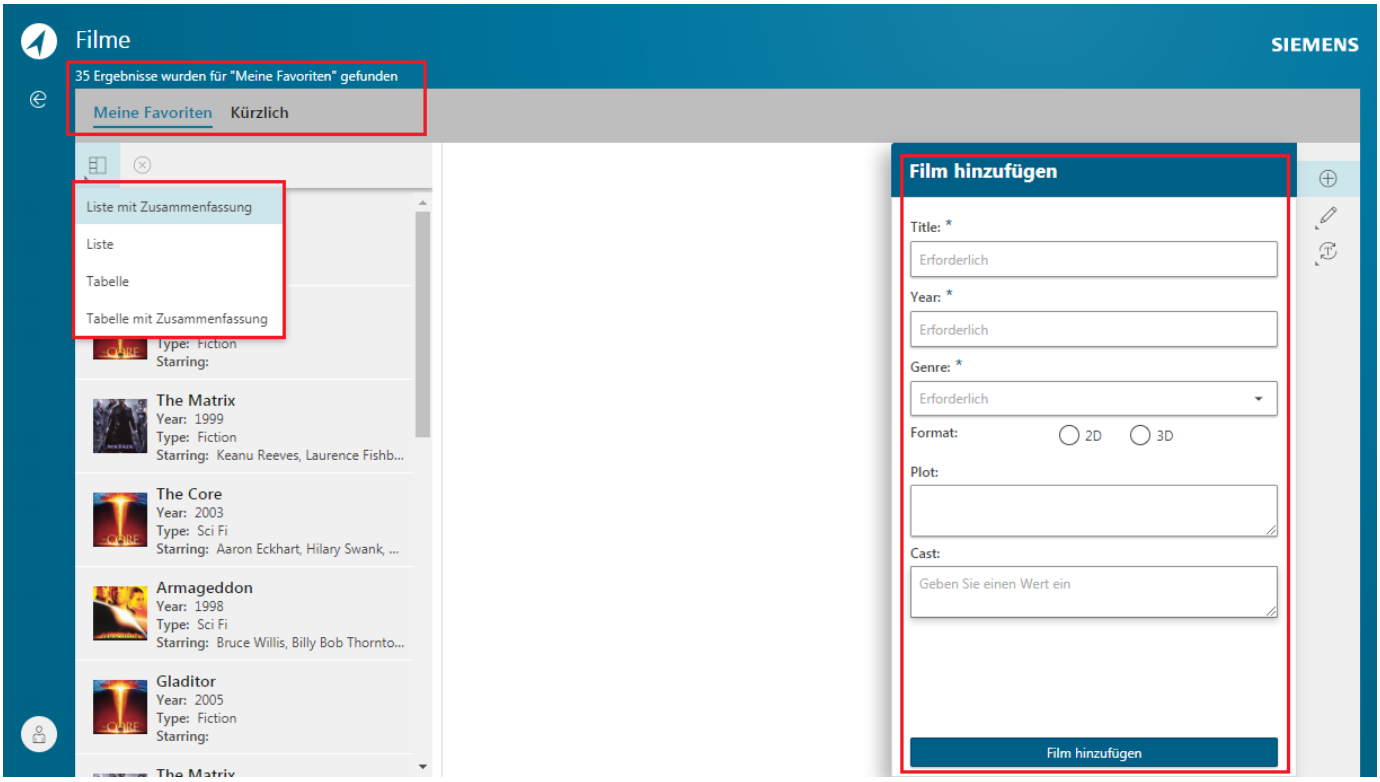
Run the build commands to build and deploy the site.

```
set SITE=site_tutorial
aw build site_tutorial
aw start
```

## Summary

In this tutorial, you learned how to:

- Provide localizations of user-facing text for your solution. This applies to all UI elements such as menu names, label name, tab names, property labels, and hover text.
- Provide a command for the user to change the language.




## References

- Commands
- View model data
- View model

# Working with tables

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from Support Center, as explained in the Tutorials Overview.

Learn how to use tables in a web application based on Siemens Web Framework. In **Working with Commands** tutorial, you learned how to add a display command which can be used to switch between different views. In this tutorial, we would use aw-splm-table to render a table in "Table" and "Table with Summary" display mode.


Unpublished work. © 2025 Siemens

## Objective

You will learn how to render a table in a declarative way.

## Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **Working with Commands** before proceeding.

**Otherwise:**

Download the source code for the **working_with_commands** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Add the table in the Table view.

Note that files `favoritesTableView.html` and `favoritesTableViewModel.json` were introduced during adding the various display modes like "Table" and "Table with Summary". As of now, when these views are selected, you see a placeholder text only. Lets add a table to this view.

```
<aw-flex-column>
    <aw-splm-table gridid="moviesTable"></aw-splm-table>
</aw-flex-column>
```

`aw-splm-table` is the declarative building block that is being used here. This table is provided with data using a data provider which needs to be defined in the view model. In a view, multiple tables may exist. Each data provider is thus identified using an unique gridid, that is "moviesTable" here.

## Step 2: Add the data provider to the view model.

1. Add the empty sections to favoritesTableViewModel.json file and add fields in subsequent steps

```
{
    "schemaVersion": "1.0.0",
    "imports": [],
    "data": {},
    "actions": {},
    "dataProviders": {},
    "columnProviders": {},
    "functions": {},
    "grids": {},
    "dataParseDefinitions": {},
```


Unpublished work. © 2025 Siemens

```
    "onEvent": [],
    "i18n": {},
    "messages": {},
    "conditions": {}
}
```

Note that several sections which are new here, like "columnProviders", "grids", etc. These are specific to tables. The "dataProviders" is responsible to provide the data(rows) to the table. The "columnProviders" are responsible for providing the properties(columns) to the table. The other sections are generic and has been used a lot till now.

2. Attach the gridid to the view model.

   The gridid "moviesTable" referred in view will be defined here. This implies that the column definition would be provided by "moviesColumnProvider" and the table row information would be provided by "moviesDataProvider". Visit aw-splm-table Storybook example to know more.

```
{
    "grids": {
        "moviesTable": {
            "dataProvider": "moviesDataProvider",
            "columnProvider": "moviesColumnProvider",
            "addIconColumn": true,
            "gridOptions": {
                "enablePinning": true,
                "enableSorting": false
            }
        }
    }
}
```

3. Define the column provider.

   Note that "moviesColumnProvider" holds the set of columns that needs to be displayed in the table. These columns have some properties by themselves, like "name", "displayName", "enableColumnMenu" and more which are configurable. "frozenColumnIndex" enables you to fix columns till the index specified. Try it yourself to see the behavior. Note, localization is in place for display names as well.

```
{
    "columnProviders": {
        "moviesColumnProvider": {
            "frozenColumnIndex": 1,
            "columns": [{
                    "name": "icon",
                    "displayName": "",
                    "maxWidth": 30,
                    "minWidth": 30,
                    "width": 30,
                    "enableColumnMenu": false,
                    "pinnedLeft": true
                },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                {
                    "name": "Title",
                    "displayName": "{{i18n.movieTitle}}",
                    "minWidth": 60,
                    "width": 150,
                    "enableColumnMenu": false,
                    "pinnedLeft": true
                },
                {
                    "name": "Year",
                    "displayName": "{{i18n.movieYear}}",
                    "minWidth": 60,
                    "width": 90,
                    "enableColumnMenu": false,
                    "pinnedLeft": true
                },
                {
                    "name": "Type",
                    "displayName": "{{i18n.movieType}}",
                    "minWidth": 80,
                    "width": 100,
                    "enableColumnMenu": false
                },
                {
                    "name": "Starring",
                    "displayName": "{{i18n.movieCast}}",
                    "minWidth": 150,
                    "width": 350,
                    "enableColumnMenu": false
                },
                {
                    "name": "Description",
                    "displayName": "{{i18n.moviePlot}}",
                    "minWidth": 150,
                    "width": 550,
                    "enableColumnMenu": false
                }
            ]
        }
    }
}
```

`Column Provider and its Actions` Columns can be either statically defined in the viewModel (as illustrated in this tutorial) or dynamically loaded by the server. If using a server to save column data, columns can be set dynamically as follows:

```
{
    "columnProviders": {
        "exampleColumnProvider": {
            "columns": "{{data.columns}}"
        }
    }
}
```

`loadColumnAction` Loads column information upon initial page load. This action is invoked when a table/tree is loaded for the first time. If columns are dynamically loaded from the server, this action must be configured to load the column data.

To illustrate, refer to the Storybook aw-splm-table tree example "Multiple Selection", which uses loadColumnAction to load dynamically defined columns.

4. Define the data provider.

   The data provider definition is same as fetching data for list view. The same set of data is being fetched and displayed in table instead of a list.

```
{
    "dataProviders": {
        "moviesDataProvider": {
            "action": "loadMovies",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            }
        }
    }
}
```

   Before rendering the view, the data provider is detected through the gridid, and the action is performed which makes the REST api call to get the data. Here, "loadMovies" action be triggered. The response from the REST call is set to "response" and the total number of movies fetched is captured in "totalFound". This would be later used for supporting pagination. For now, lets configure table with single selection.

   For more information on dataProviders, see **Data provider**

5. Add the action to get the table data.

```
{
    "actions": {
        "loadMovies": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/movies"
                }
            },
            "dataParsers": [{
                "id": "movieParser",
                "responseObjs": "data.search"
            }],
            "outputData": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "totalFound": "data.totalResults",
                "searchResults": "data.search"
            }
        }
    }
}
```

"startIndex" and "pageSize" is to support pagination. Note that the parser that will identify the column information and map each row to respective columns is "movieParser".

## Configuring the Data Provider for Tree

`initializeAction`: This action is invoked upon initial page load, to load the first page of data. For trees, configuration of this action is required to load the first page of tree data. If this action is not configured for tables and lists, initial page load will invoke the "action" method instead. Proper configuration of this action for trees requires an additional imput that is not required in tables and lists. 1. You must set "accessMode" in the "inputData" section of the dataProvider to "tree". This can be done statically (`"accessMode": "tree"`) or dynamically (`"accessMode":"{{view}}"`) 2. Note that to use tree, must also set attribute "use-tree" to true `<aw-splm-table use-tree="true">` To illustrate, refer to Storybook aw-splm-table tree example "Default".

`topNodeUID`: This is how the dataProvider determines which object should be set as the top or parent tree node. The top node is generally hidden from the user and is only used to access child nodes. This variable is set in the "inputData" object of the dataProvider. Refer to Storybook aw-splm-table tree example "Default" for working example of "topNodeUid".

```
{
    "inputData": {
        "accessMode": "tree",
        "topNodeUid": "top"
    }
}
```

"topNodeUid" is used by the dataProvider to set `treeLoadInput.parentNode`. This is the input parent node which needs to be expanded in the tree. Once defined on dataProvider.inputData.topNodeUid, it gets internally mapped to treeLoadInput.parentNode. For subsequent expansions, this will be mapped for the node on which the expansion operation has been triggered.

`treeLoadInput`: This is the input object that is created by the dataProvider for the tree. This is configured in the "inputData" object of the tree's actions (initializaAction, nextAction, etc.). Refer to Storybook aw-splm-table tree example "Multiple Selection" for a working example. To configure, set `"treeLoadInput": "request.treeLoadInput"`, as seen below.

```
{
    "loadData": {
        "inputData": {
            "treeLoadInput": "request.treeLoadInput"
        }
```

```
    }
}
```

`treeLoadResult`: This is the output object created by the dataProvider for the tree. The most common configuration of this artifact is `"action.result.treeLoadResult"`. For a working example, refer to Storybook "Multiple Selection" example.

```
{
    "exampleDataProvider": {
        "treeLoadResult": "action.result.treeLoadResult"
    },
}
```

`propertyProvider`: This is required for trees that need to load data in two server calls, the first to load the tree nodes and the second to load the properties of that node. "propertyProvider" requires configuration of "action" and "propertyLoad" artifacts. Refer to Storybook aw-splm-tree example "Multiple Selection" for a working example.

```
{
    "dataProviders": {
        "exampleDataProvider": {
        },
        "examplePropertyProvider": {
            "action": "loadProperties",
            "propertyLoadResult": "action.result.propertyLoadResult"
        }
    },
}
```

1. Add the definition of data parser.

```
{
    "dataParseDefinitions": {
        "movieParser": {
            "identifier": "{{response.id}}",
            "typeIconURL": "{{response.Poster}}",
            "type": "Movie",
            "props": {
                "Title": {
                    "name": "Title",
                    "displayName": "{{i18n.movieTitle}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Year": {
                    "name": "Year",
                    "displayName": "{{i18n.movieYear}}",
                    "value": "{{response.Year}}",
                    "displayValue": "{{response.Year}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                    "type": "String"
                },
                "Type": {
                    "name": "Type",
                    "displayName": "{{i18n.movieType}}",
                    "value": "{{response.Type}}",
                    "displayValue": "{{response.Type}}",
                    "type": "String"
                },
                "Starring": {
                    "name": "Starring",
                    "displayName": "{{i18n.movieCast}}",
                    "value": "{{response.Starring}}",
                    "displayValue": "{{response.Starring}}",
                    "type": "String"
                },
                "Description": {
                    "name": "Description",
                    "displayName": "{{i18n.Description}}",
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                },
                "object_string":{
                    "displayName": "{{response.Title}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                }
            }
        }
    }
}
```

Refer to **data parse definitions** for knowing how it works.

2. Finally add the localization for the column names.

```
{
    "i18n": {
        "movieTitle": [
            "helloModuleMessages"
        ],
        "movieType": [
            "helloModuleMessages"
        ],
        "movieFormat": [
            "helloModuleMessages"
        ],
        "Description": [
            "helloModuleMessages"
        ],
        "movieYear": [
            "helloModuleMessages"
```

```
        ],
        "movieCast": [
            "helloModuleMessages"
        ]
    }
}
```

The enties must already be present in the `helloModuleMessages.json` file.

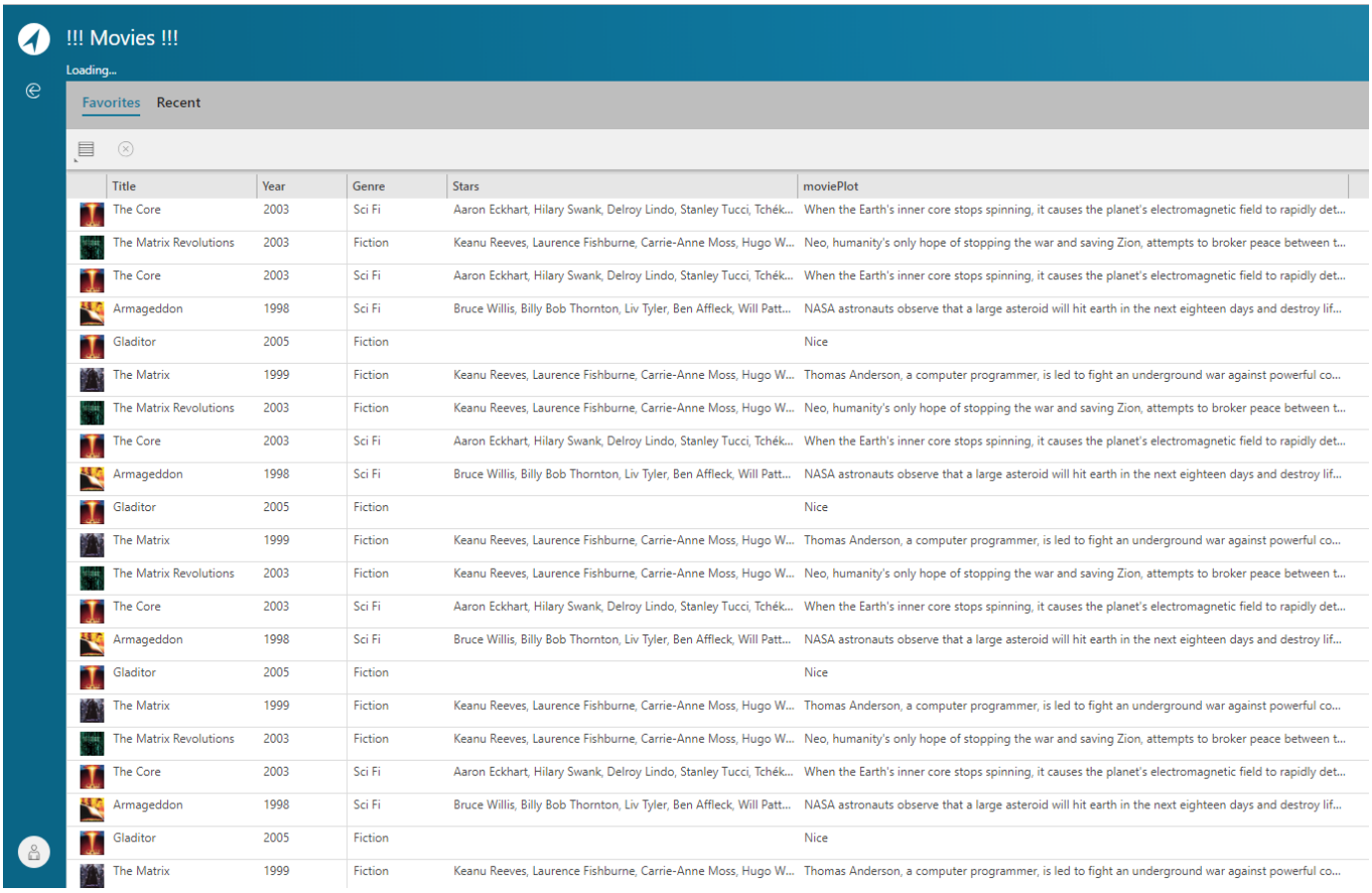
```
{
    "movieTitle": "Title",
    "movieType": "Genre",
    "movieFormat": "Format",
    "movieCast": "Stars",
    "movieYear": "Year",
    "Description": "Plot"
}
```

## Step 3: Build the Solution

Run the following build commands to build and deploy the site.

```
set SITE=site_tutorial
aw start
```


Siemens Web Framework
Unpublished work. © 2025 Siemens




| Title | Year | Genre | Stars | moviePlot |
|---|---|---|---|---|
| The Core | 2003 | Sci Fi | Aaron Eckhart, Hilary Swank, Delroy Lindo, Stanley Tucci, Tchék... | When the Earth's inner core stops spinning, it causes the planet's electromagnetic field to rapidly det... |
| The Matrix Revolutions | 2003 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Neo, humanity's only hope of stopping the war and saving Zion, attempts to broker peace between t... |
| The Core | 2003 | Sci Fi | Aaron Eckhart, Hilary Swank, Delroy Lindo, Stanley Tucci, Tchék... | When the Earth's inner core stops spinning, it causes the planet's electromagnetic field to rapidly det... |
| Armageddon | 1998 | Sci Fi | Bruce Willis, Billy Bob Thornton, Liv Tyler, Ben Affleck, Will Patt... | NASA astronauts observe that a large asteroid will hit earth in the next eighteen days and destroy lif... |
| Gladitor | 2005 | Fiction | | Nice |
| The Matrix | 1999 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Thomas Anderson, a computer programmer, is led to fight an underground war against powerful co... |
| The Matrix Revolutions | 2003 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Neo, humanity's only hope of stopping the war and saving Zion, attempts to broker peace between t... |
| The Core | 2003 | Sci Fi | Aaron Eckhart, Hilary Swank, Delroy Lindo, Stanley Tucci, Tchék... | When the Earth's inner core stops spinning, it causes the planet's electromagnetic field to rapidly det... |
| Armageddon | 1998 | Sci Fi | Bruce Willis, Billy Bob Thornton, Liv Tyler, Ben Affleck, Will Patt... | NASA astronauts observe that a large asteroid will hit earth in the next eighteen days and destroy lif... |
| Gladitor | 2005 | Fiction | | Nice |
| The Matrix | 1999 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Thomas Anderson, a computer programmer, is led to fight an underground war against powerful co... |
| The Matrix Revolutions | 2003 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Neo, humanity's only hope of stopping the war and saving Zion, attempts to broker peace between t... |
| The Core | 2003 | Sci Fi | Aaron Eckhart, Hilary Swank, Delroy Lindo, Stanley Tucci, Tchék... | When the Earth's inner core stops spinning, it causes the planet's electromagnetic field to rapidly det... |
| Armageddon | 1998 | Sci Fi | Bruce Willis, Billy Bob Thornton, Liv Tyler, Ben Affleck, Will Patt... | NASA astronauts observe that a large asteroid will hit earth in the next eighteen days and destroy lif... |
| Gladitor | 2005 | Fiction | | Nice |
| The Matrix | 1999 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Thomas Anderson, a computer programmer, is led to fight an underground war against powerful co... |
| The Matrix Revolutions | 2003 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Neo, humanity's only hope of stopping the war and saving Zion, attempts to broker peace between t... |
| The Core | 2003 | Sci Fi | Aaron Eckhart, Hilary Swank, Delroy Lindo, Stanley Tucci, Tchék... | When the Earth's inner core stops spinning, it causes the planet's electromagnetic field to rapidly det... |
| Armageddon | 1998 | Sci Fi | Bruce Willis, Billy Bob Thornton, Liv Tyler, Ben Affleck, Will Patt... | NASA astronauts observe that a large asteroid will hit earth in the next eighteen days and destroy lif... |
| Gladitor | 2005 | Fiction | | Nice |
| The Matrix | 1999 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Thomas Anderson, a computer programmer, is led to fight an underground war against powerful co... |
| The Matrix Revolutions | 2003 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Neo, humanity's only hope of stopping the war and saving Zion, attempts to broker peace between t... |
| The Core | 2003 | Sci Fi | Aaron Eckhart, Hilary Swank, Delroy Lindo, Stanley Tucci, Tchék... | When the Earth's inner core stops spinning, it causes the planet's electromagnetic field to rapidly det... |
| Armageddon | 1998 | Sci Fi | Bruce Willis, Billy Bob Thornton, Liv Tyler, Ben Affleck, Will Patt... | NASA astronauts observe that a large asteroid will hit earth in the next eighteen days and destroy lif... |
| Gladitor | 2005 | Fiction | | Nice |
| The Matrix | 1999 | Fiction | Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo W... | Thomas Anderson, a computer programmer, is led to fight an underground war against powerful co... |

## Step 4: Support pagination in table.

1. Add the following content in the `favoritesTableViewModel.json` file to support pagination.

Update the action "loadMovies" with two parameters like "startIndex" and "pageSize" in the request. This helps the view model to keep a track of how many pages of data is loaded and whats the starting index of the next page. Thus, pagination is table gets enabled using these parameters.

```
{
    "actions": {
        "loadMovies": {
            "inputData": {
                "request": {
                    "params": {
                        "startIndex":
"{{data.dataProviders.moviesDataProvider.startIndex}}",
                        "pageSize": "{{data.pageSize}}"
                    }
                }
            }
        }
```

```
    }
}
```

You may want to hard code "pageSize" here or add it as a constant in "data" section, as done here.

```
{
    "data": {
        "pageSize": 15
    }
}
```

## Step 5: Build the Solution again

Run the build commands to build and deploy the site. Check how pagination works. Change the page size and play with it.

## Summary

In this tutorial, you learned how to:

- Use aw-splm-table in a declarative way.
- Configure different parameters of the table in view model.
- Support pagination.

## Advanced Configurations - Add more to your table

## Custom Header Cells and Columns

PLTable supports three ways to configure custom header cell and column templates. These are useful if you want some custom cell display, such as icons, vertical text, etc. This template mechanism is built on top of Lodash's template, so it supports all the functionality that Lodash provides*. **https://lodash.com/docs/4.17.15#template** *(Lodash Version 4.17.15)

## Step 1: Create propertyRendererTemplates.json file.

To use any custom property renderer templates, you must first create the propertyRendererTemplates.json file. This will be a sibling to the commandsViewModel.json and module.json files.

Add below json to your propertyRendererTemplates.json file. We will configure the artifacts in later steps.

```
{
    "custom_header_template": {
        "grids": [],
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "columns": [],
        "tooltip": [],
        "template": "",
        "templateUrl": "",
        "renderFunction": "",
        "deps": "",
        "dependentServices": [],
        "headerTemplate": "",
        "headerTemplateUrl": "",
        "headerRenderFunction": ""
    },
}
```

## Step 2: Define the grids and columns

Configure `"grids"` and/or `"columns"`. These will refer to the gridids and names of applicable columns. There are three options here: 1. Define the grids to which you want your template to apply. All columns in that grid will then be defined by this template. 2. Define the columns to which you want your template to apply. The template will apply to all the tables where the columns field is matched. 3. Define both grids and columns. The template will be applied only to the specified columns within the defined table/tables.

```
{
    "custom_header_template": {
        ...,
        "grids": ["propertyRenderHeaderTemplateTable"],
        "columns": [ "nation" ],
        ...
    },
}
```

## Step 3: Define the custom header property renderer

PLTable provides three different template options to meet customization needs. These can be used to customize column headers and column cells.

## Option 1: Template

To use the template option, define the custom HTML template directly in the propertyRendererTemplates.json file.

For custom header cells, configure the "headerTemplate" artifact. For custom column cells, configure the "template" artifact.

See the Storybook example "Editable table with various property types". This example defines custom column cells for the "Country Name" column. The following files are part of Storybook example.

```
{
    "editable_table_country_name_link": {
        "grids": ["exampleEditTable"],
        "columns": [ "name" ],
        "template": "<a href='http://en.wikipedia.org/wiki/
<%=vmo.props.name.uiValue%>'><%=vmo.props.name.uiValue%></a>",
    }
}
```

## Option 2: Template URL

To use the template URL option, create a separate HTML where the custom template is defined, and point "headerTemplateUrl"/"templateUrl" artifact to that custom HTML file. Refer to Storybook example, "Property Rendering Header Template", which is using a simple switch statement to render a custom icon for each column.

```
{
    "custom_header_template_url": {
        "grids": ["propertyRenderHeaderTemplateTable"],
        "columns": [ "played", "drawn", "won", "lost" ],
        "headerTemplateUrl": "/propRenderTemplates/headerRenderer.html",
    }
}
```

## Option 3: Render Function

While this option gives the user the most control over customization, it should be the last option considered. It is recommended to use render function only in cases where template and template URL do not meet your customization needs. It is also cautioned that the custom JavaScript implemented for render function should not contain any dependency on AngularJS. To use the render function: 1. Add service file as a dependency in the "deps" artifact. 2. Create custom render function in the service file, and refer to this function in "headerRenderFunction" artifact. Refer to Storybook example, "Property Rendering Header Template" for a working example of render function implementation.

```
{
    "custom_header_render_function": {
        "grids": ["propertyRenderHeaderTemplateTable"],
        "columns": [ "position" ],
        "deps": "js/splmTablePropRenderingHeaderTemplateService",
        "headerRenderFunction": "showcaseCustomHeaderRender"
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Support for Sorting

Sorting enablement relies upon the gridOption "enableSorting". This is set to true by default. Set the column attribute "sortCriteria" with `"sortCriteria": []`.

It is recommended that the sorting logic be implemented on the server instead of the client.

To disable sorting, set gridOption "enableSorting" to false:

```
{
    "grids": {
        "exampleTable": {
            ...,
            "gridOptions": {
                ...,
                "enableSorting": false,
                ...
            }
        }
    },
}
```

When sorting is enabled at the table level, sorting can also be controlled at the column level through the column attribute "enableSorting". This value is true by default when sorting is enabled for that table. To disable sorting on specific columns, set column attribute "enableSorting" to false:

```
{
    "columnProviders": {
        "exampleColumnProvider": {
            "columns": [
                {
                    "name": "exampleColumn",
                    "enableSorting": false
                }
            ]
        }
    },
}
```

## Enable Column Filtering

To support column filtering with a non-TC server:

1. Set the gridOption "isFilteringEnabled": true
2. Pass the filter values from the user input to the server. This input can be data bound to the filter values from the column provider
    - "{

}" To support facet filters, continue to the third step.

3. Configure the required artifacts in the Data Provider `filterFacetAction`: This action will be invoked each time the user applies or removes some filter. "facetFilterAction" needs to return 2 values:
   a. filterResults: an object with display values of the table column
   b. totalFound: the length of the dataset array `filterFacetResults`: This is a reference to the outputData of the filterFacetAction's results

Refer to the Storybook example "Column Filters with Sort and Pagination" for illustration.

### Enable Editing

See the **Edit in Table** tutorial.

## Editing data in a table

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**. This tutorial demonstrates how to provide edit functionality on table (`aw-splm-table`).

Here, we would define an edit handler on a data provider and use it to edit a table. We are also showing how to leverage the server, where-in the server logic decides which cells in the table are editable, to move a table to edit mode.

### Objective

You will learn how to:

- Use the editing capabilities to edit a PL table.

### Pre-requisites

You must have already gone through **Pre-requisite software** by now.

**Recommended:**

Complete **Working with Tables** before proceeding.

**Otherwise:**


Siemens Web Framework
Unpublished work. © 2025 Siemens

Download the source code for the **working_with_table** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Guideline

### Start Editing Configuration

1. To start the editing, create a command that calls an action (e.g. `startEditMovie` in `commandsViewModel.json`) that fires an event (e.g. `movie.startEdit`).
2. In your table viewModel, subscribe to this event and call a dataProvider action (e.g. `startMovieTableEdit`). The purpose of this action is to notify the dataProvider that editing is starting.
3. The dataProvider automatically calls the `startEditAction` configured in its `editConfig` (e.g. `MovieTableStartEditAction`). The purpose of this action is to return an array of objects containing information on which cells will be editable.

### Save Editing Configuration

1. To save edited data, create a command that calls an action (e.g. `saveEditMovie` in `commandsViewModel.json`) that fires an event (e.g. `movie.saveEdit`).
2. In your table viewModel, subscribe to this event and call a dataProvider action (e.g. `saveMovieTable`). The purpose of this action is to notify the dataProvider that we want to save edited data.
3. The dataProvider automatically calls the `saveEditAction` configured in its `editConfig` (e.g. `MovieTableSaveEditAction`). This action should contain whatever logic you need to persist your changes.

### Cancel Editing Configuration

1. To cancel the editing, create a command that calls an action (e.g. `cancelEditMovie` in `commandsViewModel.json`) that fires an event (e.g. `movie.cancelEdit`).
2. In your table viewModel, subscribe to this event and call a dataProvider action (e.g. `cancelMovieTableEdit`). The purpose of this action is to notify the dataProvider that editing is is being cancelled.

### `editHandlerStateChange event management (only for aw-splm-table)`

Create a subscription to the *.editInProgress* event (e.g. `moviesDataProvider.editInProgress`), and call an action (e.g. `editEventAction`) that publishes the `editHandlerStateChange` events with the appropriate configuration. This event is used internally by `aw-splm-table` to toggle the editing state of the cells.

## Step: 1

**Note**

If you want to the traces from debug, set DEBUG=app from command prompt.

Add the following methods in `server\server.js` which saves the edited value from UI

```
let startEditOpr = function( editInputs ) {
    let output = [];
    _.forEach( editInputs, ( input ) => {
        let inputObj = {};
        inputObj.uid = input.uid;
        inputObj.props = {};
        let propNames = input.propertyNames;
        _.forEach( propNames, ( propName ) => {
            if( propName === 'Title' ) {
                inputObj.props.title = {
                    isPropertyModifiable: false,
                    propertyName: 'Title'
                };
            } else if( propName === 'Year' ) {
                inputObj.props.year = {
                    isPropertyModifiable: true,
                    propertyName: 'Year'
                };
            } else if( propName === 'Type' ) {
                inputObj.props.type = {
                    isPropertyModifiable: true,
                    propertyName: 'Type'
                };
            } else if( propName === 'Starring' ) {
                inputObj.props.starring = {
                    isPropertyModifiable: false,
                    propertyName: 'Starring'
                };
            } else if( propName === 'Description' ) {
                inputObj.props.description = {
                    isPropertyModifiable: true,
                    propertyName: 'Description'
                };
            }
        } );
        output.push( inputObj );
    } );

    return output;
};
```

```
app.post( '/afx/movies/edit', function( req, res ) {
    const dataFile = path.join( __dirname, 'data/listresponse.json' );
    const jsonResponse = require( MOVIES_PATH );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    const operation = req.query.operation;

    let output = [];
    if( operation === 'startEdit' ) {
        const editInputs = req.body.editInputs;
        output = startEditOpr( editInputs );
    } else if( operation === 'saveEdit' ) {
        const saveInputs = req.body.saveInputs;
        if( saveInputs && saveInputs.length > 0 ) {
            _.forEach( saveInputs, ( modifiedObject ) => {
                let uid = modifiedObject.identifier;
                let props = modifiedObject.props;
                if( jsonResponse && uid ) {
                    let foundMovie = _.find( jsonResponse, ( o ) =>
{ return o.id === uid; } );
                    if( foundMovie ) {
                        let updates = {};
                        _.forEach( props, ( prop ) => {
                            let dbValue = prop.dbValues.filter( value => {
                                return  value !== undefined && value !==
'undefined';
                            } );
                            updates[ prop.propertyName ] = dbValue;
                        } );
                        _.assign( foundMovie, updates );
                    }
                }
            } );
            return fs.writeFile( dataFile, JSON.stringify( jsonResponse ),
( err ) => {
                if( err ) {
                    debug( 'File Write error' );
                    return res.send( 'Error' );
                }
                let response = { success: 'true' };
                return res.send( response );
            } );
        }
    }
    debug( `${operation}` );
    return res.send( output );
} );
```

## Step 2:

Update `src\swfTutorialMain\src\viewmodel\favoritesTableViewModel.json`. Here we would add the actions corresponding to the "Start Edit", "Save Edit" and "Cancel Edit" commands for editing the table. We would not use the Edit commands of Summary, which appears on the right wall.

```
{
    "actions": {
        "startMovieTableEdit": {
```

```
            "actionType": "dataProvider",
            "method": "moviesDataProvider",
            "inputData": {
                "action": "startEdit"
            }
        },
        "cancelMovieTableEdit": {
            "actionType": "dataProvider",
            "method": "moviesDataProvider",
            "inputData": {
                "action": "cancelEdits"
            }
        },
        "saveMovieTable": {
            "actionType": "dataProvider",
            "method": "moviesDataProvider",
            "inputData": {
                "action": "saveEdits"
            }
        }
    }
}
```

## Step 3:

Start with providing the edit configuration `editConfig` inside the data provider associated with the table. The changes are to be done in `favoritesTableViewModel.json`. This configuration is used by the data provider at different stages.

```
{
    "dataProviders": {
        "moviesDataProvider": {
            "action": "loadMovies",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            },
            "editConfig": {
                "startEditAction": "MovieTableStartEditAction",
                "saveEditAction": "MovieTableSaveEditAction",
                "identifier": "uid"
            }
        }
    }
}
```

In Edit configuration, the user needs to specify the following mandatory configurations.


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. `startEditAction`: The user can configure the custom action required to start an edit action based on the server response.
2. `saveEditAction`: The user can configure the custom action required to save the edited value at server side.
3. `Identifier` is required to uniquely identify individual objects. This is required for save action to save the edited properties against some object with a unique identifier.

Now the custom actions required to configure edit is defined in "actions" section of `favoritesTableViewModel.json`.

```
{
    "actions": {
        "MovieTableStartEditAction": {
            "actionType": "RESTService",
            "method": "POST",
            "inputData": {
                "request": {
                    "method": "POST",
                    "url": "/afx/movies/edit",
                    "params": {
                        "operation": "startEdit"
                    }
                }
            }
        },
        "MovieTableSaveEditAction":{
            "actionType": "RESTService",
            "method": "POST",
            "inputData": {
                "request": {
                    "method": "POST",
                    "url": "/afx/movies/edit",
                    "params": {
                        "operation": "saveEdit"
                    }
                }
            }
        }
    }
}
```

## Step 4:

Update `favoritesTableViewModel.json` with the following entry to register the events on edit commands and perform particular actions .

```
{
    "onEvent": [
        {
```

```
            "eventId": "movie.startEditTable",
            "action": "startMovieTableEdit"
        },
        {
            "eventId": "movie.saveEditTable",
            "action": "saveMovieTable"
        },
        {
            "eventId": "movie.cancelEditTable",
            "action": "cancelMovieTableEdit"
        },
        {
            "eventId": "moviesDataProvider.editInProgress",
            "cacheEventData": true,
            "action": "editEventAction"
        }
    ]
}
```

movie.startEditTable, movie.saveEditTable and movie.cancelEditTable are the events that are fired by the **Start Edit**, **Save Edit**, and **Cancel Edit** commands for table. It could be any event name. It is not necessary to use the event names used in this tutorial. These events are fired from the commandsViewModel.json which would be defined in the following section.

SPLM table will go into edit mode if the *editHandlerStateChange* event is published with eventData that has a *state* of *starting* and *dataSource* matching/containing the tables dataProvider. So the `editEventAction` does the same to make the PL table editable.

Update `favoritesTableViewModel.json` with the following.

```
{
    "actions": {
        "editEventAction": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "editHandlerStateChange",
                    "eventData": {
                        "state": "{{data.eventData === true ? 'starting' :
'saved'}}",
                        "dataSource":
"{{data.dataProviders.moviesDataProvider}}"
                    }
                } ]
            }
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 5:

We would now be adding a new set of table edit commands in `commandsViewModel.json`. For that first let us deactivate the existing edit commands.

For defining the handlers for the new table commands, we would first deactivate the existing edit command. This will reduce confusion in the UI.

Update the existing edit command handler definitions for `startEditMovieCommandHandler` to set the visibility of the commands conditionally.

```
{
    "commandHandlers": {
         "startEditMovieCommandHandler": {
            "visibleWhen": {
                "condition": "conditions.EnableStartEdit"
            }
        }
    }
}
```

Similarly update the existing edit command handler definitions for `saveEditMovieCommandHandler` and `cancelEditMovieCommandHandler` to set the visibility of the commands conditionally.

```
{
    "commandHandlers": {
         "saveEditMovieCommandHandler": {
            "visibleWhen": {
                "condition": "conditions.EnableSaveCancelEdit"
            }
        }
    }
}
```

To add to this, let us also update these conditions accordingly, please understand the additional condition added here is that table view is not active.

```
{
     "conditions": {
        "isTableView": {
            "expression": "ctx.ViewModeContext.ViewModeContext ===
'TableView'"
        },
        "SingleSelection": {
            "expression": "ctx.mselected.length === 1 && !
conditions.isTableView"
        },
        "EnableStartEdit": {
            "expression": "ctx.mselected.length === 1 && !
```


Unpublished work. © 2025 Siemens

```
conditions.isTableView && ctx.editInProgress === false"
        },
        "EnableSaveCancelEdit": {
            "expression": "ctx.editInProgress === true && !
conditions.isTableView"
        },

     }
}
```

## Step 6:

Open the `commandsViewModel.json`, peer to `module.json` and add the following definitions to contribute the **Start Edit**, **Save Edit** and **Cancel Edit** commands for editing table. All these commands would be grouped under **Edit** command.

Defining the commands separately for editing table:

```
{
    "commands": {
        "EditMovieTableGroupCmd": {
            "iconId": "cmdEdit",
            "isGroup": true,
            "title": "{{i18n.editTooltip}}"
        },
        "Awp0StartEditTable": {
            "iconId": "cmdEdit",
            "title": "{{i18n.startEditTooltip}}"
        },
        "Awp0SaveEditTable": {
            "iconId": "cmdSave",
            "title": "{{i18n.saveEditTooltip}}"
        },
        "Awp0CancelEditTable": {
            "iconId": "cmdCancel",
            "title": "{{i18n.cancelEditTooltip}}"
        }
    }
}
```

Defining the respective command handlers:

```
{
    "commandHandlers": {
        "Awp0StartEditTableCommandHandler": {
            "id": "Awp0StartEditTable",
            "action": "startEditMovieTable",
            "activeWhen": {
                "condition": "conditions.isMovieTableInNonEdit"
            },
            "visibleWhen": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "condition": "conditions.isMovieTableInNonEdit"
            }
        },
        "Awp0SaveEditTableCommandHandler": {
            "id": "Awp0SaveEditTable",
            "action": "saveEditMovieTable",
            "activeWhen": {
                "condition": "conditions.isMovieTableInEdit"
            },
            "visibleWhen": {
                "condition": "conditions.isMovieTableInEdit"
            }
        },
        "Awp0CancelEditTableCommandHandler": {
            "id": "Awp0CancelEditTable",
            "action": "cancelEditMovieTable",
            "activeWhen": {
                "condition": "conditions.isMovieTableInEdit"
            },
            "visibleWhen": {
                "condition": "conditions.isMovieTableInEdit"
            }
        }
    }
}
```

As used in the handlers, let us add the new condition mentioned. These condition would evaluate to true/false if the table view is currently active and edit is in progress.

```
{
     "conditions": {
        "isMovieTableInEdit": {
            "expression": "conditions.isTableView &&
ctx.moviesDataProvider_editInProgress === true"
        },
        "isMovieTableInNonEdit": {
            "expression": "conditions.isTableView &&
(ctx.moviesDataProvider_editInProgress === undefined ||
ctx.moviesDataProvider_editInProgress === false)"
        }
     }
}
```

Defining the command position/placement, through an anchor

```
{
    "commandPlacements": {
        "Awp0EditTableGroupCmdPlacement": {
            "id": "EditMovieTableGroupCmd",
            "uiAnchor": "aw_workArea_right",
            "priority": 25
        },
```

```
        "Awp0StartEditTableCmdPlacement": {
            "id": "Awp0StartEditTable",
            "uiAnchor": "aw_workArea_right",
            "parentGroupId": "EditMovieTableGroupCmd",
            "priority": 30
        },
        "Awp0SaveEditTableCmdPlacement": {
            "id": "Awp0SaveEditTable",
            "uiAnchor": "aw_workArea_right",
            "parentGroupId": "EditMovieTableGroupCmd",
            "priority": 30
        },
        "Awp0CancelEditTableCmdPlacement": {
            "id": "Awp0CancelEditTable",
            "uiAnchor": "aw_workArea_right",
            "parentGroupId": "EditMovieTableGroupCmd",
            "priority": 30
        }
    }
}
```

Defining the actions to be performed when the command is triggered:

```
{
    "actions": {
        "startEditMovieTable": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "movie.startEditTable",
                    "excludeLocalDataCtx": true
                } ]
            }
        },
        "saveEditMovieTable": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "movie.saveEditTable",
                    "excludeLocalDataCtx": true
                } ]
            }
        },
        "cancelEditMovieTable": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "movie.cancelEditTable",
                    "excludeLocalDataCtx": true
                } ]
            }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        }
    }
}
```

In the above code snippet, three different commands are registered to perform edit actions on the table.

1. **Start Edit** - fires movie.startEditTable event.
2. **Cancel Edits** - fires movie.cancelEditTable event.
3. **Save Edits** - fires movie.saveEditTable event.

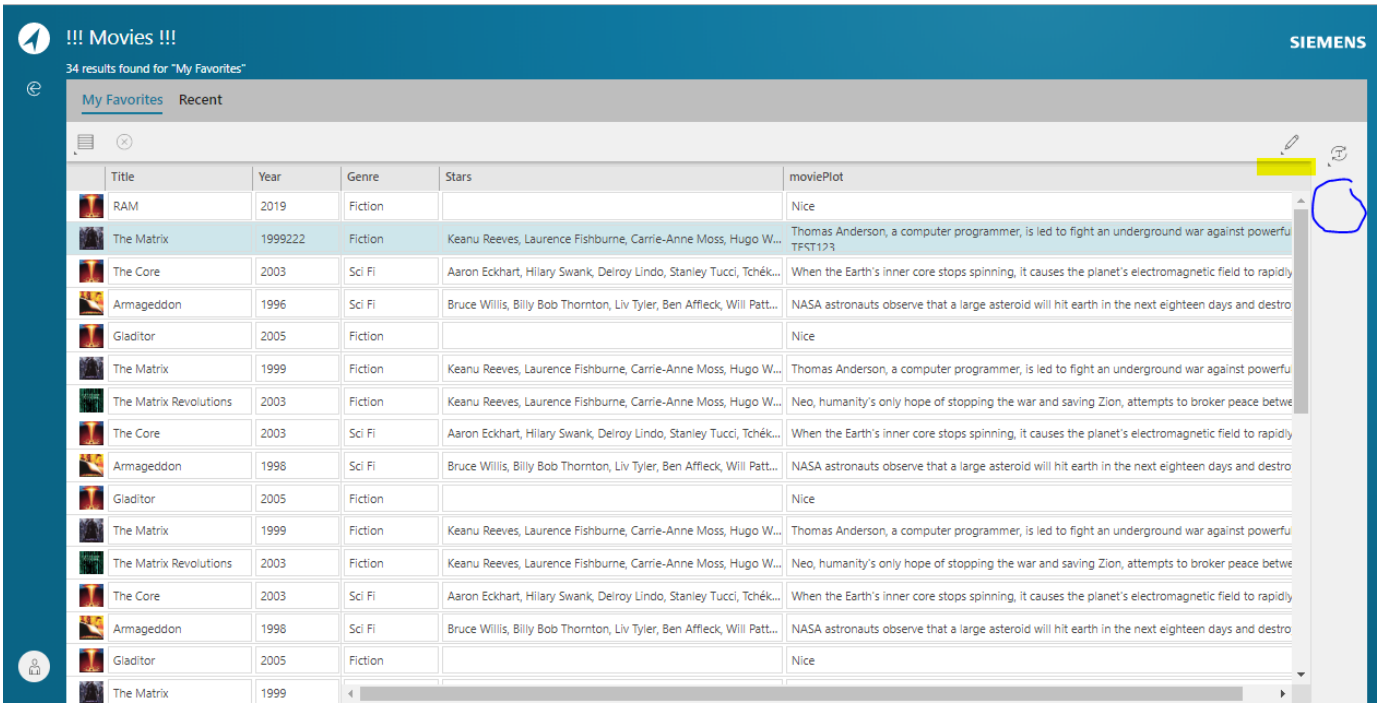
These events are registered in the Step 4 in `favoritesTableViewModel.json`.

## Step 6: Build the Solution

Run the build commands to build and deploy the site.

```
set SITE=site_tutorial
aw build site_tutorial
aw start
```

Launch the application to see this:




Note that the Edit commands on the right wall is no longer visible, while new set of commands are appearing on the top.

### Summary

In this tutorial, you learned how to configure editing in a table.

## Selection summary with table

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from **Support Center**, as explained in the **Tutorials Overview**.

### Objective

You will learn how to configure a table in the selection summary to display in the Secondary Work Area.

### Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **Selection Summary** before proceeding.

**Otherwise:**

Download the source code for the **selection_summary** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

### Step 1: Setup the backend server for table response.

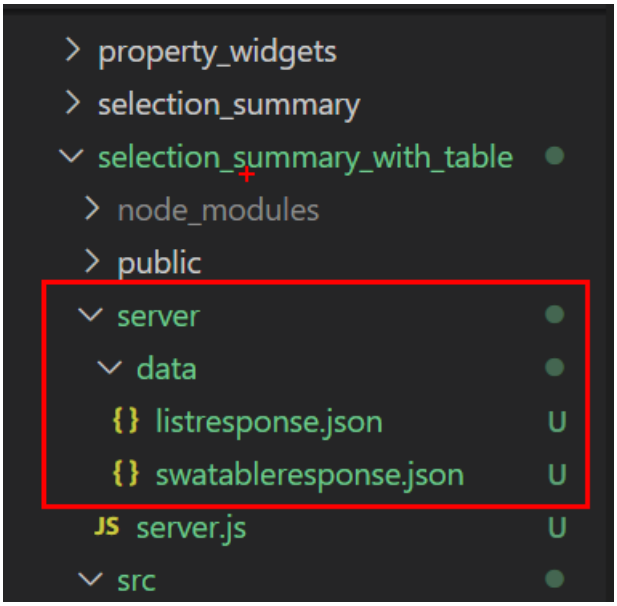
Download the source code for the **selection_summary_with_table** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

**Tip**

Note that this folder contains the source code of the *completed* tutorial, so unzip it outside your current project folder if you are planning to follow along with the rest of the tutorial.

Inside the **selection_summary_with_table** folder, copy the **server** folder to the root folder of your project.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Step 2: Configure table in secondary work area.

Lets create two files named `MovieInfoView.html` and `MovieInfoViewModel.json` in `html` and `viewmodel` folders in `src` directory of `swfTutorialMain`.

Add the following code snippets for each file:

`MovieInfoView.html` - Use table with gridID defined in `MovieInfoViewModel.json`.

```
<aw-panel-section caption='Movie Info'>
    <aw-panel-body>
        <aw-splm-table gridid="movieTableGrid"></aw-splm-table>
    </aw-panel-body>
</aw-panel-section>
```

`MovieInfoViewModel.json` - On selecting an object in list, action "getSelectedObject" is performed.

```
{
    "schemaVersion": "1.0.0",
    "grids": {
        "movieTableGrid": {
            "dataProvider": "movieDataProvider",
            "columnProvider": "movieColumnProvider",
            "addIconColumn": false,
            "gridOptions": {
                "enableGridMenu": false,
                "enablePinning": false,
                "enableSorting": false
            }
        }
```

```
    },
    "dataProviders": {
        "movieDataProvider": {
            "action": "getSelectedObject",
            "response": "{{data.results}}",
            "totalFound": "{{data.totalFound}}",
            "inputData": {
                "selectionModel": "{{props.context.selectionModel}}"
            },
            "selectionModelMode": "single"
        }
    },
    "actions": {
        "loadMovieData": {
            "actionType": "dataProvider",
            "method": "movieDataProvider"
        },
        "getSelectedObject": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/swamovies",
                    "params": {
                        "movieName": "{{props.context.selection[0].uid}}"
                    }
                }
            },
            "outputData": {
                "results": "data",
                "totalFound": "data.length"
            }
        }
    },
    "columnProviders": {
        "movieColumnProvider": {
            "columns": [ {
                    "name": "name",
                    "displayName": "Name",
                    "minWidth": 150,
                    "width": 150
                },
                {
                    "name": "review",
                    "displayName": "Review",
                    "minWidth": 300,
                    "width": 300
                },
                {
                    "name": "rating",
                    "displayName": "Rating",
                    "minWidth": 350,
                    "width": 350
                }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            ],
            "frozenColumnIndex": -1
        }
    },
    "lifecycleHooks": {
        "onUpdate": [ {
            "action": "loadMovieData",
            "observers": [ "props.context.selection.uid",
"props.context.selection[0].uid", "props.context.selection.length" ]
        } ]
    },
    "props": {
        "context": {
            "type": "object"
        }
    }
}
```

## Step 3: Consume in Secondaryworkarea.

Use `MovieInfoView.html` in `favSelectionDetailsView.html` as shown below:

```
<movie-info context="props.subPanelContext"></movie-info>
```

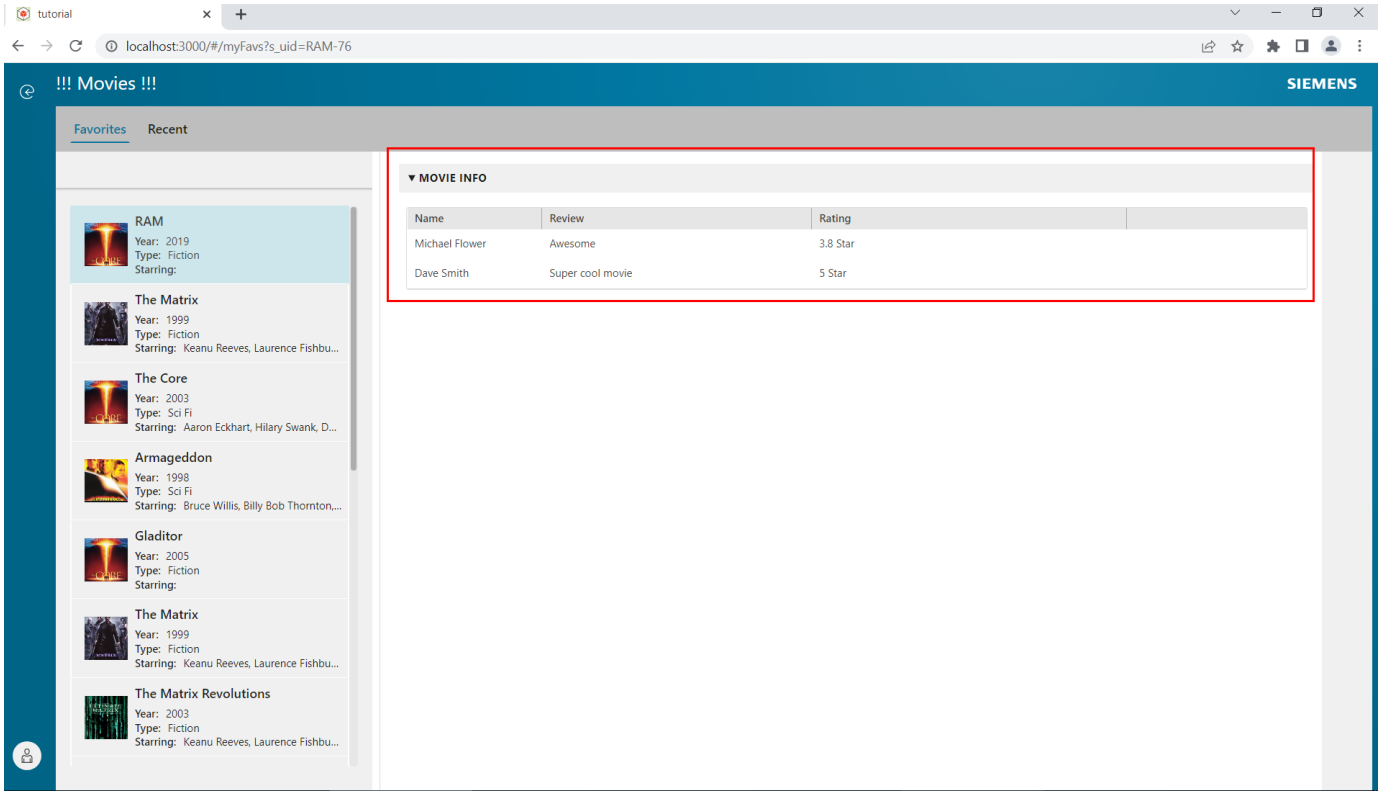
## Step 4: Build and Test.

1. Run the following build commands to build and deploy the site.

   ```
   set SITE=site_tutorial
   aw build site_tutorial
   aw start
   ```

2. Launch your application at following port (Port number is defined in server.js ).

   ```
   localhost:8181
   ```




## Summary

In this tutorial, you learned how to display contents in a table in the Secondary Work Area.

## References

- Data provider
- Data parser
- Actions
- Localization

# Local and global selections

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from Support Center, as explained in the Tutorials Overview.


Unpublished work. © 2025 Siemens

## Objective

You will learn how to configure local and global selections.

## Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **Selection Summary with table** before proceeding.

**Otherwise:**

Download the source code for the **selection_summary_with_table** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Add a command anchor in custom secondary workarea.

In `MovieInfoView.html`, add the following code to introduce a custom command anchor `fav_secondaryWorkArea` for the said view.

```
<aw-panel-section caption='Movie Info'>
    <aw-panel-body>
        <aw-command-bar anchor="fav_secondaryWorkArea"
alignment="HORIZONTAL" context="props.context" class="afx-alt-content-
background aw-toolbar-layout aw-toolbar-horizontal"></aw-command-bar>
        <aw-splm-table gridid="movieTableGrid"></aw-splm-table>
    </aw-panel-body>
</aw-panel-section>
```

## Step 2: Pass selection data to custom table in secondary workarea.

In `MovieInfoViewModel.json`, modify the input to the dataprovider to take the selectionData received in subPanelContext.

```
{
    "dataProviders": {
        "movieDataProvider": {
            "action": "getSelectedObject",
            "response": "{{data.results}}",
            "totalFound": "{{data.totalFound}}",
            "inputData": {
                "selectionModel": "{{props.context.selectionModel}}",
                "selectionData": "{{props.context.selectionData}}"
            },
```


Unpublished work. © 2025 Siemens

```
            "selectionModelMode": "single"
        }
    }
}
```

## Step 3: Contribute three commands in Primary workarea, Secondary workarea and right wall.

Add `commandsViewModel.json` with following content at the module level.

```
{
    "commands": {
        "SampleGlobalCopyCmd": {
            "iconId": "cmdCopy",
            "title": "{{i18n.sampleGlobalCopyCmdTitle}}",
            "description": "{{i18n.sampleCopyCmdDesc}}"
        },
        "SamplePwaCopyCmd": {
            "iconId": "cmdCopy",
            "title": "{{i18n.samplePwaCopyCmdTitle}}",
            "description": "{{i18n.sampleCopyCmdDesc}}"
        },
        "SampleSwaCopyCmd": {
            "iconId": "cmdCopy",
            "title": "{{i18n.sampleSwaCopyCmdTitle}}",
            "description": "{{i18n.sampleCopyCmdDesc}}"
        }
    },
    "commandHandlers": {
        "SamplePwaCopyCmdHandler": {
            "id": "SamplePwaCopyCmd",
            "action": "CopyObject",
            "activeWhen": true,
            "visibleWhen": {
                "condition": "conditions.shouldShowCopyCmd"
            },
            "enableWhen": {
                "condition": "conditions.shouldEnableCopyCmd"
            }
        },
        "SampleSwaCopyCmdHandler": {
            "id": "SampleSwaCopyCmd",
            "action": "CopyObject",
            "activeWhen": true,
            "visibleWhen": {
                "condition": "conditions.shouldShowCopyCmd"
            },
            "enableWhen": {
                "condition": "conditions.shouldEnableCopyCmd"
            }
        },
        "SampleCopyCmdHandler": {
            "id": "SampleGlobalCopyCmd",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "action": "CopyObject",
            "activeWhen": true,
            "visibleWhen": {
                "condition": "conditions.shouldShowCopyCmd"
            },
            "enableWhen": {
                "condition": "conditions.shouldEnableCopyCmd"
            }
        }
    },
    "commandPlacements": {
        "SampleCopyPrimaryCmdPlacement": {
            "id": "SamplePwaCopyCmd",
            "uiAnchor": "aw_primaryWorkArea",
            "priority": 200
        },
        "SampleCopySecondaryCmdPlacement": {
            "id": "SampleSwaCopyCmd",
            "uiAnchor": "fav_secondaryWorkArea"
        },
        "SampleCopyRightWallCmdPlacement": {
            "id": "SampleGlobalCopyCmd",
            "uiAnchor": "aw_workarea_toolbar",
            "priority": 10
        }
    },
    "actions": {
        "CopyObject": {
            "actionType": "Event",
            "method": "Event",
            "actionMessages": {
                "success": [ {
                    "message": "showCopiedMessage"
                } ]
            }
        }
    },
    "conditions": {
        "shouldShowCopyCmd": {
            "expression": "commandContext.nameToken
=== 'com.siemens.splm.client.favorites:favoritesSubLocation'
|| commandContext.context.nameToken ===
'com.siemens.splm.client.favorites:favoritesSubLocation' ||
commandContext.provider.nameToken ===
'com.siemens.splm.client.favorites:favoritesSubLocation'"
        },
        "shouldEnableCopyCmd": {
            "expression": "commandContext.selectionData.value.selected &&
commandContext.selectionData.value.selected.length === 1"
        }
    },
    "messages": {
        "showCopiedMessage": {
            "messageType": "INFO",
            "messageText": "{{i18n.copySuccessful}}",
            "messageTextParams": [
```

```
"{{commandContext.selectionData.value.selected[0].displayName}}"
            ]
        }
    },
    "i18n": {
        "sampleGlobalCopyCmdTitle": [ "helloModuleMessages" ],
        "samplePwaCopyCmdTitle": [ "helloModuleMessages" ],
        "sampleSwaCopyCmdTitle": [ "helloModuleMessages" ],
        "sampleCopyCmdDesc": [ "helloModuleMessages" ],
        "copySuccessful": [ "helloModuleMessages" ]
    }
}
```

Add i18n in `helloModuleMessages.json` as shown below:

```
{
    "sampleGlobalCopyCmdTitle": "Copy global selection",
    "samplePwaCopyCmdTitle": "Copy PWA selection",
    "sampleSwaCopyCmdTitle": "Copy SWA selection",
    "sampleCopyCmdDesc": "Copy selected object in Clipboard",
    "copySuccessful": "\"{0}\" was copied to OS clipboard"
}
```

## Step 4: Build and Test.

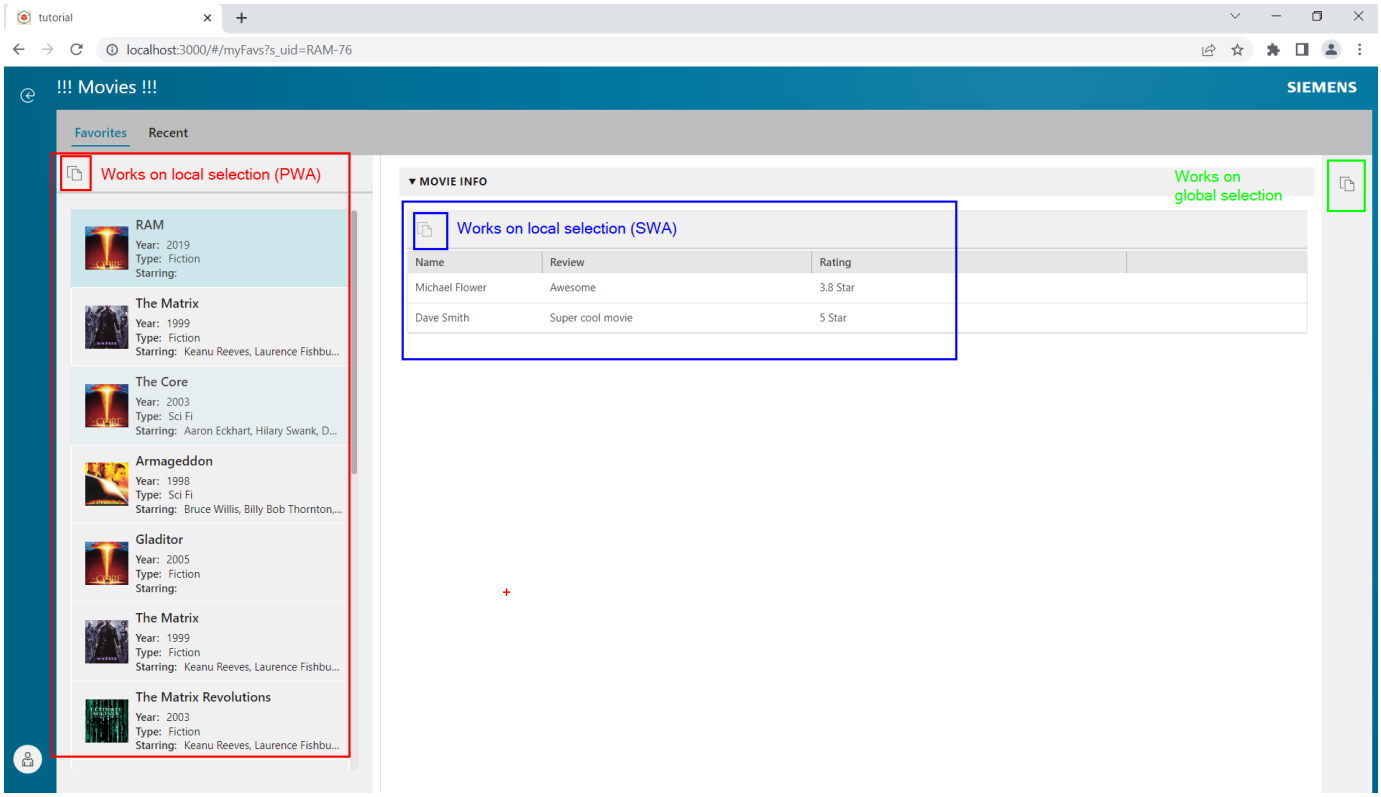
1. Run the following build commands to build and deploy the site.

```
set SITE=site_tutorial
aw build
aw start
```

2. Launch your application at following port (Port number is defined in server.js ).

```
localhost:8181
```


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Summary

In this tutorial, you learned how to configure local and global selections.

## References

- Selection Data

# Selection master-details pattern

**Note**

This tutorial is compatible with Siemens Web Framework v6.0 or higher. Older versions of this tutorial that are compatible with older versions of Siemens Web Framework are included in the Siemens Web Framework .zip file that can be downloaded from Support Center, as explained in the Tutorials Overview.

## Objective

You will learn how to configure the master/details selection pattern.

## Pre-requisites

You must have already gone through **pre-requisite software** by now.

**Recommended:**

Complete **Local and Global Selections** before proceeding.

**Otherwise:**

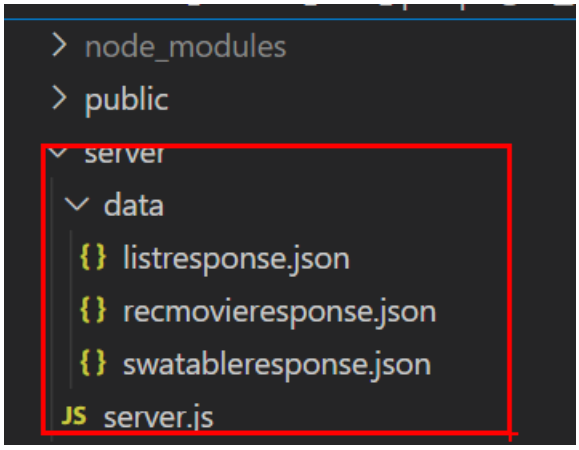
Download the source code for the **local_and_global_selections** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

## Step 1: Setup the backend server that serves data to the second table in SWA - recommended movies.

Download the source code for the **selection_master_details_pattern** tutorial, available in the **Tutorials/src** folder of the Siemens Web Framework release ZIP file you downloaded from Support Center (see **Pre-requisite software** for information on how to download Siemens Web Framework).

> **Tip**
>
> Note that this folder contains the source code of the *completed* tutorial, so unzip it outside your current project folder if you are planning to follow along with the rest of the tutorial.

Inside the **selection_master_details_pattern** folder, copy the **server** folder to the root folder of your project.





Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 2: Add the view and viewodel for recommended movies.

Add `RecommendedMovieView.html` and `RecommendedMovieViewModel.json` in the html and viewmodel folder of the module respectively.

```
<aw-panel-section caption='Recommended Movies'>
    <aw-panel-body>
        <aw-splm-table gridid="recMovieTableGrid"></aw-splm-table>
    </aw-panel-body>
</aw-panel-section>
```

```
{
    "schemaVersion": "1.0.0",
    "grids": {
        "recMovieTableGrid": {
            "dataProvider": "recMovieDataProvider",
            "columnProvider": "recMovieColumnProvider",
            "addIconColumn": false,
            "gridOptions": {
                "enableGridMenu": false,
                "enablePinning": false,
                "enableSorting": false
            }
        }
    },
    "dataProviders": {
        "recMovieDataProvider": {
            "action": "getSelectedObject",
            "response": "{{data.results}}",
            "totalFound": "{{data.totalFound}}",
            "inputData": {
                "selectionModel": "{{props.context.selectionModel}}",
                "selectionData": "{{props.context.selectionData}}"
            },
            "selectionModelMode": "single"
        }
    },
    "actions": {
        "loadMovieData": {
            "actionType": "dataProvider",
            "method": "recMovieDataProvider"
        },
        "getSelectedObject": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/recmovies",
                    "params": {
                        "userName": "{{props.recommendationsBy[0].uid}}"
```

```
                    }
                }
            },
            "outputData": {
                "results": "data",
                "totalFound": "data.length"
            }
        }
    },
    "columnProviders": {
        "recMovieColumnProvider": {
            "columns": [ {
                    "name": "name",
                    "displayName": "Name",
                    "minWidth": 150,
                    "width": 150
                },
                {
                    "name": "review",
                    "displayName": "Review",
                    "minWidth": 300,
                    "width": 300
                },
                {
                    "name": "rating",
                    "displayName": "Rating",
                    "minWidth": 350,
                    "width": 350
                }
            ],
            "frozenColumnIndex": -1
        }
    },
    "lifecycleHooks": {
        "onUpdate": [ {
            "action": "loadMovieData",
            "observers": [ "props.recommendationsBy" ]
        } ]
    },
    "props": {
        "context": {
            "type": "object"
        },
        "recommendations-by":{
            "type": "object"
        }
    }
}
```

This component takes `recommendations-by` as an input and on change of that it does the REST call to fetch the movies recommended by that user.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Step 3: Consume the new table.

Consume the second table in `favSelectionDetailsView.html` by adding the code shown below:

```
<aw-flex-column>
    <aw-flex-row>
        <movie-info context="{...props.subPanelContext, selectionData:
data.selectionData, selectionModel:selectionModels.masterSelectionModel}"></
movie-info>
    </aw-flex-row>
    <aw-flex-row>
        <recommended-movie context="{...props.subPanelContext,
selectionData:data.selectionData,selectionModel:selectionModels.detailsSelec
tionModel}" recommendations-by="data.masterSelection"></recommended-movie>
    </aw-flex-row>
</aw-flex-column>
```

Define the selection-data and selection-model in `favSelectionDetailsViewModel.json`

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "selectionData": {
            "initialValues": {},
            "meta": {}
        }
    },
    "lifecycleHooks": {
        "onUpdate": [ {
                "action": "handleSelectionChange",
                "observers": [ "viewModel.atomicData.selectionData" ]
            }
        ]
    },
    "actions":{
        "handleSelectionChange": {
            "actionType": "JSFunction",
            "method": "handleSelectionChangeForMasterDetails",
            "inputData": {
                "localSelectionData": "{{data.selectionData}}",
                "parentSelectionData":
"{{props.subPanelContext.selectionData}}",
                "dispatch": "{{data.dispatch}}",
                "atomicDataRef": "{{data.atomicDataRef}}",
                "masterSelection": "{{data.masterSelection}}",
                "masterSelectionModel":
"{{selectionModels.masterSelectionModel}}"
            },
            "deps": "js/selectionHelperService"
        }
    },
    "selectionModels": {
```

```
        "masterSelectionModel": {
            "selectionMode": "single"
        },
        "detailsSelectionModel": {
            "selectionMode": "single"
        }
    }
}
```

This component defines two selection models - `masterSelectionModel` and `detailsSelectionModel` and sends them to master (`movieInfo`) and details (`recommendedMovie`) respectively. It also defines the selection data and observes it to manage the master details selection management business logic. In order to get that, add a file called `selectionHelperService` in the JS folder of the module

```
import _ from 'lodash';

export const handleSelectionChangeForMasterDetails = ( localSelectionData,
parentSelectionData, dispatch, atomicDataRef, masterSelection,
masterSelectionModel ) => {
    if( !_.isEmpty( localSelectionData && parentSelectionData ) &&
localSelectionData.selected ) {
        if( localSelectionData.id === 'masterSelectionModel' ) {
            handleMasterSelectionChange( localSelectionData,
parentSelectionData, dispatch );
        }
        if( localSelectionData.id === 'detailsSelectionModel' ) {
            handleDetailsSelectionChange( localSelectionData,
parentSelectionData, atomicDataRef, masterSelection, masterSelectionModel );
        }
    }
};

const handleMasterSelectionChange = ( localSelectionData,
parentSelectionData, dispatch ) => {
    parentSelectionData &&
parentSelectionData.update( { ...localSelectionData } );
    dispatch && dispatch( { path: 'data.masterSelection', value:
localSelectionData.selected } );
};

const handleDetailsSelectionChange = ( localSelectionData,
parentSelectionData, atomicDataRef, masterSelection, masterSelectionModel )
=> {
    if( localSelectionData.selected.length <= 0 ) {
        var finalSelectionData = { ...localSelectionData };
        finalSelectionData.selected = masterSelection;
        finalSelectionData.id = 'masterSelectionModel';
        masterSelectionModel.setSelectionModelActivated( true );
        atomicDataRef &&
atomicDataRef.selectionData.setAtomicData( finalSelectionData );
    } else {
        masterSelectionModel.setSelectionModelActivated( false );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        parentSelectionData &&
parentSelectionData.update( { ...localSelectionData } );
    }
};
```

When selection from details view is lost, the selection in master view needs to be highlighted or activated which is why it is maintained as `data.masterSelection` in the parent view which is also sent as a separate prop to the details view as the input on which it needs to work on.

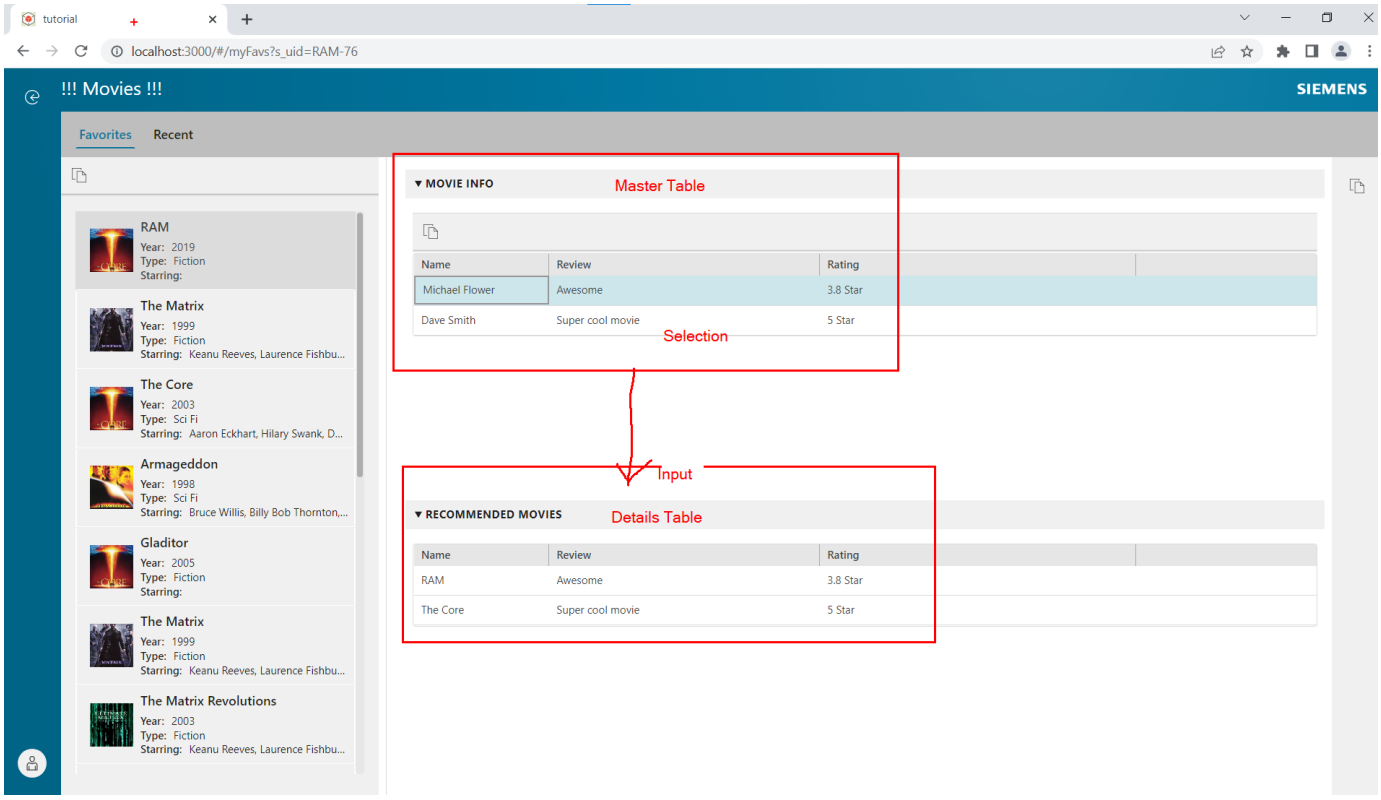
## Step 4: Build and Test.

1. Run the following build commands to build and deploy the site.

```
set SITE=site_tutorial
aw build site_tutorial
aw start
```

2. Launch your application at following port (Port number is defined in server.js ).

```
localhost:8181
```



## Summary

In this tutorial, you learned how to configure the master/details selection pattern.

## References

- **Selection Data**


Siemens Web Framework
Unpublished work. © 2025 Siemens

# 4. Advanced Tutorials

If you are new to Siemens Web Framework, the best way to learn and get started is to follow the step-by-step **tutorials**.

Once mastered you mastered the step-by-step tutorials and you know your way around the framework, you can proceed with the following advanced-level tutorials which will help you understand the next-level concepts and usage patterns.

- **Creating Reusable Components**
- **Interaction Patterns between Components**
- **Building Custom Pages**

## Creating reusable components

A **Component** is a reusable, composable, debuggable unit that has a specific behavior. It typically has a render function and a **view model**.

The **render function** can be written codefully (in JSX) or via a **view template**, which will be converted into JSX by the view compiler of Siemens Web Framework, at build time.

The following tutorials provide some simple but useful examples on how to create reusable components:

- **Creating a component using a declarative view**
- **Creating a component using a JSX render function - Example 1**
- **Creating a component using a JSX render function - Example 2**
- **Creating a component performing DOM manipulation**
- **Creating a component integrating a third-party React component**
- **Creating a component integrating a third-party vanilla JavaScript component**

> **Tip**
>
> If you are familiar with the React framework, the following page can be useful to understand how common React hooks can be mapped to Active Workspace constructs: **Mapping React hooks to declarative constructs**


Unpublished work. © 2025 Siemens

## Creating a component using a declarative view

This example features a mid-level component that is based on the existing components from Siemens Web Framework, say `aw-list` and `aw-search-box`. This new component will be reusable and will have the behavior of providing a list with a search box which can assist in performing a search within that list.

Here are the steps to be followed for creating this reusable component using a declarative **view**.

1. Create a new component called AwListWithSearchboxView.html which has the usage of AwList and AwSearchBox

```
<aw-panel>
    <aw-panel-body>
        <aw-panel-section caption="{{props.dataCollectionName}}">
            <aw-search-box prop="data.searchBox"
placeholder="data.searchBoxPlaceHolder.dbValue" action="fetchData"></aw-
search-box>
            <aw-list dataprovider="data.dataProviders.searchData">
                <aw-default-cell vmo="item"> </aw-default-cell>
            </aw-list>
        </aw-panel-section>
    </aw-panel-body>
</aw-panel>
```

2. Add the corresponding view model for this component, called `AwListWithSearchboxViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "searchBox": {
            "displayName": "Search",
            "type": "string"
        },
        "searchBoxPlaceHolder": {
            "dbValue": "Search"
        }
    },
    "dataProviders": {
        "searchData": {
            "action": "search",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}"
        }
    },
    "actions": {
        "fetchData": {
            "actionType": "dataProvider",
            "method": "searchData"
        },
        "search": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "{{function:getDataUrl}}"
                }
            },
            "dataParsers": [ {
                "id": "fetchedDataParser",
                "responseObjs": "data.results"
            } ],
            "outputData": {
                "totalFound": "data.totalResults",
                "searchResults": "data.results"
            },
            "deps": "js/listWithSearchBoxHelper"
        }
    },
    "dataParseDefinitions": {
        "fetchedDataParser": {
            "hasThumbnail" : false,
            "cellHeader1": "{{response.name}}"
        }
    },
    "lifecycleHooks": {
        "onMount": "fetchData"
    },
    "functions": {
        "getDataUrl": {
            "functionName": "getDataUrl",
            "parameters": [
                "{{props.dataSource}}",
                "{{data.searchBox.uiValue}}"
            ]
        }
    },
    "props":{
        "data-source":{
            "type": "string"
        },
        "data-collection-name":{
            "type": "string"
        }
    }
}
```

3. Add the required glueCode in `listWithSearchBoxHelper.js` as:

```
var exports = {};
export let getDataUrl= function( dataSource, filterString ) {
    return dataSource + encodeURIComponent( filterString );
};
export default exports = {
```

```
    getDataUrl
};
```

Together this would provide a reusable component called AwListWithSearchbox which would have a search box and a list shown by default.

Notice the `props` section in the view model. This reusable component works off `data-source` provided by the parent component. As long as these data sources are different, different sets of data can be shown by this component. The same applies to `data-collection-name` which is accepted as a prop/input from the parent component and can be used by the parent component to specify a different name to be used for the panel section name or caption.

Here are the steps to consume this component from any parent component:

1. Create a consumer component say ListWithSearchBoxExamplePageView.html with the following content

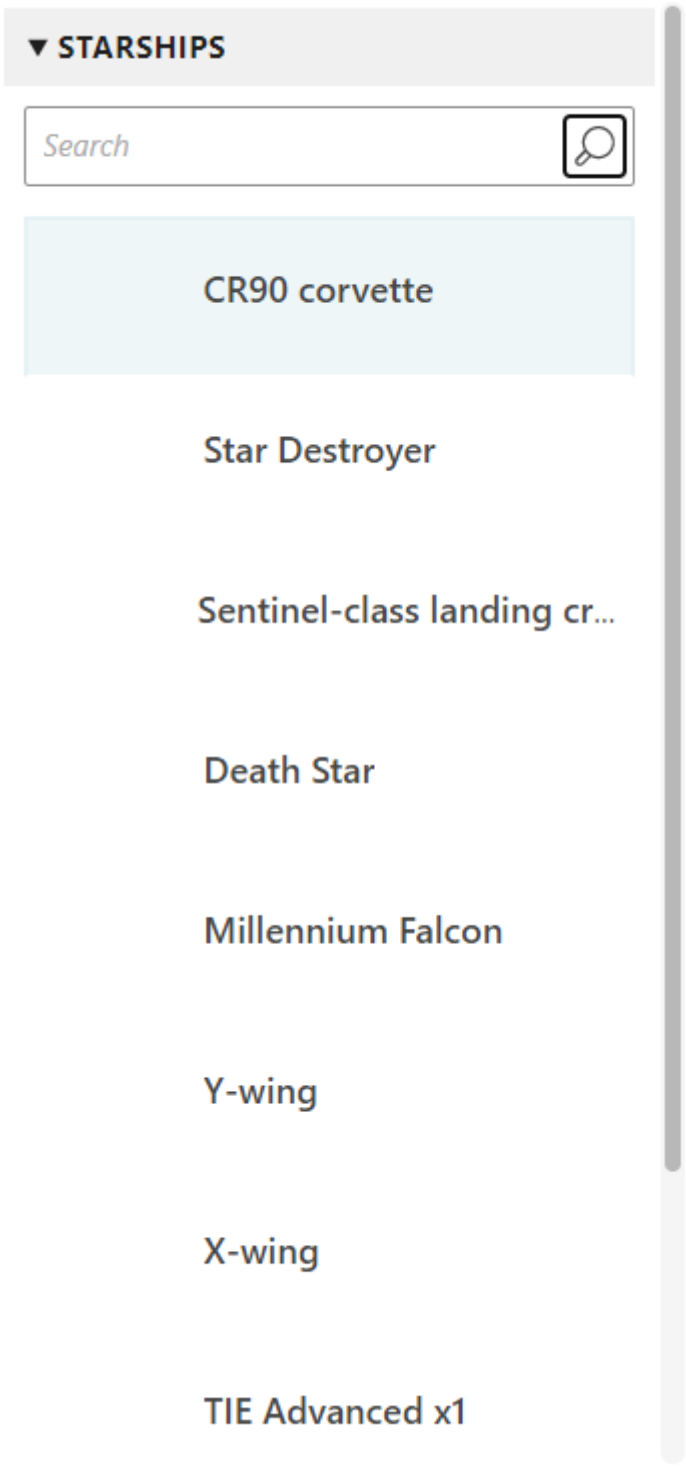
```
<aw-row>
    <aw-column width="3">
        <aw-list-with-searchbox data-collection-
name="Starships" data-source="https://swapi.dev/api/starships/?
search="></aw-list-with-searchbox>
    </aw-column>
</aw-row>
```

2. The corresponding consuming view model may not need any special behavior so can be empty like

```
{
    "schemaVersion": "1.0.0"
}
```

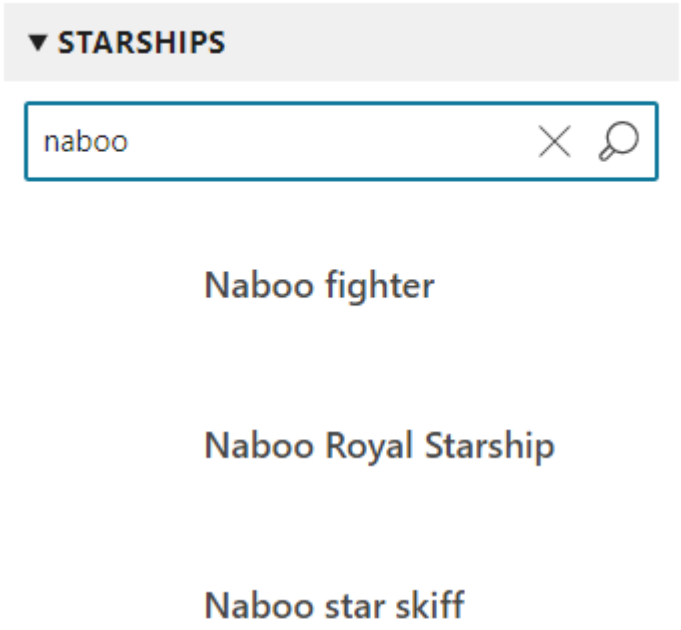

Siemens Web Framework
Unpublished work. © 2025 Siemens

Once executed, this will show the output as follows:




Search can be performed

using the search box to get the filtered results like this:



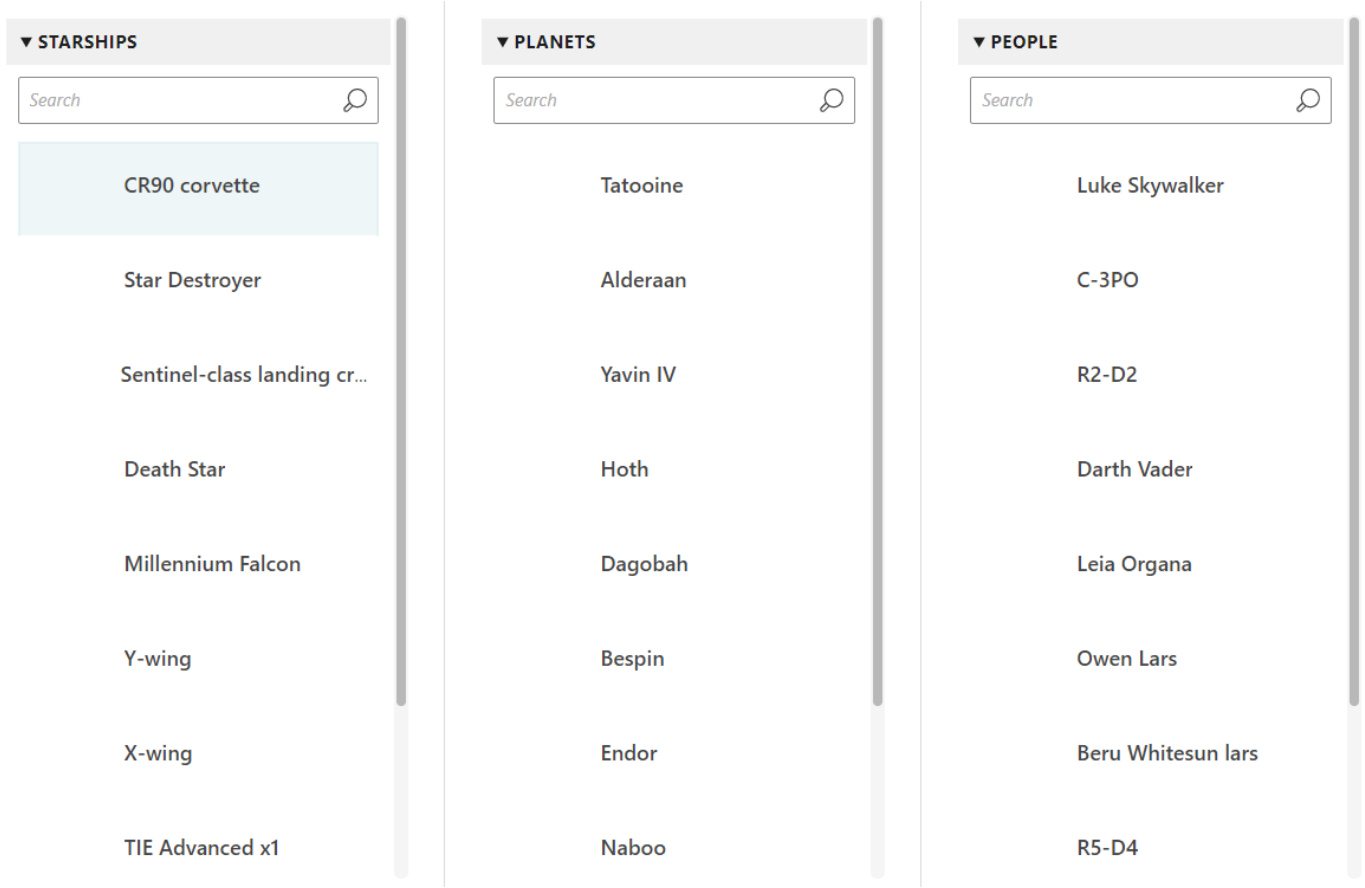

3. To appreciate the reusable component, now add the following into `ListWithSearchBoxExamplePageView.html` to see how with variation in `data-source`, different data can be fetched using the same component.

```
<aw-row>
    <aw-column width="3">
        <aw-list-with-searchbox data-collection-
name="Starships" data-source="https://swapi.dev/api/starships/?
search="></aw-list-with-searchbox>
    </aw-column>
    <aw-splitter direction="VERTICAL"></aw-splitter>
    <aw-column width="3">
        <aw-list-with-searchbox data-collection-
name="Planets" data-source="https://swapi.dev/api/planets/?search="></
aw-list-with-searchbox>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

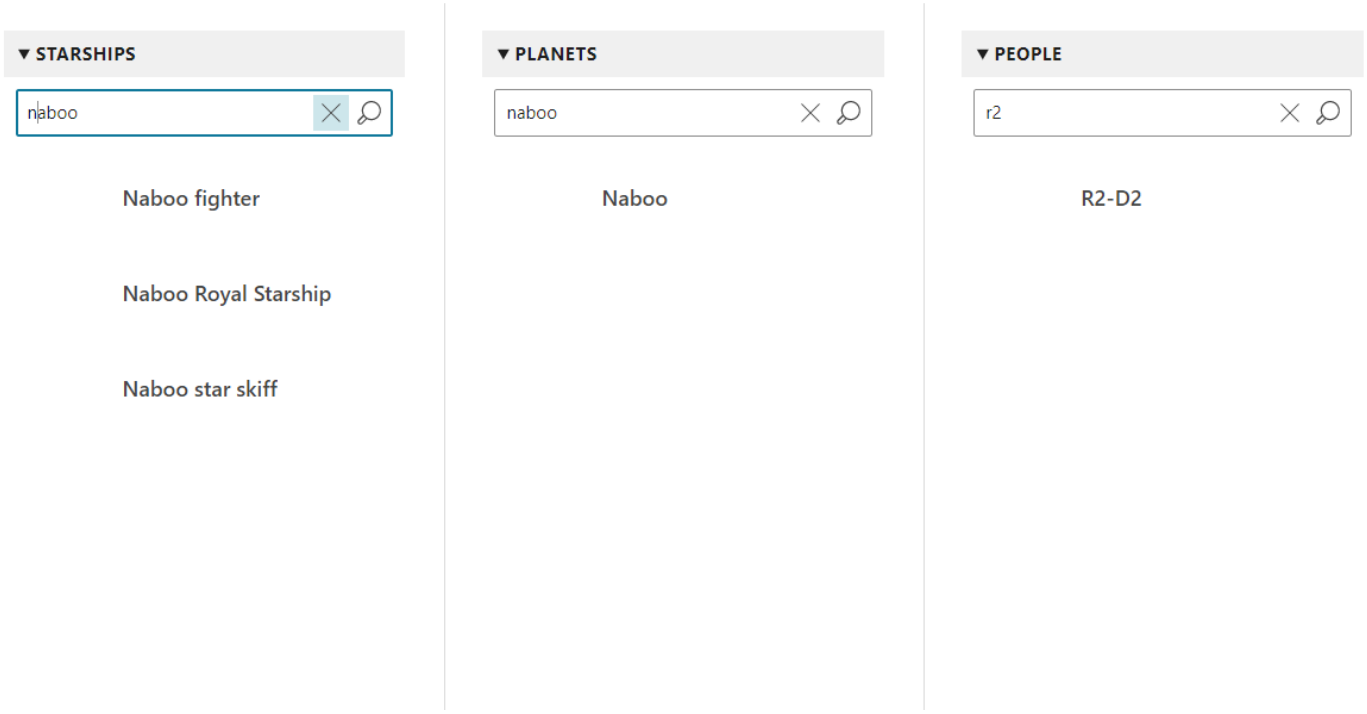
```
    </aw-column>
    <aw-splitter direction="VERTICAL"></aw-splitter>
    <aw-column width="3">
    <aw-list-with-searchbox data-collection-name="People" data-
source="https://swapi.dev/api/people/?search="></aw-list-with-searchbox>
    </aw-column>
</aw-row>
```

Once executed, this will show the output as follows:




Here, we are using the same component to show the data from Starships, Planets, and People.

The search can be performed independently like this:




## Creating a component using a JSX render function - Example 1

This example features a low-level component that is based on the HTML Color input element. This new component will be reusable and will have the color picker behavior wherein the selected color will be rendered in a custom div.

Here are the steps to be followed for creating this reusable component using a JSX render function.

1. Create a view model for this component called `AwColorPickerViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "render": "render"
    },
    "actions": {
        "render": {
            "method": "awColorPickerRenderFunction",
            "deps": "js/AwColorPickerService"
        }
    },
    "props": {
        "selected-color": {
            "type": "object"
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

2. Create a new service file called `AwColorPickerService.js` which has a render function called `awColorPickerRenderFunction`:

```
import AwFlexRow from 'viewmodel/AwFlexRowViewModel';
import AwFlexColumn from 'viewmodel/AwFlexColumnViewModel';

export const awColorPickerRenderFunction = props => {
    const { viewModel } = props;
    const setColor = value => {
        let newColor = { ...props.selectedColor.getValue() };
        newColor.value = value;
        props.selectedColor.update( newColor );
    };

    return (
        <input type='color' value={props.selectedColor.value}
onChange={e => setColor( e.target.value )} />
    );
};
```

Together this would provide a reusable component called AwColorPicker which would have a color picker and a flex column rendered in the selected color. The default color is initialized as black.

Notice the render function has the entire view model available on the props that it receives. `props.selectedColor` is received from the consuming component and on selecting a different value (onChange on input element), the callback function is invoked which is going to update the parent state received by the AwColorPicker component. This is the same color that will be displayed in the input element.

Here are the steps to consume this component from any parent component:

1. Create a consumer component say `AwColorPickerExampleView.html` with following content:

```
<aw-panel>
     <aw-panel-body class="sw-row">
         <aw-panel-section class="h-12 w-6" caption="Left">
             <aw-flex-row height="50f">
                 <aw-flex-column width="12">
                     <aw-color-picker class="h-1" selected-
color="fields.selectedColorL"></aw-color-picker>
                     <br></br>
                     <aw-flex-row class="h-11"
color="{{fields.selectedColorL.value}}">
                     </aw-flex-row>
                 </aw-flex-column>
             </aw-flex-row>
         </aw-panel-section>
         <aw-panel-section class="h-12 w-6" caption="Right">
             <aw-flex-row height="50f">
                 <aw-flex-column width="12">
                     <aw-color-picker class="h-1" selected-
```


Unpublished work. © 2025 Siemens

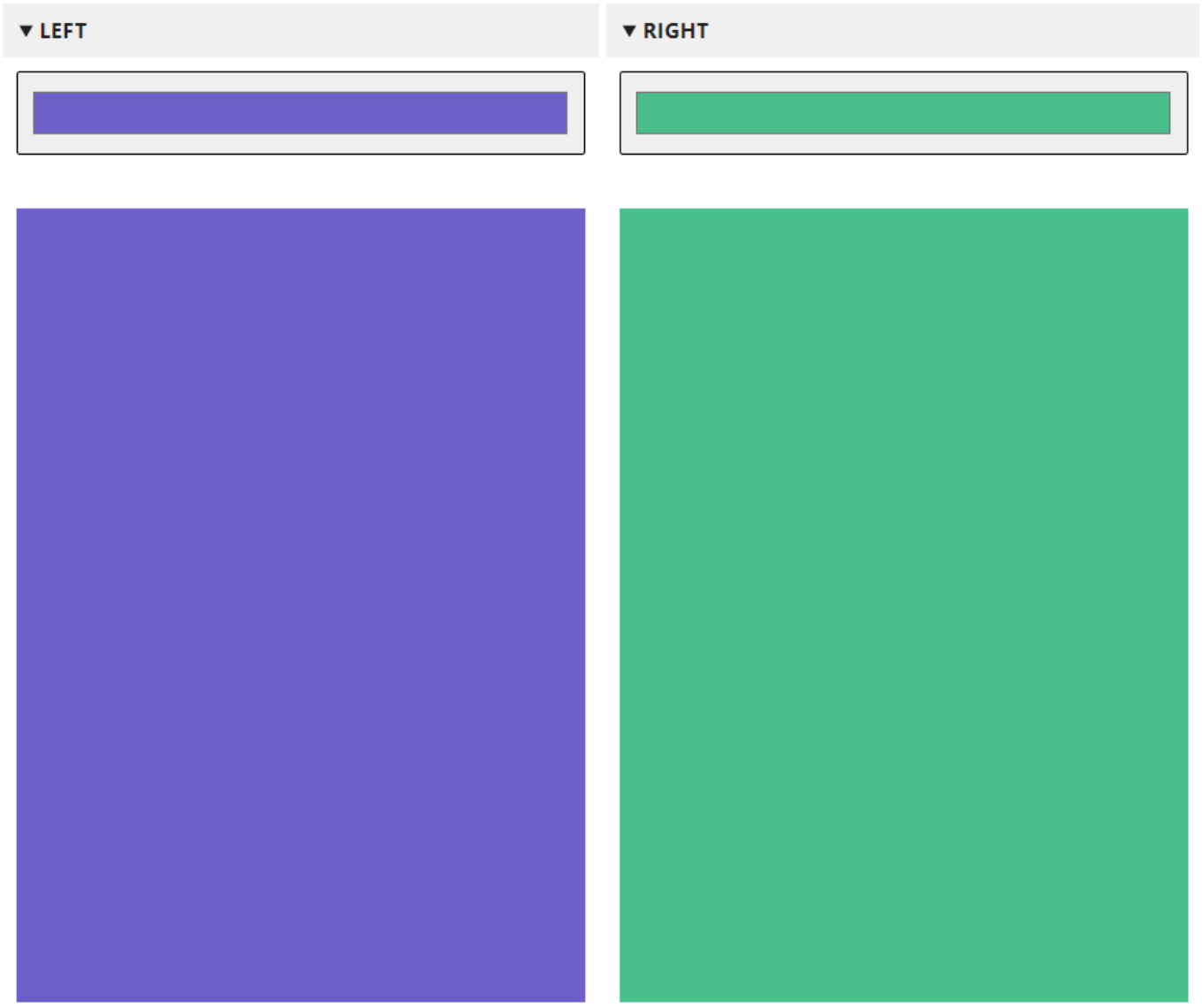
```
color="fields.selectedColorR"></aw-color-picker>
                    <br></br>
                    <aw-flex-row class="h-11"
color="{{fields.selectedColorR.value}}">
                    </aw-flex-row>
                </aw-flex-column>
            </aw-flex-row>
        </aw-panel-section>
    </aw-panel-body>
</aw-panel>
```

2. The corresponding consuming view model will need to define the initial colors using **atomic data**:

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "selectedColorL": {
            "initialValues": {
                "value": "#6c5fc8"
            },
            "meta": {}
        },
        "selectedColorR": {
            "initialValues": {
                "value": "#4abf8c"
            },
            "meta": {}
        }
    }
}
```

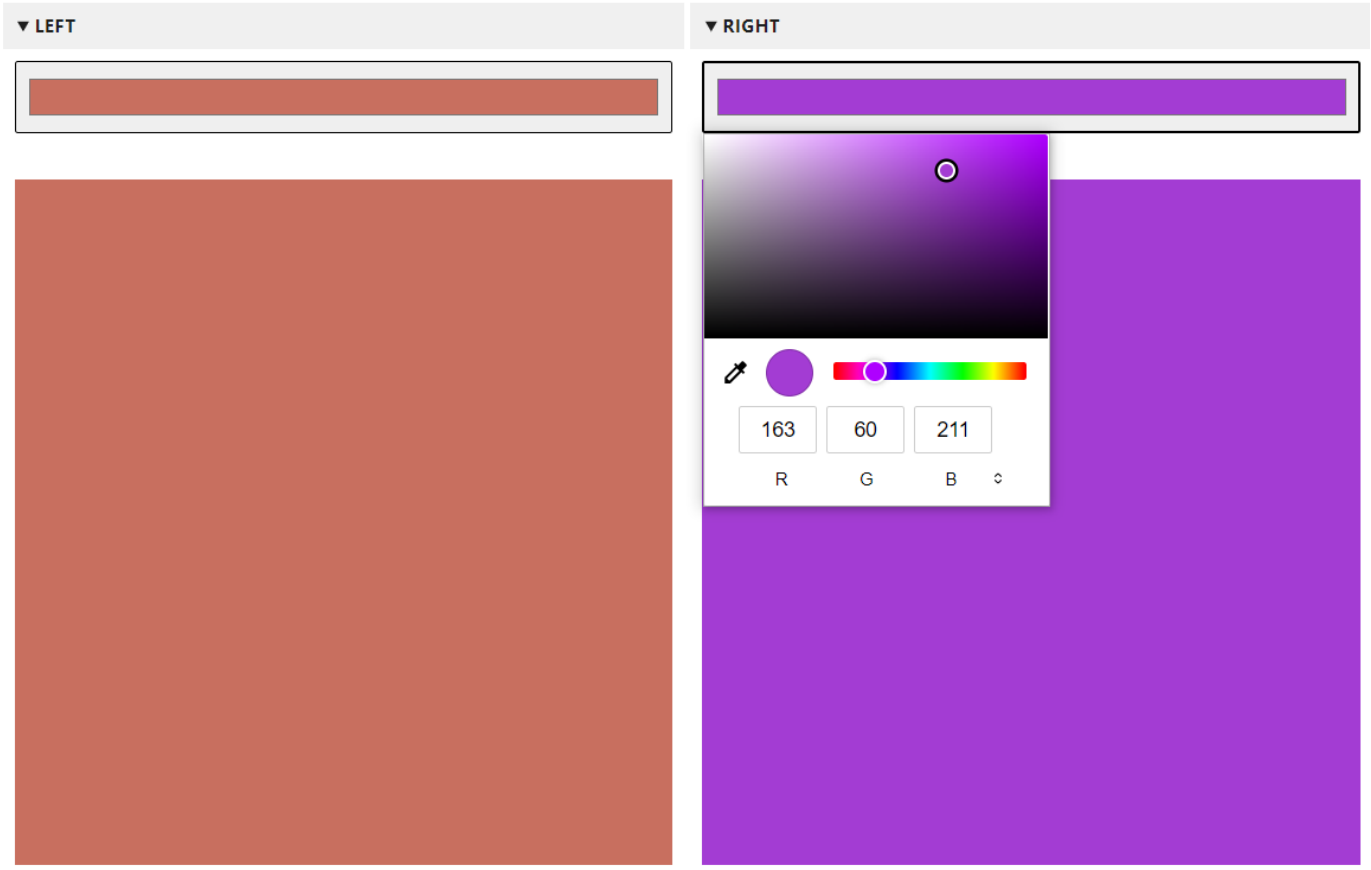
Once executed this will show the output as follows:


Siemens Web Framework
Unpublished work. © 2025 Siemens




On selecting a different color, the same would be shown in the input element as well as the column:


Unpublished work. © 2025 Siemens




## Creating a component using a JSX render function - Example 2

This example features a low-level component that is based on the HTML input elements. This new component will be reusable and will have the counter behavior wherein the count will be incremented every time the button is clicked and the same information will be dumped into the console.

Here are the steps to be followed for creating this reusable component using a JSX render function.

1. Create a view model for this component called `AwCounterViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "render": "render",
        "onUpdate": {
            "observers": [ "viewModel.data._count" ],
            "action": "showCount"
        }
    },
    "data":{
        "_count": 0
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    },
    "actions": {
        "render": {
            "method": "awCounterRenderFunction",
            "deps": "js/AwCounterService"
        },
        "setCount": {
            "actionType": "JSFunction",
            "method": "setCount",
            "inputData": {
                "count": "{{data._count}}"
            },
            "outputData": {
                "_count": ""
            },
            "deps": "js/AwCounterService"
        },
        "showCount": {
            "actionType": "JSFunction",
            "method": "showCount",
            "inputData": {
                "count": "{{data._count}}"
            },
            "deps": "js/AwCounterService"
        }
    }
}
```

2. Create a new service file called `AwCounterService.js` which has a render function called `awCounterRenderFunction`:

```
import AwButton from 'viewmodel/AwButtonViewModel';
import AwBreak from 'viewmodel/AwBreakViewModel';

export const awCounterRenderFunction = ( props ) => {
    const { viewModel } = props;
    return (
        <div>
            <div>You clicked { viewModel.data._count } times</div>
            <AwBreak></AwBreak>
            <AwButton size='auto'
action={ props.actions.setCount }>Click Me</AwButton>
        </div> );
};

export const showCount = count => {
    console.log( count );
};

export const setCount = count => {
    return count + 1;
};
```

Together this would provide a reusable component called AwCounter which would have a count dumped on the screen as well as the console.log every time the button is clicked. The default count is initialized as 0.

Notice the render function has the entire view model available on the props that it receives. `viewModel.data._count_` is initialized to 0 and on clicking the button, the action `setCount` is invoked which is going to increment the count and put it on the view model via the `outputData`. This is a declarative way to do a dispatch (equivalent to setState) on the AwCounter component. This is the same count that will be displayed both on the screen as well as on the console.

Here are the steps to consume this component from any parent component:

1. Create a View for a consumer component say `AwCounterExampleView.html` with following content:

```
<aw-panel>
    <aw-panel-body>
        <aw-row height="fill">
            <aw-column width="6">
                <aw-panel-section caption="LEFT VIEW">
                    <aw-counter></aw-counter>
                </aw-panel-section>
            </aw-column>
        </aw-row>
    </aw-panel-body>
</aw-panel>
```

2. The corresponding consuming view model may not need any special behavior so can be empty like

```
{
    "schemaVersion": "1.0.0"
}
```

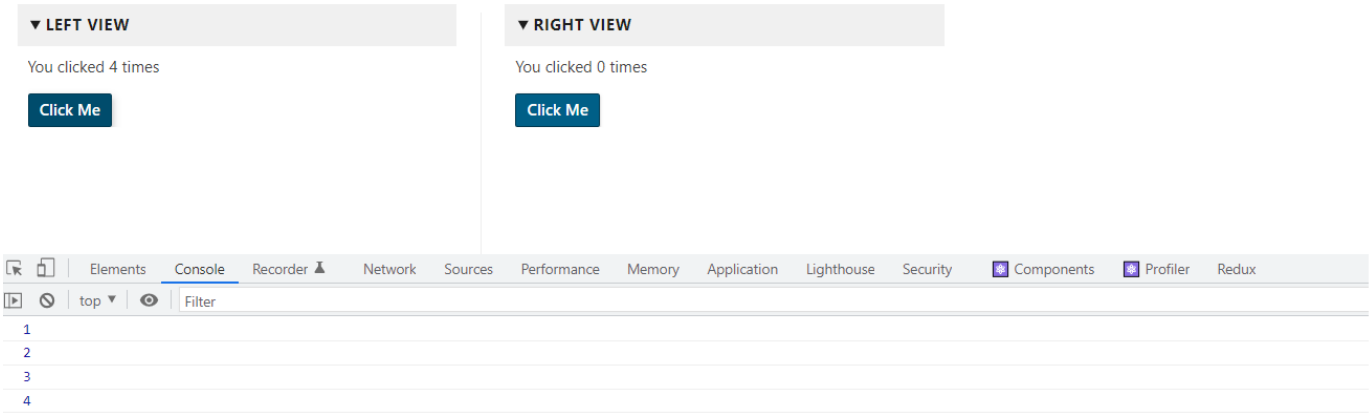
3. Inorder to appreciate the reusable component, now add following into `AwCounterExampleView.html`

```
<aw-panel>
    <aw-panel-body>
        <aw-row height="fill">
            <aw-column width="6">
                <aw-panel-section caption="LEFT VIEW">
                    <aw-counter></aw-counter>
                </aw-panel-section>
            </aw-column>
            <aw-separator is-vertical="true"></aw-separator>
            <aw-column width="6">
                <aw-panel-section caption="RIGHT VIEW">
                    <aw-counter></aw-counter>
                </aw-panel-section>
            </aw-column>
```

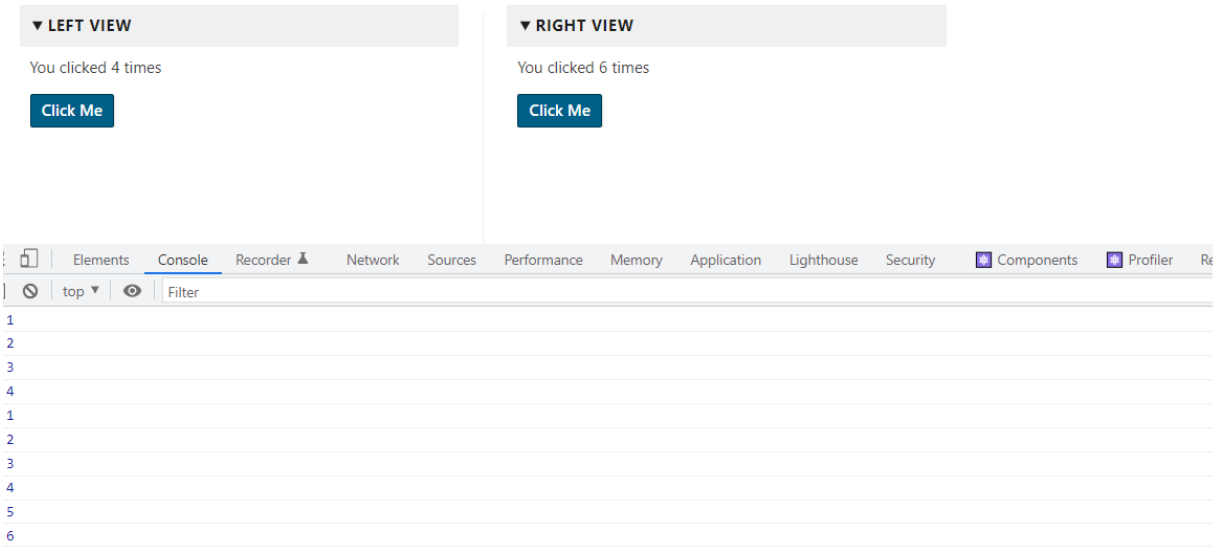

Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        </aw-row>
    </aw-panel-body>
</aw-panel>
```

Once executed this will show the output as follows:




On selecting from the right view, the output will be as follows:




## Creating a component performing DOM manipulation

This example features a low-level component that demonstrates accessing the DOM. This new component will be reusable and will have the behavior of focusing on the input box when the corresponding button is clicked.

Here are the steps to be followed for creating this reusable component using a JSX render function.

1. Create a view model for this component called `AwFocusInputBoxViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "onMount": "initialize",
        "render": "render"
    },
    "actions": {
        "render": {
            "method": "awFocusInputBoxRenderFunction",
            "deps": "js/AwFocusInputBoxService",
            "options": {
                "elementRefList": [ "inputBox" ]
            }
        },
        "initialize": {
            "actionType": "JSFunction",
            "method": "initialize",
            "inputData": {
                "elementRefList": "{{parameters.elementRefList}}"
            },
            "outputData": {
                "elementRefList": ""
            },
            "deps": "js/AwFocusInputBoxService"
        },
        "focusInput": {
            "actionType": "JSFunction",
            "method": "focusInput",
            "inputData": {
                "elementRefList": "{{data.elementRefList}}"
            },
            "deps": "js/AwFocusInputBoxService"
        }
    }
}
```

It has an onMount lifecycle hook wherein the elementRefList is accessed from `parameters` and stored on the view model. This is a temporary workaround till the time elementRefList is made consistently available to all actions. Right now, they are accessible only to the actions corresponding to the Lifecycle hooks, and hence this workaround has been done.

It also has a render lifecycle hook wherein the dependency to the file where the render function is defined is provided and an additional option called `elementRefList` is also used. This elementRefList has been used for `inputBox` which will be mapped to the input element whose DOM reference is required in the render function.


Siemens Web Framework
Unpublished work. © 2025 Siemens

2. Create a new service file called `AwFocusInputBoxService.js` which has a render function called `awFocusInputBoxRenderFunction`:

```
import AwButton from 'viewmodel/AwButtonViewModel';
import AwBreak from 'viewmodel/AwBreakViewModel';

export const awFocusInputBoxRenderFunction = ( props ) => {
    const inputElementRef = props.elementRefList.get( 'inputBox' );

    return <div>
        <input type='text' ref={inputElementRef}></input>
        <AwBreak></AwBreak>
        <AwButton size='auto' action={props.actions.focusInput}>Focus
Input</AwButton>
    </div>;
};

export const initialize = elementRefList => {
    return elementRefList;
};

export const focusInput = elementRefList => {
    const inputElementRef = elementRefList.get( 'inputBox' );
    inputElementRef.current.focus();
};
```

Together this would provide a reusable component called `AwFocusInputBox` which would have an inputBox that would be focused whenever the corresponding button is clicked.

Here are the steps to consume this component from any parent component:

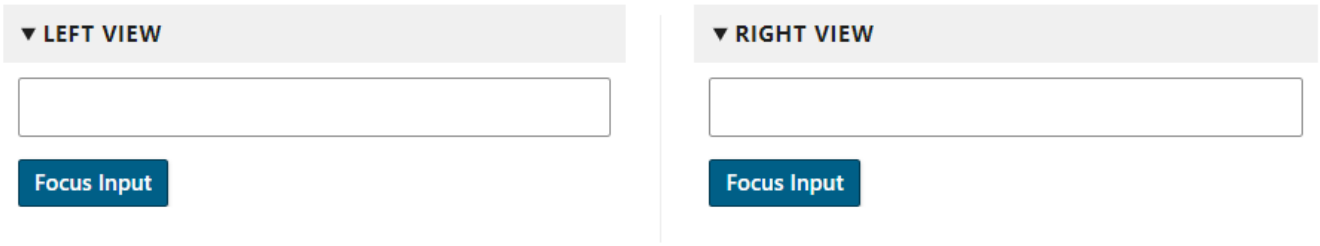
1. Create a consumer component say `AwFocusInputBoxExampleView.html` with following content:

```
<aw-panel>
    <aw-panel-body>
        <aw-row height="fill">
            <aw-column width="6">
                <aw-panel-section caption="LEFT VIEW">
                    <aw-focus-input-box></aw-focus-input-box>
                </aw-panel-section>
            </aw-column>
            <aw-separator is-vertical="true"></aw-separator>
            <aw-column width="6">
                <aw-panel-section caption="RIGHT VIEW">
                    <aw-focus-input-box></aw-focus-input-box>
                </aw-panel-section>
            </aw-column>
        </aw-row>
    </aw-panel-body>
</aw-panel>
```

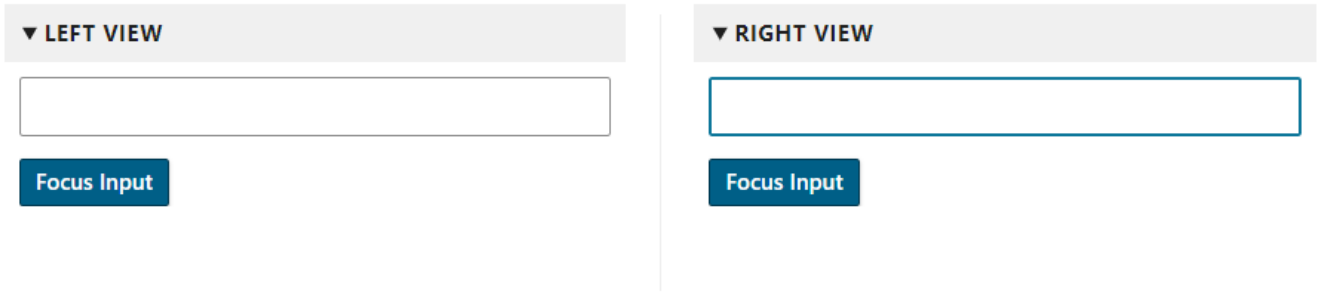
2. The corresponding consuming view model may not need any special behavior so can be empty like this:

```
{
    "schemaVersion": "1.0.0"
}
```

Once executed this will show the output as follows:




Individual components can be focused by clicking on the focus button like this:




## Creating a component integrating a third-party React component

This example features a low-level component that is using React color picker as a third-party integration. This new component will be reusable and will have the color picker behavior wherein the selected color will be rendered in a custom DIV element.

**Note**

This example features React SketchPicker, which is a third-party component based on React. Although this currently works and integrates well with Siemens Web Framework because it is based on React as well, it is an abstraction leak and we recommend integrating vanilla JavaScript third-party widgets


Siemens Web Framework
Unpublished work. © 2025 Siemens

instead. That guarantees there is no technology leak and that your component will continue to work in future versions of Siemens Web Framework, regardless of the underlying framework.

Here are the steps to be followed for creating this reusable component using a JSX render function.

1. Create a view model for this component called `AwThirdPartyColorPickerViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "render": "render"
    },
    "actions": {
        "render": {
            "method": "awThirdPartyColorPickerRenderFunction",
            "deps": "js/AwThirdPartyColorPickerService"
        }
    },
    "props": {
        "selected-color": {
            "type": "object"
        }
    }
}
```

2. Create a new service file called `AwThirdPartyColorPickerService.js` which has a render function called `awThirdPartyColorPickerRenderFunction`:

```
import AwFlexRow from 'viewmodel/AwFlexRowViewModel';
import { SketchPicker } from 'react-color';

export const awThirdPartyColorPickerRenderFunction = props => {
    const { viewModel } = props;
    const setColor = value => {
        let newColor = { ...props.selectedColor.getValue() };
        newColor.value = value.hex;
        props.selectedColor.update( newColor );
    };

    return (
    <AwFlexRow>
        <SketchPicker color={props.selectedColor.value}
onChangeComplete={color => setColor( color )}></SketchPicker>
    </AwFlexRow>
    );
};
```

Together this would provide a reusable component called AwThirdPartyColorPicker which would have a color picker and a flex column rendered in the selected color. The default color is initialized as black.

Notice the render function has the entire view model available on the props that it receives. `props.selectedColor` is received from the consuming component and on selecting a different value (onChangeComplete on SketchPicker), the callback function is invoked which is going to update the parent state received by the AwThirdPartyColorPicker component. This is the same color that will be displayed on SketchPicker component.

Here are the steps to consume this component from any parent component:

1. Create a consumer component say `AwThirdPartyColorPickerExampleView.html` with the following content:

```
 <aw-panel>
     <aw-panel-body class="sw-row">
         <aw-panel-section class="h-12 w-6" caption="Left">
             <aw-flex-row height="60f">
                 <aw-flex-column width="12">
                     <aw-third-party-color-picker class="h-6 justify-
center" selected-color="fields.selectedColorL"></aw-third-party-color-
picker>
                     <br></br>
                     <aw-flex-row class="h-6"
color="{{fields.selectedColorL.value}}">
                     </aw-flex-row>
                 </aw-flex-column>
             </aw-flex-row>
         </aw-panel-section>
         <aw-panel-section class="h-12 w-6" caption="Right">
             <aw-flex-row height="60f">
                 <aw-flex-column width="12">
                     <aw-third-party-color-picker class="h-6 justify-
center" selected-color="fields.selectedColorR"></aw-third-party-color-
picker>
                     <br></br>
                     <aw-flex-row class="h-6"
color="{{fields.selectedColorR.value}}">
                     </aw-flex-row>
                 </aw-flex-column>
             </aw-flex-row>
         </aw-panel-section>
     </aw-panel-body>
 </aw-panel>
```

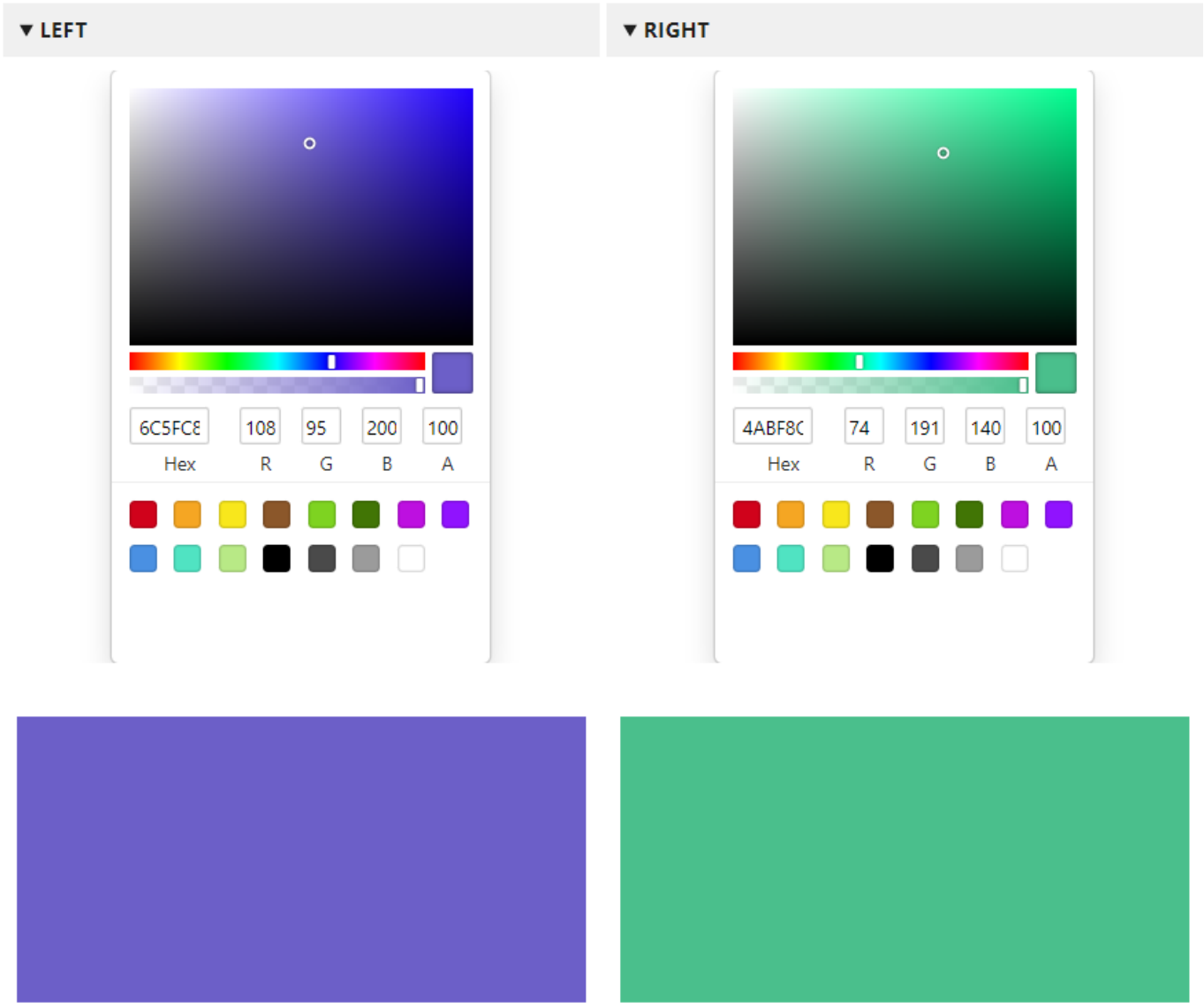
2. The corresponding consuming view model will need to define the initial colors using **atomic data**:

```
 {
     "schemaVersion": "1.0.0",
     "data": {
         "selectedColorL": {
             "initialValues": {
                 "value": "#6c5fc8"
             },
             "meta": {}
```

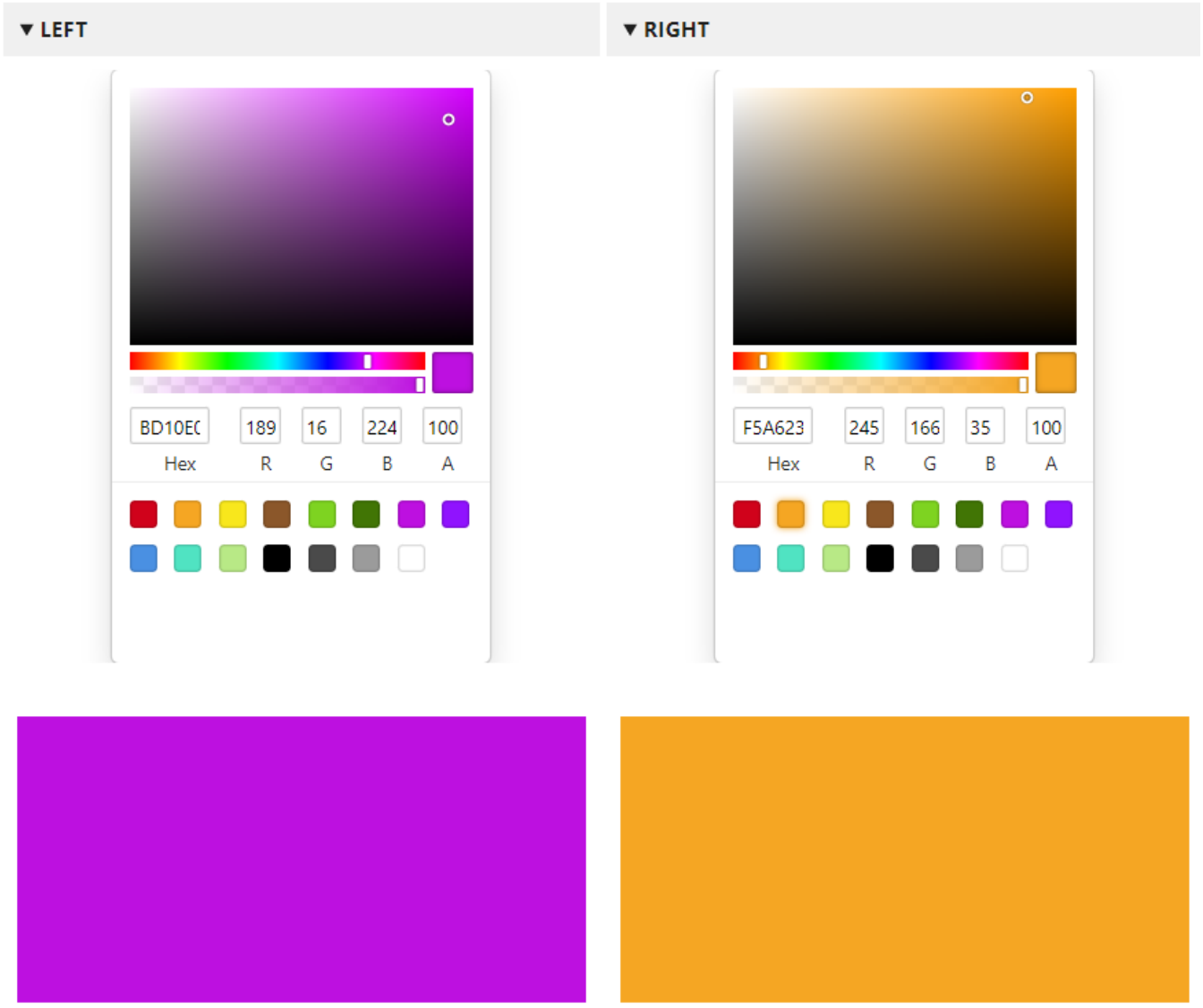

Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        },
        "selectedColorR": {
            "initialValues": {
                "value": "#4abf8c"
            },
            "meta": {}
        }
    }
}
```

Once executed this will show the output as follows:




On selecting a different color, the same would be shown in SketchPicker element as well as the column:




## Creating a component integrating a third-party vanilla JavaScript component

This example features a low-level component that is using the **Toast UI color picker** as third-party integration. This new component will be reusable and will have the color picker behavior wherein the color to be selected will be sent and rendered in the color picker widget.

Here are the steps to be followed for creating this reusable component using a JSX render function.


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. Create a view model for this component called `AwNativeThirdPartyColorPickerViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "lifecycleHooks": {
        "onMount": "initialize",
        "render": "render",
        "onUpdate": {
            "observers": [ "props.color" ],
            "action": "setColor"
        }
    },
    "actions": {
        "render": {
            "method": "awNativeThirdPartColorPickerRenderFunction",
            "deps": "js/AwNativeThirdPartyColorPickerService",
            "options": {
                "elementRefList": [ "colorPickerContainer" ]
            }
        },
        "initialize": {
            "actionType": "JSFunction",
            "method": "initialize",
            "inputData": {
                "elementRefList": "{{parameters.elementRefList}}"
            },
            "outputData": {
                "elementRefList": ""
            },
            "deps": "js/AwNativeThirdPartyColorPickerService"
        },
        "setColor": {
            "actionType": "JSFunction",
            "method": "setColor",
            "inputData": {
                "colorPickerInstance": "{{data.colorPickerInstance}}",
                "color": "{{props.color}}"
            },
            "deps": "js/AwNativeThirdPartyColorPickerService"
        }
    },
    "props":{
        "color": {
            "type": "string"
        }
    }
}
```


Unpublished work. © 2025 Siemens

2. Create a new service file called `AwNativeThirdPartyColorPickerService.js` which has a render function called `awNativeThirdPartyColorPickerRenderFunction`:

```
 import colorPicker from 'tui-color-picker';

 export const awNativeThirdPartColorPickerRenderFunction = ( props ) =>
{
     const colorPickerContainerRef =
props.elementRefList.get( 'colorPickerContainer' );

     if( colorPickerContainerRef && !
props.viewModel.data.colorPickerInstance &&
colorPickerContainerRef.current ) {
         let colorPickerInstance = colorPicker.create( {
             container: colorPickerContainerRef.current,
             color: '#FF0000'
         } );
         props.viewModel.dispatch( { path: 'data.colorPickerInstance',
value: colorPickerInstance } );
     }

     return <div ref = { colorPickerContainerRef }> </div>;
 };

 export const initialize = elementRefList => {
     return elementRefList;
 };

 export const setColor = ( colorPickerInstance, color ) => {
     colorPickerInstance.setColor( color );
 };
```

Together, these are used to implement a reusable component called `AwNativeThirdPartyColorPicker` that would have a color picker rendered and the color value passed as a prop. The default color is initialized as red.

Here are the steps to consume this component from any parent component:

1. Create an example view called `AwNativeThirdPartyColorPickerExampleView.html` with following content:

```
 <aw-panel>
     <aw-panel-body>
         <aw-row height="fill">
             <aw-column width="6">
                 <aw-panel-section caption="LEFT VIEW">
                     <aw-radiobutton prop="data.colorL"
list="data.colorToSetRadioValues.dbValue"></aw-radiobutton>
                 <aw-native-third-party-color-picker
color="{{data.colorL.dbValue}}"></aw-native-third-party-color-picker>
                 </aw-panel-section>
```

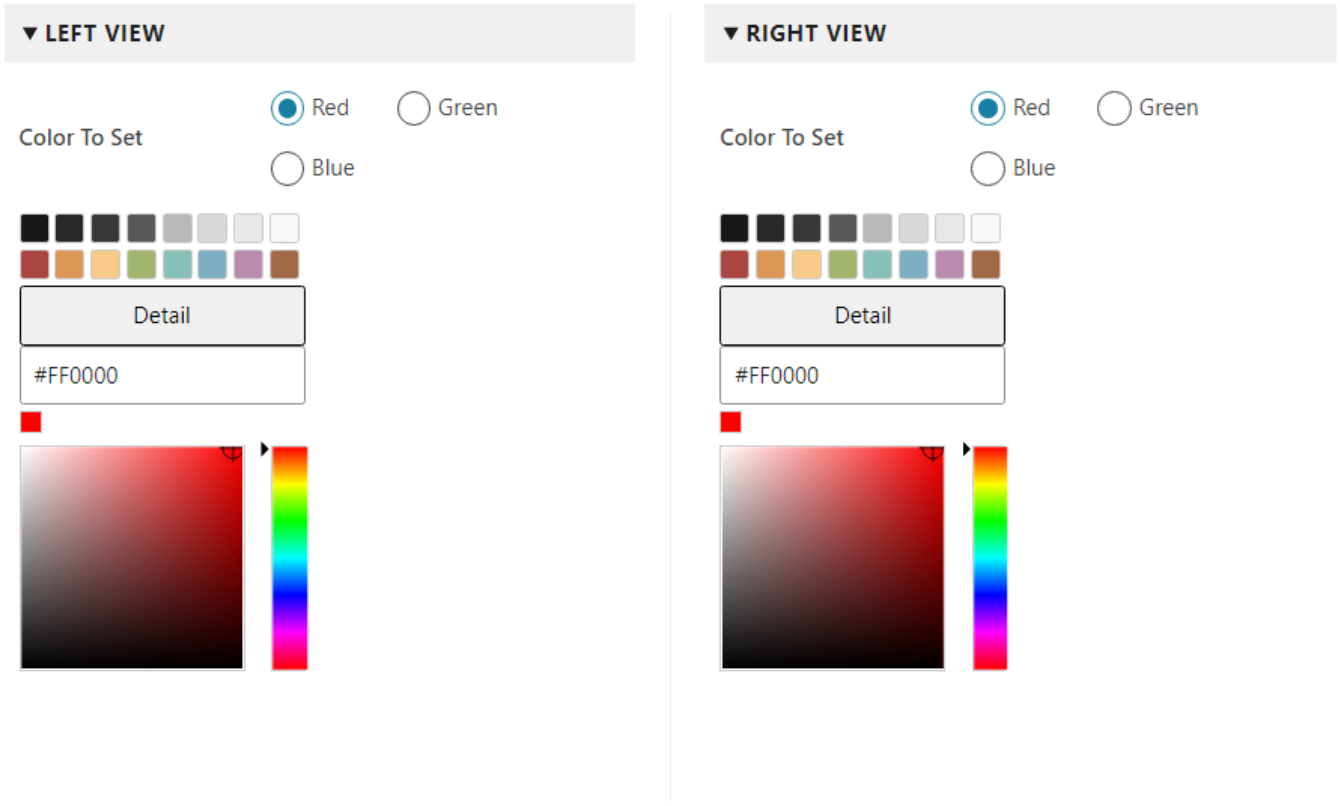

Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            </aw-column>
            <aw-separator is-vertical="true"></aw-separator>
            <aw-column width="6">
                <aw-panel-section caption="RIGHT VIEW">
                    <aw-radiobutton prop="data.colorR"
list="data.colorToSetRadioValues.dbValue"></aw-radiobutton>
                    <aw-native-third-party-color-picker
color="{{data.colorR.dbValue}}"></aw-native-third-party-color-picker>
                </aw-panel-section>
            </aw-column>
        </aw-row>
    </aw-panel-body>
 </aw-panel>
```

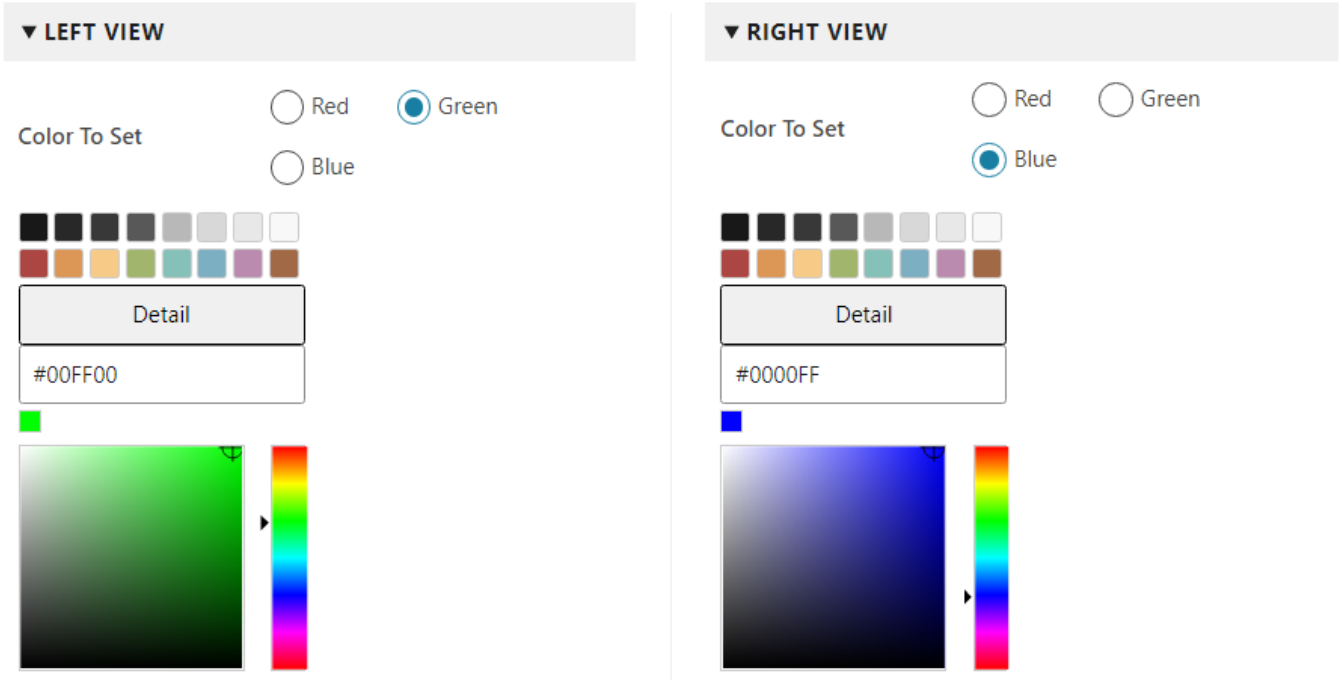
2. Create the corresponding view model containing a list of colors:

```
{
    "schemaVersion": "1.0.0",
    "data":{
        "colorL": {
            "displayName": "Color To Set",
            "type": "STRING",
            "isEditable": true,
            "dbValue": "#FF0000"
        },
        "colorR": {
            "displayName": "Color To Set",
            "type": "STRING",
            "isEditable": true,
            "dbValue": "#FF0000"
        },
        "colorToSetRadioValues":{
            "type": "STRING",
            "dbValue": [
                {
                    "propDisplayValue": "Red",
                    "propInternalValue": "#FF0000"
                },
                {
                    "propDisplayValue": "Green",
                    "propInternalValue": "#00FF00"
                },
                {
                    "propDisplayValue": "Blue",
                    "propInternalValue": "#0000FF"
                }
            ]
        }
    }
}
```

Once executed, this example will be rendered like this:




When you select a different color, it will be displayed in the color picker:





Siemens Web Framework
Unpublished work. © 2025 Siemens

# Interaction Patterns between Components

A **Component** is a reusable, composable, debuggable unit that has a specific behavior. It typically has a **render function** and a **view model**.

A business usecase would require various components to interact with each other.

The following tutorials provide some simple but useful examples on various patterns in which these components can interact with each other:

- **Component Interaction using Global Configuration**
- **Component Interaction using Subscribe/Publish of Events**
- **Component Interaction using Sync Strategy**
- **Component Interaction using Inverse Flow**
- **Component Interaction Comparison**

> **Tip**
>
> Please follow **Pre-requisite Component Creation** first to create the components between which the interaction (say Master Details) needs to be implemented.

## Example Components

This tutorial explains how to create the example components which will be used to explain the various Interaction Patterns in which they can be made to interact with each other.

This example will explain how to implement the two components between which Master Details relationship will be exhibited using various **interaction patterns**

Here are the steps to be followed for creating the first component. It is a component which uses `aw-list` to present a list of movies.

1. Create a new component called `AwMoviesView.html` which has the usage of AwList

```
<aw-panel>
    <aw-panel-body>
        <aw-panel-section>
            <aw-list dataprovider="data.dataProviders.movieSearch">
                <aw-default-cell vmo="item"> </aw-default-cell>
            </aw-list>
        </aw-panel-section>
```

```
    </aw-panel-body>
</aw-panel>
```

2. Add the corresponding view model for this component, called `AwMoviesViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "selectionData": {
            "initialValues": {},
            "meta": {}
        }
    },
    "dataProviders": {
        "movieSearch": {
            "action": "search",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "inputData": {
                "selectionData": "{{fields.selectionData}}"
            }
        }
    },
    "actions": {
        "reveal": {
            "actionType": "dataProvider",
            "method": "movieSearch"
        },
        "search": {
            "actionType": "RESTService",
            "serviceName": "",
            "method": "GET",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/afx/movies"
                }
            },
            "dataParsers": [
                {
                    "id": "myMovieDataParser",
                    "responseObjs": "data.search"
                }
            ],
            "outputData": {
                "totalFound": "data.totalResults",
                "searchResults": "data.search"
            }
        }
    },
    "dataParseDefinitions": {
        "myMovieDataParser": {
            "hasThumbnail": false,
            "typeIconURL": "{{response.Poster}}",
            "cellHeader1": "{{response.Title}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "cellHeader2": "{{response.Year}}",
            "cellProperties": {
                "Type": {
                    "key": "Type",
                    "value": "{{response.Type}}"
                }
            },
            "props": {
                "movieId": {
                    "name": "movieId",
                    "displayName": "movieId",
                    "value": "{{response.id}}",
                    "displayValue": "{{response.id}}",
                    "type": "STRING"
                }
            }
        }
    }
}
```

Once executed, this will show the output as follows:

1. Create a new component called `AwMovieDetailsView.html` which has the usage of AwList

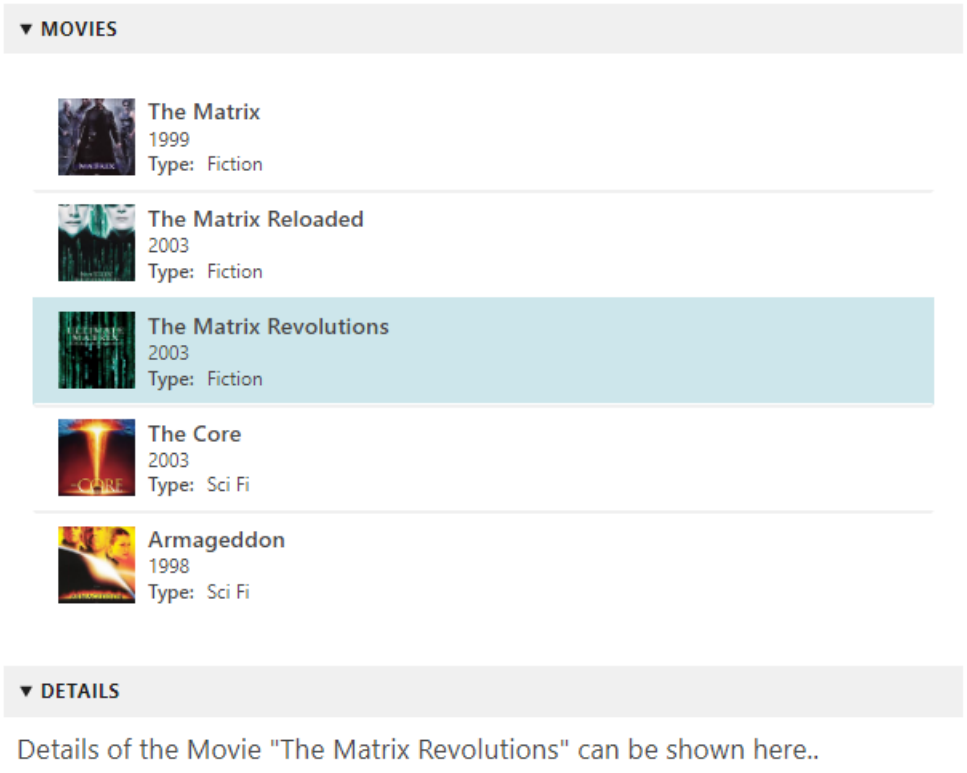
```
 <div visible-when="conditions.selectedObjectIsAvailable">
     <h4>Details of the Movie
"{ctx.movieSearch.selectedObject.cellHeader1}" can be shown here..</h4>
 </div>
```

2. Add the corresponding view model for this component, called `AwMovieDetailsViewModel.json`:

```
{
    "schemaVersion": "1.0.0",
    "ctx": {
        "movieSearch.selectedObject.cellHeader1": null
    },
    "conditions": {
        "selectedObjectIsAvailable": {
            "expression": "ctx.movieSearch.selectedObject.cellHeader1"
        }
    }
}
```

Now that the two independent components are created, the tutorials will show establishing master details between these two components using various **component interaction patterns**.

Once those tutorials are done, this is how the output will be shown:




Let's go back to the **component interaction patterns**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Component Interaction using Global Configuration

This example features an interaction between two components - Movie and MovieDetails using the Global Configuration.

The interaction required is that when there is a movie selected in the first component which shows a list of movies, the details/information about that selected movie should be displayed in the second component called MovieDetails.

Before we can implement this Master Details interaction pattern between these components, let's follow **tutorial** to create these two independent components.

Now, that the components are created; here are the steps to be followed for implementing the interaction between them using Global Configuration:

1. Create a copy of `AwMoviesView.html` and `AwMoviesViewModel.json` and name them as `AwMoviesWithGlobalConfigurationView.html` and `AwMoviesWithGlobalConfigurationViewModel.json` respectively.
2. Create a copy of `AwMovieDetailsView.html` and `AwMovieDetailsViewModel.json` and name them as `AwMovieDetailsWithGlobalConfigurationView.html` and `AwMovieDetailsWithGlobalConfigurationViewModel.json` respectively.
3. Create a new component called `AwParentWithGlobalConfigurationView.html` which has the usage of components created above between which the interaction is required to be implemented.

```
 <aw-panel>
     <aw-panel-body>
         <aw-panel-section caption="MOVIES">
             <aw-movies-with-global-configuration></aw-movies-with-
global-configuration>
             </aw-panel-section>
         <aw-panel-section caption="DETAILS">
             <aw-movie-details-with-global-configuration></aw-movie-
details-with-global-configuration>
         </aw-panel-section>
     </aw-panel-body>
 </aw-panel>
```

4. Add the corresponding view model for this component, called `AwParentWithGlobalConfigurationViewModel.json`:

```
{
    "schemaVersion": "1.0.0"
}
```

5. Modify `AwMoviesWithGlobalConfigurationViewModel.json` to add an onUpdate hook on `selectionData`:

```
{
    "lifecycleHooks": {
        "onUpdate": [
            {
                "action": "handleSelectionChange",
                "observers": [
                    "viewModel.atomicData.selectionData"
                ]
            }
        ]
    }
}
```

6. Add the action called `handleSelectionChange` in which update the **Global Configuration** i.e. AppCtx using updateCtx API of `appCtxService`:

```
{
    "actions":{
        "handleSelectionChange": {
            "actionType": "JSFunction",
            "method": "updateCtx",
            "inputData": {
                "name": "movieSearch.selectedObject",
                "value": "{{data.selectionData.selected[0]}}"
            },
            "deps": "js/appCtxService"
        }
    }
}
```

This code is going to put the object selected in Component `AwMoviesWithGlobalConfiguration` on the Global AppCtx.

7. Modify `AwMovieDetailsWithGlobalConfigurationViewModel.json` and `AwMovieDetailsWithGlobalConfigurationView.html` to listen to ctx updates and render one of the property on the object received on the Global AppCtx:

```
{
    "schemaVersion": "1.0.0",
    "ctx": {
        "movieSearch.selectedObject.cellHeader1": null
    },
    "conditions": {
        "selectedObjectIsAvailable": {
            "expression": "ctx.movieSearch.selectedObject.cellHeader1"
        }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

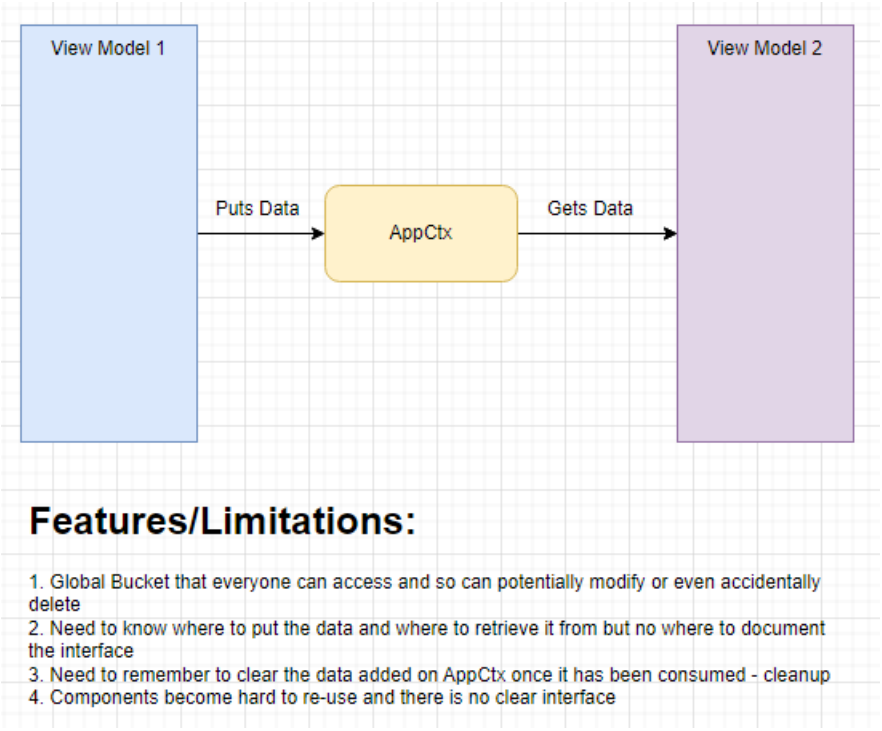
```
    }
}
```

```
<div visible-when="conditions.selectedObjectIsAvailable">
    <h4>Details of the Movie
"{ctx.movieSearch.selectedObject.cellHeader1}" can be shown here..</h4>
</div>
```

Now, on execution, when you select something from the Movies component, you would see that the selected object is shared or sent to the Details component. And this sharing is achieved via Global Configuration i.e. AppCtx.

## Summary

Two components can interact with each other using a common global configuration object like AppCtx. One of the components can put information to be shared on the global bucket and the other component can retrieve it from there.




### When to use this Interaction Pattern

- When there are simple components which do not need re-use.
- When the use case is simple as this is the easiest pattern to share data stored in a common place.
- Is useful when the components are far apart in the component hierarchy.

### Limitations with this Interaction Pattern

- There is a tight coupling between the two components. Both have to handshake on the name of the global context which needs to be accessed for sharing of information. Example - `ctx.movieSearch.selectedObject` in the example is an interface/contract that both the components need to be aware of.
- Reuse is difficult. Imagine the components need to be re-used such that it can show information coming from multiple source components, in this case, using a global context does not work.
- We need to be careful with the context names conflicts.

## Component Interaction using Subscribe/Publish of Events

This example features an interaction between two components - Movie and MovieDetails using the Subscribe/Publish of events.


Siemens Web Framework
Unpublished work. © 2025 Siemens

The interaction required is that when there is a movie selected in the first component which shows a list of movies, the details/information about that selected movie should be displayed in the second component called MovieDetails.

Before we can implement this Master Details interaction pattern between these components, let's follow **tutorial** to create these two independent components.

Now, that the components are created; here are the steps to be followed for implementing the interaction between them using subscription/publish of events:

1. Create a copy of `AwMoviesView.html` and `AwMoviesViewModel.json` and name them as `AwMoviesWithEventsView.html` and `AwMoviesWithEventsViewModel.json` respectively.
2. Create a copy of `AwMovieDetailsView.html` and `AwMovieDetailsViewModel.json` and name them as `AwMovieDetailsWithEventsView.html` and `AwMovieDetailsWithEventsViewModel.json` respectively.
3. Create a new component called `AwParentWithEventsView.html` which has the usage of components created above between which the interaction is required to be implemented.

```
 <aw-panel>
     <aw-panel-body>
         <aw-panel-section caption="MOVIES">
             <aw-movies-with-events></aw-movies-with-events>
         </aw-panel-section>
         <aw-panel-section caption="DETAILS">
             <aw-movie-details-with-events></aw-movie-details-with-
events>
         </aw-panel-section>
     </aw-panel-body>
 </aw-panel>
```

4. Add the corresponding view model for this component, called `AwParentWithEventsViewModel.json`:

```
{
    "schemaVersion": "1.0.0"
}
```

5. Modify `AwMoviesWithEventsViewModel.json` to add an onUpdate hook on `selectionData`

```
{
    "lifecycleHooks": {
        "onUpdate": [
            {
                "action": "handleSelectionChange",
                "observers": [
                    "viewModel.atomicData.selectionData"
                ]
            }
```

```
            ]
        }
}
```

6. Also add the action called `handleSelectionChange` in which fire an **Event** with the selected object information populated in the eventData.

```
{
    "actions":{
        "handleSelectionChange": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [
                    {
                        "name": "objectSelectedInMovieSearch",
                        "eventData": {
                            "selectedObject":
"{{data.selectionData.selected[0]}}"
                        },
                        "excludeLocalDataCtx": true
                    }
                ]
             }
        }
    }
}
```

This code is going to put the object selected in Component `AwMoviesWithEvent` on the EventBus.

7. Modify `AwMovieDetailsWithEventViewModel.json` and `AwMovieDetailsWithEventView.html` to subscribe to this event and render one of the property on the object received on the eventData.

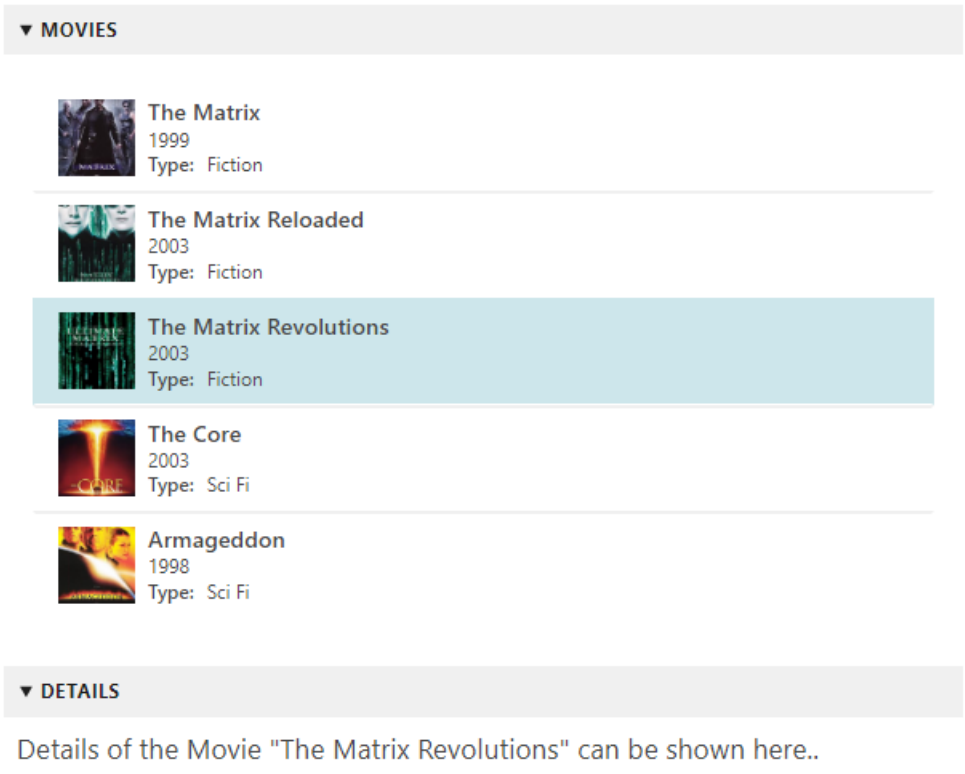
```
{
    "schemaVersion": "1.0.0",
    "data": {
        "objectToShowDetailsFor": {}
    },
    "actions": {
        "showDetailsOfSelectedObject": {
            "actionType": "JSFunction",
            "method": "updateObjectToShowDetailsFor",
            "inputData": {
                "selectedObject": "{{data.eventData.selectedObject}}"
            },
            "outputData": {
                "objectToShowDetailsFor": ""
            },
            "deps": "js/awMoviesDetailsHelperService"
        }
    },
    "onEvent": [
```


Unpublished work. © 2025 Siemens

```
        {
            "eventId": "objectSelectedInMovieSearch",
            "action": "showDetailsOfSelectedObject",
            "cacheEventData": true
        }
    ],
    "conditions": {
        "selectedObjectIsAvailable": {
            "expression": "data.objectToShowDetailsFor.cellHeader1"
        }
    }
}
```

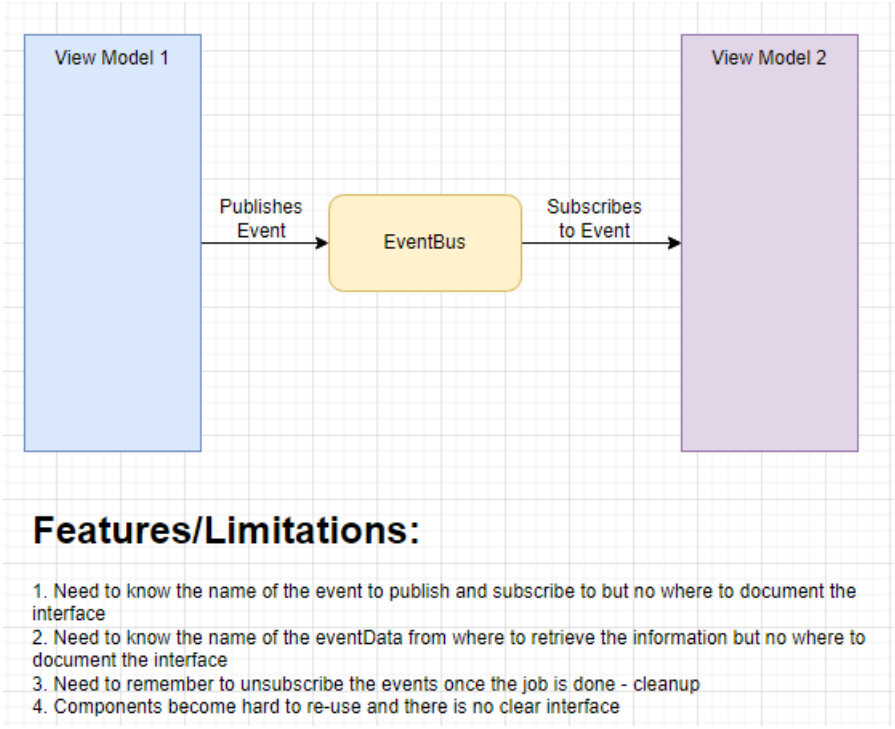
```
<div visible-when="conditions.selectedObjectIsAvailable">
    <h4>Details of the Movie
"{data.objectToShowDetailsFor.cellHeader1}" can be shown here..</h4>
</div>
```

Now, on execution, when you select something from the Movies component, you would see that the selected object is shared or sent to the Details component. And this sharing is achieved via Subscription and Publish of events.



## Summary

Two components can interact with each other using a events. One of the components can publish an event with the required information to be shared and the other component can subscribe to the event and receive the shared information from the event data.




## When to use this Interaction Pattern

- When there are simple components which do not need re-use.
- When the use case is simple to notify that something happened without sharing data.
- Is useful when the components are far apart in the component hierarchy.

## Limitations with this Interaction Pattern

- There is a tight coupling between the two components. Both have to handshake on the name of the event and eventData which needs to be accessed for sharing of information. Example - `objectSelectedInMovieSearch` in the example is an interface/contract that both the components need to be aware of.
- Reuse is difficult. Imagine the components need to be re-used such that it can show information coming from multiple source components, in this case, using events do not work good.
- Not suitable to keep data in sync as the data gets lost if the event is not handled properly.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Component Interaction using Sync Strategy

This example features an interaction between two components - Movie and MovieDetails using **Sync Strategy**.

The interaction required is that when there is a movie selected in the first component which shows a list of movies, the details/information about that selected movie should be displayed in the second component called MovieDetails.

Before we can implement this Master Details interaction pattern between these components, let's follow **tutorial** to create these two independent components.

Now, that the components are created; here are the steps to be followed for implementing the interaction between them using Sync Strategy:

1. Create a copy of `AwMoviesView.html` and `AwMoviesViewModel.json` and name them as `AwMoviesWithSyncStrategyView.html` and `AwMoviesWithSyncStrategyViewModel.json` respectively.
2. Create a copy of `AwMovieDetailsView.html` and `AwMovieDetailsViewModel.json` and name them as `AwMovieDetailsWithSyncStrategyView.html` and `AwMovieDetailsWithSyncStrategyViewModel.json` respectively.
3. Create a new component called `AwParentWithSyncStrategyView.html` which has the usage of components created above between which the interaction is required to be implemented.

```
 <aw-panel>
     <aw-panel-body>
         <aw-panel-section caption="MOVIES">
             <aw-movies-with-sync-strategy></aw-movies-with-sync-
strategy>
         </aw-panel-section>
         <aw-panel-section caption="DETAILS">
             <aw-movie-details-with-sync-strategy></aw-movie-details-
with-sync-strategy>
         </aw-panel-section>
     </aw-panel-body>
 </aw-panel>
```

4. Add the corresponding view model for this component, called `AwParentWithSyncStrategyViewModel.json`:

```
{
    "schemaVersion": "1.0.0"
}
```

5. Modify `AwMoviesWithSyncStrategyViewModel.json` to define an input port with name `moviesSelection`

```
{
    "schemaVersion": "1.0.0",
    "ports": {
        "moviesSelection": {
            "syncObject": {}
        }
    }
}
```

6. Modify `AwMoviesWithSyncStrategyViewModel.json` to add an onUpdate hook on `selectionData`

```
{
    "lifecycleHooks": {
        "onUpdate": [
            {
                "action": "handleSelectionChange",
                "observers": [
                    "viewModel.atomicData.selectionData"
                ]
            }
        ]
    }
}
```

7. Add the action called `handleSelectionChange` in which invoke the **Sync Action** with the selected object information populated in the syncObject.

```
{
    "actions":{
        "handleSelectionChange": {
            "actionType": "Sync",
            "inputData": {
                "port": "moviesSelection",
                "syncObject": "{{data.selectionData.selected[0]}}"
            }
        }
    }
}
```

This code is going to put the object selected in Component `AwMoviesWithSyncStrategy` on the target port defined in step 9 below.


Siemens Web Framework
Unpublished work. © 2025 Siemens

8. Modify `AwMovieDetailsWithSyncStrategyViewModel.json` and `AwMovieDetailsWithSyncStrategyView.html` to define the target port and a corresponding onChangeAction. Add an additional `awMoviesDetailsHelperService.js` to add the glue code.

```
{
    "schemaVersion": "1.0.0",
    "ports": {
        "movieDetailsInput": {
            "onChangeAction": "showDetailsOfSelectedObject",
            "syncObject": {}
        }
    },
    "data": {
        "objectToShowDetailsFor": {}
    },
    "actions": {
        "showDetailsOfSelectedObject": {
            "actionType": "JSFunction",
            "method": "updateObjectToShowDetailsFor",
            "inputData": {
                "selectedObject":
"{{ports.movieDetailsInput.syncObject}}"
            },
            "outputData": {
                "objectToShowDetailsFor": ""
            },
            "deps": "js/awMoviesDetailsHelperService"
        }
    },
    "conditions": {
        "selectedObjectIsAvailable": {
            "expression": "data.objectToShowDetailsFor.cellHeader1"
        }
    }
}
```

```
<div visible-when="conditions.selectedObjectIsAvailable">
    <h4>Details of the Movie
"{data.objectToShowDetailsFor.cellHeader1}" can be shown here..</h4>
</div>
```

```
export const updateObjectToShowDetailsFor = selectedObject => {
    return selectedObject;
};
```

9. Define a `syncStrategy.json` at module level to wed the two ports defined on these two components together

```
{
    "schemaVersion": "1.0.0",
```

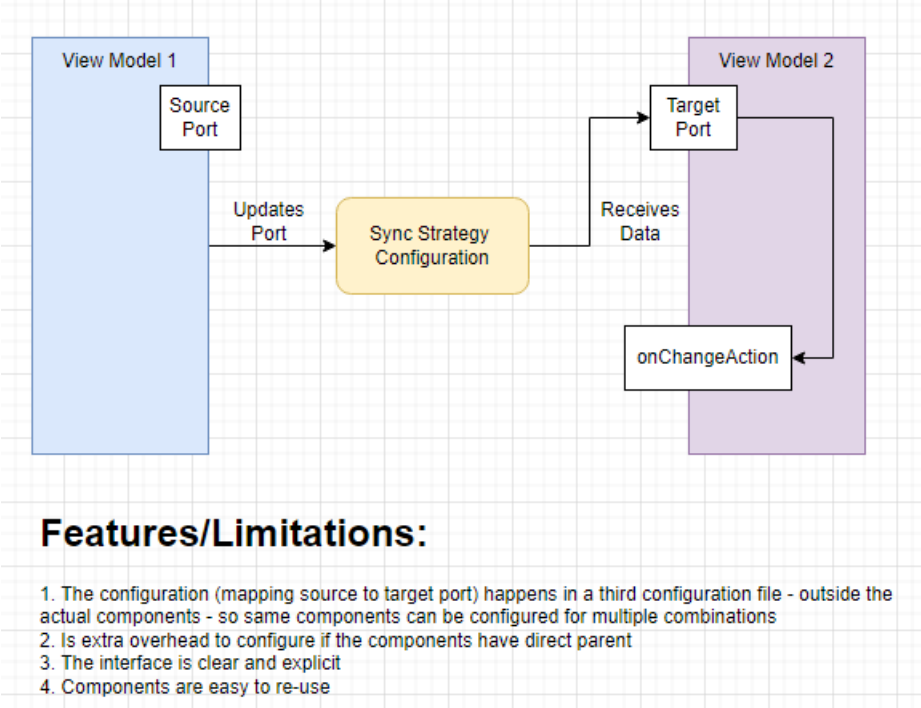
```
    "syncStrategies": {
        "compInteractionSyncStrategyObject": {
            "source": {
                "view": "AwMoviesWithSyncStrategy",
                "port": "moviesSelection"
            },
            "target": {
                "view": "AwMovieDetailsWithSyncStrategy",
                "port": "movieDetailsInput"
            }
        }
    },
    "syncStrategyHandlers":{
        "compInteractionMoviesHandler": {
            "id": "compInteractionSyncStrategyObject",
            "activeWhen": true
        }
    }
}
```

Now, on execution, when you select something from the Movies component, you would see that the selected object is shared or sent to the Details component. And this sharing is achieved via Sync Strategy.

Siemens Web Framework
Unpublished work. © 2025 Siemens

### Summary

Two components can interact with each other using Sync Strategy. One of the components acts as a source and puts the shared information on the input port and the other component can retrieve it on the target port if sync strategy is configured between them.




### When to use this Interaction Pattern

- When there are components which need re-use.
- When there are components which do not have a common parent i.e. components are distant siblings.

### Limitations with this Interaction Pattern

- There is a performance cost associated with configuring sync strategy between two components if they anyway have a direct parent which can share data directly between the siblings instead of routing it via a third configuration like sync strategy.

## Component Interaction using Inverse Flow

This example features an interaction between two components - Movie and MovieDetails using **Inverse Flow**.

The interaction required is that when there is a movie selected in the first component which shows a list of movies, the details/information about that selected movie should be displayed in the second component called MovieDetails.

Before we can implement this Master Details interaction pattern between these components, let's follow **tutorial** to create these two independent components.

Now, that the components are created; here are the steps to be followed for implementing the interaction between them using Inverse Flow:

1. Create a copy of `AwMoviesView.html` and `AwMoviesViewModel.json` and name them as `AwMoviesWithInverseFlowView.html` and `AwMoviesWithInverseFlowViewModel.json` respectively.
2. Create a copy of `AwMovieDetailsView.html` and `AwMovieDetailsViewModel.json` and name them as `AwMovieDetailsWithInverseFlowView.html` and `AwMovieDetailsWithInverseFlowViewModel.json` respectively.
3. Create a new component called `AwParentWithInverseFlowView.html` which has the usage of components created above between which the interaction is required to be implemented. Notice the props being sent to both the child components.

```
 <aw-panel>
     <aw-panel-body>
         <aw-panel-section caption="MOVIES">
             <aw-movies-with-inverse-flow parent-
context="data.parentContext"></aw-movies-with-inverse-flow>
         </aw-panel-section>
         <aw-panel-section caption="DETAILS">
             <aw-movie-details-with-inverse-
flow input-object="data.parentContext.selectedObj"></aw-movie-details-
with-inverse-flow>
         </aw-panel-section>
     </aw-panel-body>
 </aw-panel>
```

4. Add the corresponding view model for this component, called `AwParentWithInverseFlowViewModel.json` which defines the atomic data:

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "parentContext": {
            "initialValues": {
                "selectedObj": {}
            },
            "meta": {}
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

5. Modify `AwMoviesWithInverseFlowViewModel.json` to add a `prop` section to receive the parentContext sent by the parent component

```
{
    "props": {
        "parent-context": {
            "type": "object"
        }
    }
}
```

6. Modify `AwMoviesWithInverseFlowViewModel.json` to add an onUpdate hook on `selectionData`

```
{
    "lifecycleHooks": {
    "onUpdate": [
        {
            "action": "handleSelectionChange",
            "observers": [
                "viewModel.atomicData.selectionData"
            ]
        }
    ]}
}
```

7. Add the action called `handleSelectionChange` in which invoke the update API on the **Atomic Data** on the data received in props from the parent component.

```
{
    "actions":{
        "handleSelectionChange": {
            "actionType": "JSFunction",
            "method": "updateParentContext",
            "inputData": {
                "selectedObject": "{{data.selectionData.selected[0]}}",
                "parentContext": "{{props.parentContext}}"
            },
            "deps": "js/awMoviesDetailsHelperService"
        }
    }
}
```

8. Add an additional `awMoviesDetailsHelperService.js` to add the glue code to update the parent context received in the props section of the Movies component.

```
export const updateParentContext = ( selectedObject, parentContext ) =>
{
    if ( selectedObject ) {
        let localParentContext = { ... parentContext };
        localParentContext.selectedObj = selectedObject;
```

```
        return parentContext.update( localParentContext );
    }
};
```

This code is going to put the object selected in Component `AwMoviesWithInverseFlow` on the parentContext i.e. the object received from the parent component.

9. Modify `AwMovieDetailsWithInverseFlowViewModel.json` and `AwMovieDetailsWithInverseFlowView.html` to consume the props received from the parent component.

```
{
    "schemaVersion": "1.0.0",
    "conditions": {
        "selectedObjectIsAvailable": {
            "expression": "props.inputObject.cellHeader1"
        }
    },
    "props": {
        "input-object": {
            "type": "object"
        }
    }
}
```

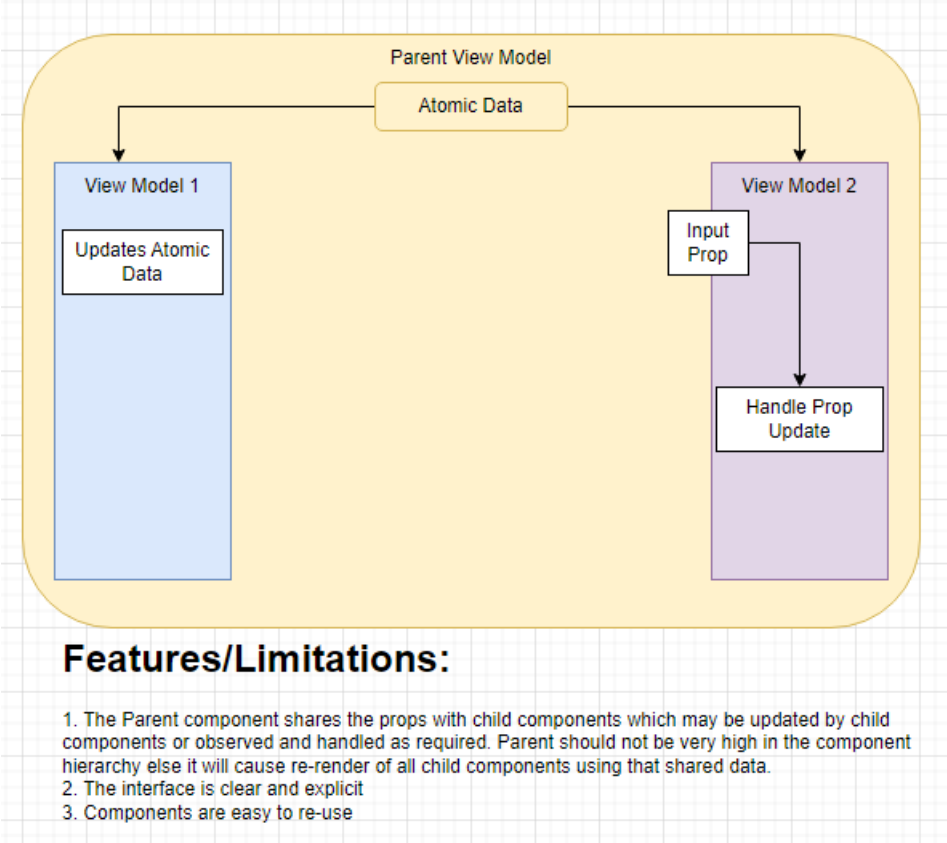
```
<div visible-when="conditions.selectedObjectIsAvailable">
    <h4>Details of the Movie "{props.inputObject.cellHeader1}" can be
shown here..</h4>
</div>
```

Now, on execution, when you select something from the Movies component, you would see that the selected object is shared or sent to the Details component. And this sharing is achieved via Inverse Flow.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Summary

Two components can interact with each other using Inverse Flow. The parent component creates an atomic data and shares with all the child components. The component which needs to share the information, updates the parent's atomic data and the component which needs to retrieve the information reads it from the atomic data.




### When to use this Interaction Pattern

- When there are components which need re-use. The interface of these components becomes extremely clear and explicit.
- When there are components which have a common parent i.e. components are close/siblings.
- When there are components which need to share the state hoisted up to their lowest common parent.

### Limitations with this Interaction Pattern

- We need to be careful in hoisting the state to lowest common parent. There is performance cost associated with sharing the common state way too up in the hierarchy.

## Comparison between various Component Interaction Patterns

Here is a table summarizing the various component interaction patterns:

| Interaction Pattern | Depends on | Facilitates Re-use | Component Interface |
|---|---|---|---|
| Using Global Configuration | AppCtx | No | Implicit or hidden |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| Using Subscribe/Publish of events | Events | No | Implicit or hidden |
| Using Sync Strategy | SyncStrategy Configuration | Yes | Explicit or clear |
| Using Inverse Flow | Props and Atomic data | Yes | Explicit or clear |

# Building Custom Pages

SWF provides various **Page Level Components** which can be used as a template to build your custom pages. These page level components will bring in certain wireframe/layout for free and more customizations can be added on top of it.

The following tutorials provide some simple but useful examples on various page level components which can be used to build your application pages:

- **Building pages using Common Location**
- **Building sub-pages using Common Location**
- **Building lists using Standard Sublocation**
- **Building tables using Standard Sublocation**
- **Building reusable pages using Standard Sublocation**
- **Building details pages using Standard Sublocation**

**Tip**

Please refer **Common Frame** and **Common Root** first to understand the various slots or configuration points that come with it.

**Tip**

These examples assume that there is an empty **module** called "trackerApp" already available in your example application.

## Setup mock server for Tracking App

1. Create a bootstrap.json in your app (parallel to module.json) with following contents:

```
{
    "import": [
        "js/indexDb"
    ],
```

```
    "postInit": [
        "import('js/indexDb').then(function( indexDb )
{ indexDb.initialize('trackerAppDB',['issue', 'comments']);});"
    ]
}
```

1. Create a file called indexDb.js with following contents in src\js folder:

```
let db;
export const initialize = ( databaseName, tableNames ) => {
    let request = indexedDB.open( databaseName, 1 );

    request.onupgradeneeded = function( event ) {
        db = event.target.result;
        for( let idx in tableNames ){
            db.createObjectStore( tableNames[idx], { keyPath: 'uid',
autoIncrement: true } );
        }
    };

    request.onsuccess = function( event ) {
        db = event.target.result;

        db.getObjectStore = storeName => {
            const transaction = db.transaction( [ storeName ],
'readwrite' );
            return transaction.objectStore( storeName );
        };

        db.addToStore = ( storeName, data ) => {
            const objectStore = db.getObjectStore( storeName );
            return objectStore.add( data );
        };

        db.updateStore = ( storeName, data ) => {
            const objectStore = db.getObjectStore( storeName );
            return objectStore.put( data );
        };

        db.deleteFromStore = async( storeName, index ) => {
            const objectStore = db.getObjectStore( storeName );
            let getRequest = objectStore.delete( index );
            return await processRequest( getRequest );
        };

        db.getFromStore = async( storeName, index ) => {
            const objectStore = db.getObjectStore( storeName );
            let getRequest = objectStore.get( index );
            return await processRequest( getRequest );
        };

        db.getAllFromStore = async( storeName ) => {
            const objectStore = db.getObjectStore( storeName );
            let getRequest = objectStore.getAll();
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            return await processRequest( getRequest );
        };
    };

    request.onerror = function( event ) {
        db = event.target.result;
        console.log( 'Error opening the Database' );
        db.close();
    };
};

export const getDB = () => {
    return db;
};

export const cleanup = () => {
    db.close();
};

const processRequest = async( getRequest ) => {
    return await new Promise( ( resolve, reject ) => {
        getRequest.onsuccess = function( event ) {
            resolve( event.target.result );
        };
        getRequest.onerror = function( event ) {
            reject( event.target.error );
        };
    } );
};
```

1. Create the sample issue data by creating a file called sampleIssueData.js with following content:

```
export const generateSampleIssueData = async db => {
    let result = await db.getAllFromStore( 'issue' );
    if( result.length === 0 ) {
        db.addToStore( 'issue', {
            IssueID: 'PR-239045',
            Title: 'WCAG Issue',
            Description: 'The focus does not come back on the parent
component',
            TypeIcon: 'assets/images/typeIssue48.svg',
            Type: 'Feature',
            Assignee: 'John Doe',
            ExpectedStoryPoints:1,
            ActualStoryPoints:2
        } );

        db.addToStore( 'issue', {
            IssueID: 'PR-335782',
            Title: 'Timing Issue',
            Description: 'The data is fetched even before the event is
processed',
            TypeIcon: 'assets/images/typeIssue48.svg',
            Type: 'Feature',
```

```
            Assignee: 'Sally Smith',
            ExpectedStoryPoints:2,
            ActualStoryPoints:4
        } );

        db.addToStore( 'issue', {
            IssueID: 'PR-589614',
            Title: 'Ux Issue',
            Description: 'The buttons are too big on this page',
            TypeIcon: 'assets/images/typeIssue48.svg',
            Type: 'Usability',
            Assignee: 'Micheal Jones',
            ExpectedStoryPoints:3,
            ActualStoryPoints:3
        } );

        db.addToStore( 'issue', {
            IssueID: 'PR-205783',
            Title: 'Missing documentation',
            Description: 'Add documentation for events',
            TypeIcon: 'assets/images/typeIssue48.svg',
            Type: 'Documentation',
            Assignee: 'Ed',
            ExpectedStoryPoints:4,
            ActualStoryPoints:3
        } );
    }
};
```

1. Create the sample comments data by creating a file called sampleCommentsData.js with following content:

```
export const generateSampleCommentsData = async db => {
    let result = await db.getAllFromStore( 'comments' );
    if( result.length === 0 ) {
        db.addToStore( 'comments', {
            Title: 'John Doe',
            Description: 'This is not replicable on latest environment.',
            TypeIcon: 'assets/images/typeComment48.svg'
        } );

        db.addToStore( 'comments', {
            Title: 'Mark Langella',
            Description: 'The issue seems to be as designed. It cannot be
fixed.',
            TypeIcon: 'assets/images/typeComment48.svg'
        } );
    }
};
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. On the launch of your custom page, add a onMount action to inisitalize the database as shown below:

```
import { generateSampleIssueData } from 'js/sampleIssueData';
import { generateSampleCommentsData } from 'js/sampleCommentsData';
import { getDB } from 'js/indexDb';

export const initialize = async dispatch => {
    let db = getDB();
    await generateSampleIssueData( db );
    await generateSampleCommentsData( db );
};
```

1. Create a folder src\images and add the required icon image files into it.

## Building pages using Common Location

A **Location** is a page level component which acts as a container grouping together one or more **Sublocations**, accessible via tabs.

SWF provides **Common Location** as a component which can be re-used to quickly build your application pages.

Here is an example to demonstrate the use of Common Location:

1. **Adding states configuration**

In your sample "trackerApp" module, open the **states.json** and add an entry for your sample location:

```
    "trackerAppRoot": {
        "abstract": true,
        "view": "commonRoot"
    },
    "trackerLocation": {
        "abstract": true,
        "data": {
            "browserSubTitle": {
                "source": "/i18n/TrackerAppMessages",
                "key": "trackerAppLocationTitle"
            },
            "headerTitle": {
                "source": "/i18n/TrackerAppMessages",
                "key": "trackerAppLocationTitle"
            }
        },
        "view": "commonLocation",
        "parent": "trackerAppRoot"
    },
```

```
    "trackerAppCommonLocationExample": {
        "url": "/trackerAppCR",
        "parent": "trackerLocation",
        "view": "trackerAppCommonLocationExample"
    }
```

Here, "trackerAppRoot" points to **Common Root** which is going to add the **Common Frame** to your application page.

Here, "trackerLocation" is a sample **Location** that you have defined. Defining it as `abstract` means that it cannot be instantiated as is and it needs at least one sublocation to be shown in it. Here, `view` specified as **commonLocation** brings in a default layout and behavior that this page level component offers. It has `parent` pointing to Common Root. The `data` section lets you define the **overridable** configurations like

- `browserSubTitle`
- `headerTitle`

Both of these can be configured by specifying the `key` and a corresponding entry in the appropriate localization file `source`.

Here, "trackerAppCommonLocationExample" is a sample **Sublocation**. A Sublocation is a page-level component which brings in the actual content of the page which should render when the `url` ("/trackerAppCR") specified in states.json is hit. Every sublocation is rendered as a tab inside its `parent` location("trackerLocation") component and it launches the `view` ("trackerAppCommonLocationExample") configured with it.

### 1. Adding localization

As explained above, to support the localization for "optional" `browserSubTitle` and `headerTitle`; add "TrackerAppMessages.json" in the `i18n` folder of your module. Make sure it has the entries corresponding to the `key` specified in the states.json.

In this example, the same `key` has been re-used for both `browserSubTitle` and `headerTitle`. You can choose to define them separately. This is how the entry in "TrackerAppMessages.json" would look like:

```
{
    "trackerAppLocationTitle": "Tracker App"
}
```

### 1. Adding sublocation views

As explained above, the location needs at least one sublocation to be defined and the `view` corresponding to that sublocation will show up when the `url` specified in sublocation definition in states.json is hit.

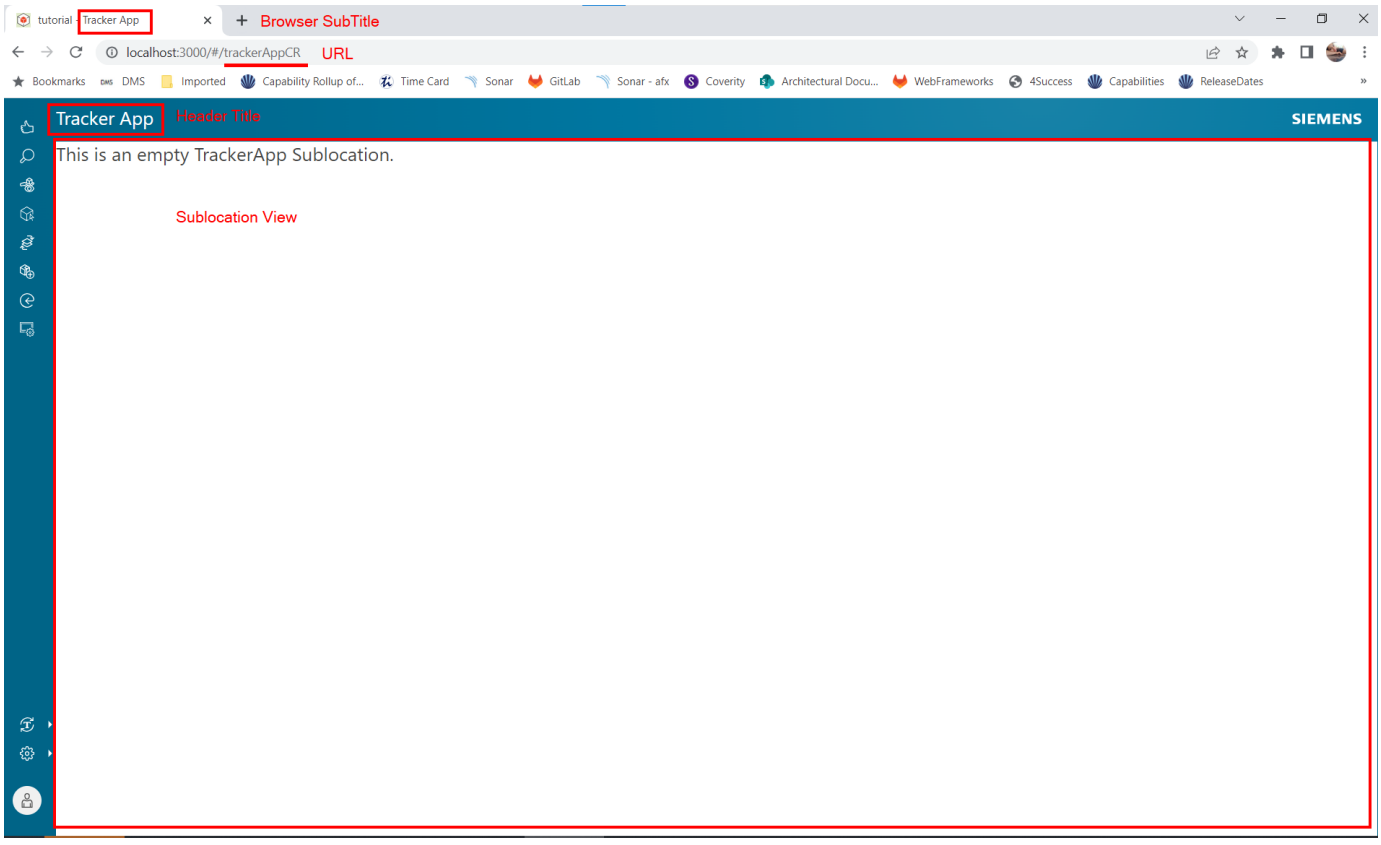

Siemens Web Framework
Unpublished work. © 2025 Siemens

Let us add a `view` and `viewModel`. In your module's, src\html folder, create a file "trackerAppCommonLocationExampleView.html" and in src\viewmodel folder create a file "trackerAppCommonLocationExampleViewModel.json" with following contents.

```
<h3>This is an empty TrackerApp Sublocation.</h3>
```

```
{
    "schemaVersion": "1.0.0"
}
```

When you build your application, refresh the browser and hit the `url` specified in your states.json; you should see:




The header component is bringing the Siemens logo by default. This can be **overridden** by using the **layoutSlots** configuration.

## What comes for free from Fx components (so far)

1. The layout for header, logo and sublocation
2. The routing support for opening the view when the specific URL is hit

### What you can override (so far)

1. The browser subtitle, header title
2. The content which should show up in sublocation view

## Building sub-pages using Common Location

A **Sublocation** is a page level component which brings in the actual content of the page which should render when the URL specified in the sublocation definition in states.json is hit. Every sublocation is rendered as a tab inside its parent **location** component and switching these tabs changes the sublocation view that is rendered.

SWF provides various templates of sublocations which can be re-used to quickly build your application pages.

- **AwDefaultSublocation** [Deprecated]
- **AwStandardSublocation**

Here is an example to demonstrate the use of Standard Sublocation:

1. **Adding states configuration**

In your sample "trackerApp" module, open the **states.json** and add an entry for your sample sublocations:

```
    "trackerAppRoot": {
        "abstract": true,
        "view": "commonRoot"
    },
    "trackerLocation": {
        "abstract": true,
        "data": {
            "browserSubTitle": {
                "source": "/i18n/TrackerAppMessages",
                "key": "trackerAppLocationTitle"
            },
            "headerTitle": {
                "source": "/i18n/TrackerAppMessages",
                "key": "trackerAppLocationTitle"
            }
        },
        "view": "commonLocation",
        "parent": "trackerAppRoot"
    },
    "issueTracker": {
        "url": "/issueTracker",
        "parent": "trackerLocation",
        "view": "issueTrackerPage",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "data": {
            "label": {
                "source": "/i18n/TrackerAppMessages",
                "key": "issueTrackerSubLocationTitle"
            }
        }
    },
    "commentsTracker": {
        "url": "/commentsTracker",
        "parent": "trackerLocation",
        "view": "commentsTrackerPage",
        "data": {
            "label": {
                "source": "/i18n/TrackerAppMessages",
                "key": "commentsTrackerSubLocationTitle"
            }
        }
    }
```

Here, "trackerAppRoot" and "trackerLocation" were already available in states.json as part of the **previous example** but they have been added here again only to show the full content of states.json.

Here, we have added two sublocations - "issueTracker" and "commentsTracker". Both of them will be rendered as tabs in the parent location. The `data` section defined with them provides the localization of the title of these tabs.

**1. Adding localization**

To support the localization for "mandatory" `label`; open "TrackerAppMessages.json" in the `i18n` folder of your module. Make sure it has the entries corresponding to the `key` specified in the states.json.

This is how the entry in "TrackerAppMessages.json" would look like:

```
{
    ...
    "issueTrackerSubLocationTitle": "Issue Tracker",
    "commentsTrackerSubLocationTitle": "Comments Tracker"
}
```

**1. Adding sublocation views**

Let us add the `view` and `viewModel`. In your module's, src\html folder, create a file "issueTrackerPageView.html" and in src\viewmodel folder create a file "issueTrackerPageViewModel.json" with following contents.

```
<h3>This is an empty Issue Tracker Page.</h3>
```

```
{
    "schemaVersion": "1.0.0"
}
```

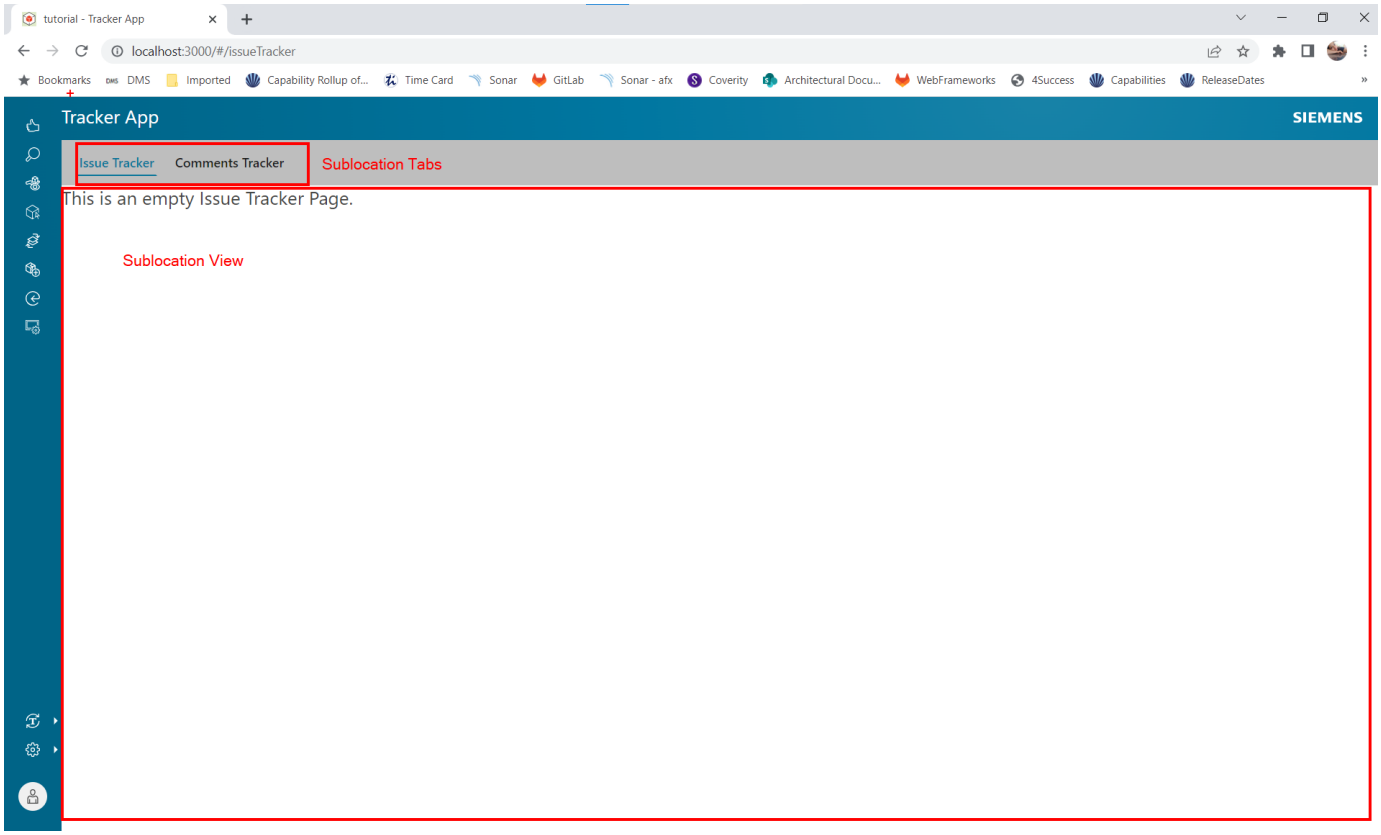
Similarly, add the corresponding files for "commentsTrackerPageView.html" and "commentsTrackerPageViewModel.json" with following contents.

```
<h3>This is an empty Comments Tracker Page.</h3>
```

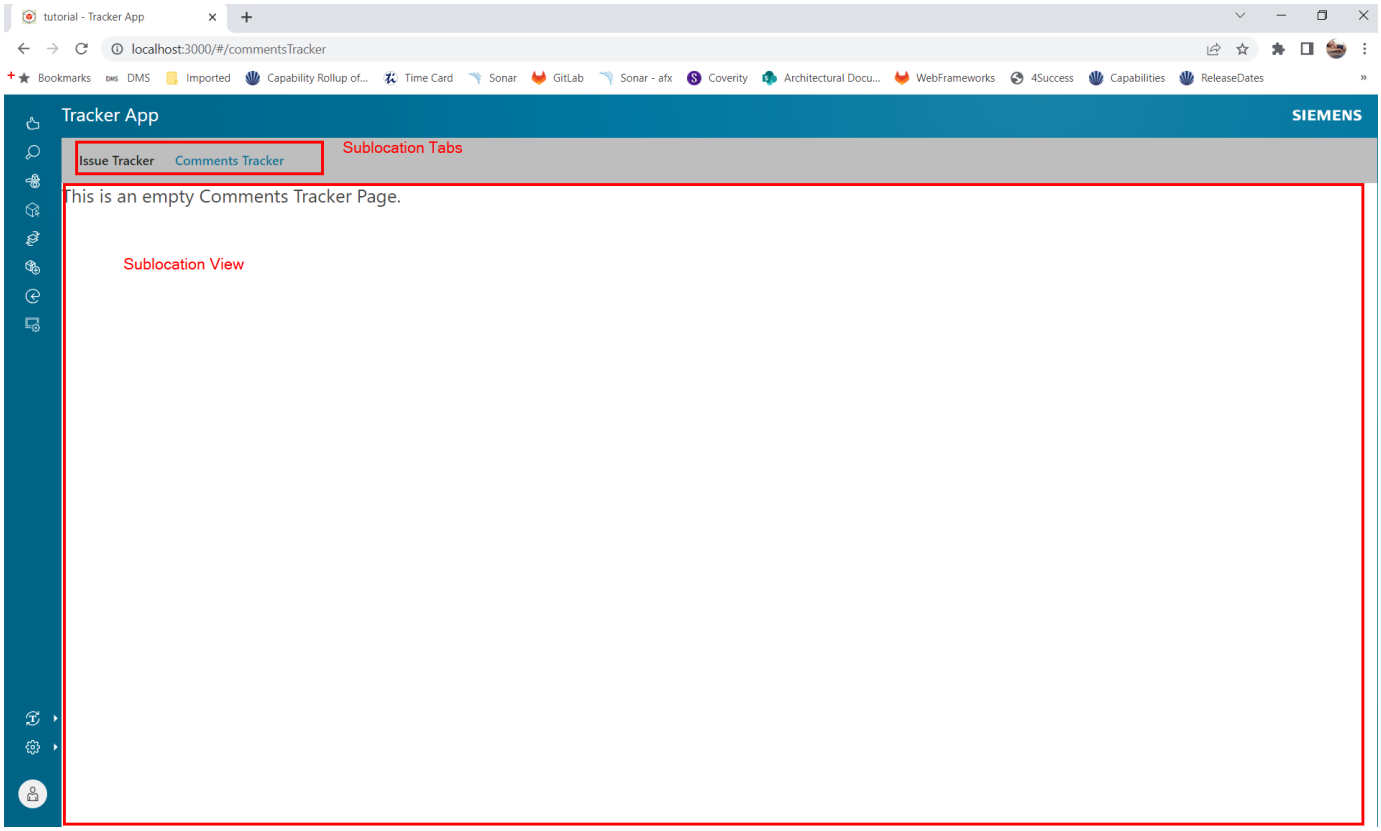
```
{
    "schemaVersion": "1.0.0"
}
```

When you build your application, refresh the browser and hit the `url` specified in your states.json; you should see:


Siemens Web Framework
Unpublished work. © 2025 Siemens




On switching the tab:




### What comes for free from Fx components (so far)

1. The layout for header, logo and sublocation
2. The routing support for opening the view when the specific URL is hit
3. The sub-pages rendered as tabs

### What you can override (so far)

1. The browser subtitle, header title
2. The content which should show up in sublocation view
3. The tabs that should be rendered as subpages
4. The content that should show up in each of these tab views
5. The order in which these tabs should appear

## Building lists using Standard sublocation

SWF provides **Standard sublocation** as a component which can be re-used to quickly build your application pages which have the need for showing the primary and secondary views. The interaction


Siemens Web Framework
Unpublished work. © 2025 Siemens

between the views shown in primary (master) and secondary (details) is also managed by this sublocation component.

The Primary Workarea can show the data in various **view modes** like list/table/tree/image. Each of these views need to be optionally configured by the application owner. The command to toggle between these views is added by default by the Standard sublocation component. However, what views to show in it is something that can be controlled by the application. By default, list, table and images views are shown.

Here is an example to demonstrate how to build a list by using Standard Sublocation:

**Tip** This example assumes that the **previous example** is already executed which means we have a tracker location with two sub-locations - issueTracker and commentsTracker.

In current example, we will build a list in the "issueTracker" sublocation.

**1. Adding list view mode configuration**

In your `issueTrackerPageViewModel.json` define a provider object containing the information about **viewBase and viewModes**. `viewModes` can be configured by specifying `pwaViewModes`, `secondaryWorkarea` and `supportedLayouts`.

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "provider": {
            "nameToken": "IssueTracker",
            "viewBase": "issue",
            "pwaViewModes": [ "list" ],
            "secondaryWorkarea": true
        }
    }
}
```

Here, we have configured two view modes - `SummaryView` which shows a list and a summary (secondaryWorkArea) and `ListView` which shows only the list view. We have also specified "issue" as the `viewBase` which means we are instructing the framework to look for a component (view and viewmodel) with name `viewBase` + applicable display mode (like "list" in this case) and render it accordingly.

In the `issueTrackerPageView.html` consume the Fx component - **AwStandardSublocation** and pass the mandatory prop `provider` with the object created in the corresponding json file.

```
<aw-standard-sublocation provider="data.provider"></aw-standard-sublocation>
```

**1. Adding component containing list**

In your src\html folder, add a file `issueListView.html` with following content:

```
<aw-panel>
    <aw-list dataprovider="data.dataProviders.getIssues">
        <aw-default-cell vmo="item"></aw-default-cell>
    </aw-list>
</aw-panel>
```

**Tip** The name of the file is viewBase specified in `provider` and the applicable display mode specified in the `provider` object sent to AwStandardSublocation component.

Here, we are using the **AwList** component and passing it the **dataProvider** created in the corresponding viewmodel.

In your src\viewmodel folder, add a file `issueListViewModel.json` with following content:

```
{
    "schemaVersion": "1.0.0",
    "dataProviders": {
        "getIssues": {
            "action": "fetchIssueData",
            "response": "{{data.issueResults}}",
            "totalFound": "{{data.totalFound}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            }
        }
    },
    "actions": {
        "fetchData": {
            "actionType": "dataProvider",
            "method": "getIssues"
        },
        "fetchIssueData": {
            "actionType": "JSFunctionAsync",
            "method": "fetchIssueData",
            "inputData":{
                "store": "issue"
            },
            "dataParsers": [{
                "id": "issueDataParser",
                "responseObjs": "data.result"
            }],
            "outputData": {
                "totalFound": "data.totalResults",
                "issueResults": "data.result"
            },
            "deps": "js/issueHelperService"
        }
    },
    "dataParseDefinitions": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "issueDataParser": {
            "identifier": "{{response.uid}}",
            "cellHeader1": "{{response.Title}}",
            "typeIconURL": "{{response.TypeIcon}}",
            "cellProperties": {
                "Title": {
                    "key": "Title",
                    "displayName": "Title",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Description": {
                    "key": "Description",
                    "displayName": "Description",
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                }
            },
            "props": {
                "Title":{
                    "displayName": "{{response.Title}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Description":{
                    "displayName": "{{response.Description}}",
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                }
            }
        }
    },
    "lifecycleHooks":{
        "onMount": "fetchData"
    }
}
```

Here, **lifecycleHooks** have been used to execute a dataProvider action when the component is mounted. **DataParseDefinition** have been used to create the **ViewModelObjects** corresponding to the data received in the response.

Here, **selectionModel** received/managed by the SubLocation component has been re-used and sent to the dataProvider of the list. This is an important step which will be required even in our later tutorials to have sync between primary workarea list and details view that we will be adding in secondary workarea.

**NOTE** This example, uses a dummy server wherein the pre-created data is stored in browser's indexedDB and is fetched from there. Naturally, this implementation for your sample application will change with the corresponding data source and business logic that you would need.

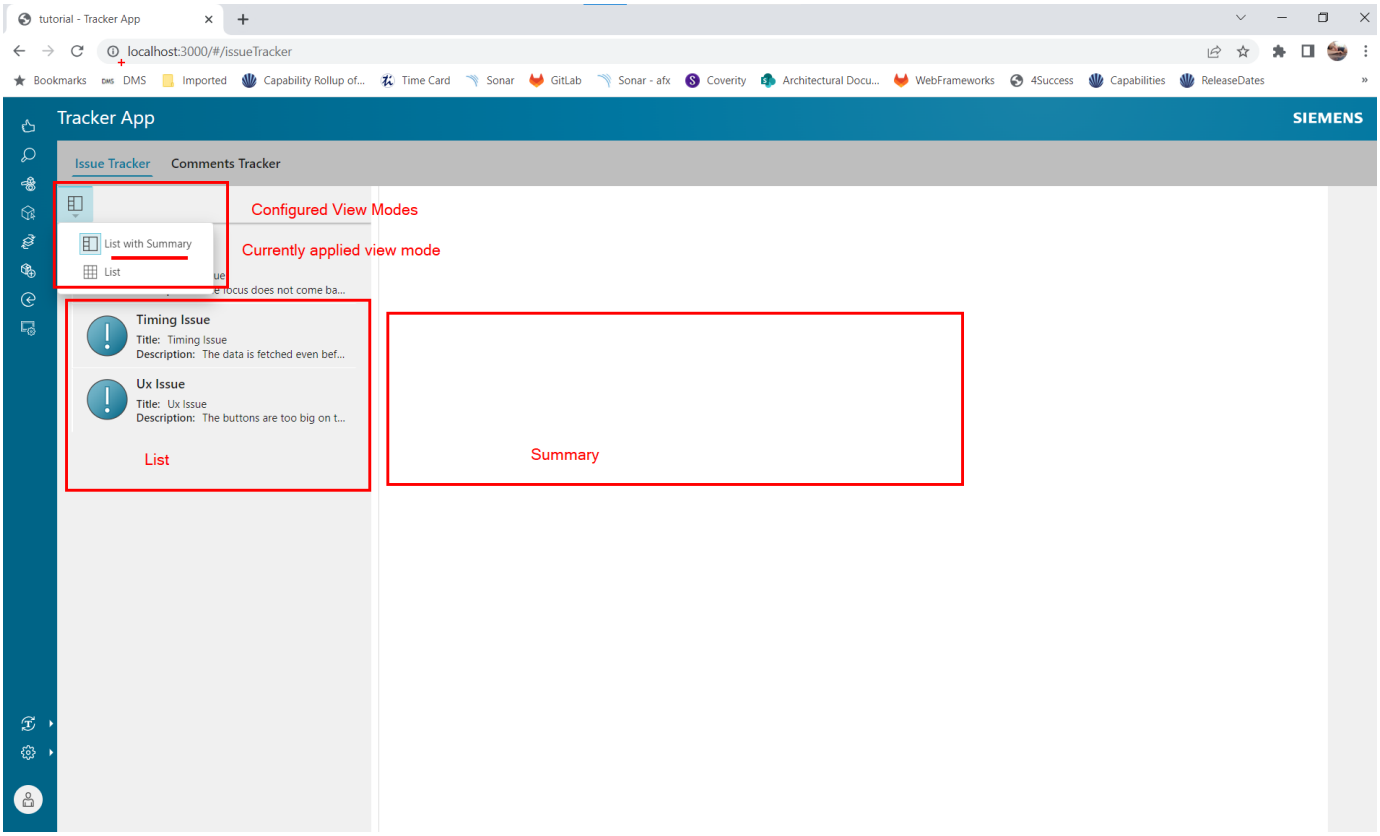
This example depends on a glue code which is defined in "issueHelperService", so for this specific example, in src\js folder, create a JS file with this name and add the following content in it:

```
import { getDB } from 'js/indexDb';

export const fetchIssueData = async() => {
    let result = await getDB().getAllFromStore( 'issue' );
    return Promise.resolve( { data: { result: result } } );
};
```

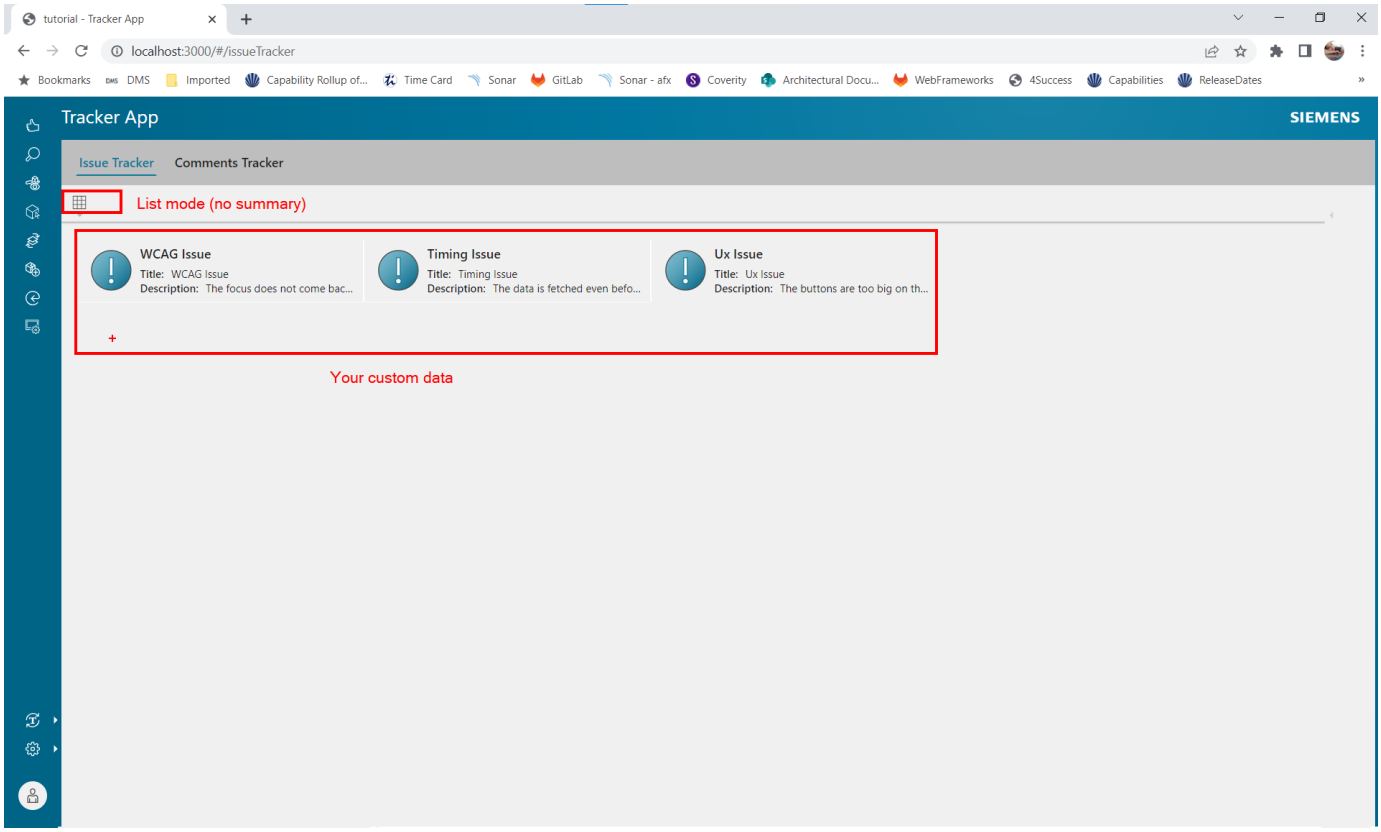
**Tip** Please note that the imported file `indexDb` and everything which is required to mock the server response is available in the **MockServerForTrackingApp**

When you build your application, refresh the browser and hit the `url` specified in your states.json; you should see:




On switching the view mode:


Siemens Web Framework
Unpublished work. © 2025 Siemens




### What comes for free from Fx components (so far)

1. The commands for switching the display modes
2. The layout for Primary view, moveable Splitter and Secondary view (summary)

### What you can override (so far)

1. The view modes that you want to configure
2. The default view mode (if not specified, the first view mode is picked up)
3. The content to render in these views

## Building tables using Standard sublocation

SWF provides **Standard sublocation** as a component which can be re-used to quickly build your application pages which have the need for showing the primary and secondary views. The interaction between the views shown in primary (master) and secondary (details) is also managed by this sublocation component.

The Primary Workarea can show the data in various **view modes** like list/table/tree/image. Each of these views need to be optionally configured by the application owner. The command to toggle between these

views is added by default by the Standard sublocation component. However, what views to show in it is something that can be controlled by the application. By default, list, table and images views are shown.

Here is an example to demonstrate how to build a table by using Standard Sublocation:

**Tip** This example assumes that the **previous example** is already executed which means we have a tracker location with two sub-locations - issueTracker and commentsTracker. We have a list added in issueTracker sublocation.

In current example, we will build a table in the "issueTracker" sublocation.

### 1. Adding table view mode configuration

In your `issueTrackerPageViewModel.json` add the viewModes for tables views in the provider object containing the information about **viewBase and viewModes**

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "provider": {
            "nameToken": "IssueTracker",
            "viewBase": "issue",
            "pwaViewModes":  [ "list", "table" ],
            "secondaryWorkarea": true
        }
    }
}
```

Here, we have configured two additional view modes - `TableSummaryView` which shows a table and a summary (secondaryWorkArea) and `TableView` which shows only the table view. We have also specified "issue" as the `viewBase` which means we are instructing the framework to look for a component (view and viewmodel) with name `viewBase` + applicable display mode (like "table" in this case) and render it accordingly.

In the `issueTrackerPageView.html` the Fx component - **AwStandardSublocation** is already consumed and the mandatory prop `provider` with the object created in the corresponding json file is already passed.

```
<aw-standard-sublocation provider="data.provider"></aw-standard-sublocation>
```

### 1. Adding component containing table

In your src\html folder, add a file `issueTableView.html` with following content:

```
<aw-splm-table gridid="issueTable"></aw-splm-table>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Tip** The name of the file is viewBase specified in `provider` and the applicable display mode specified in the `provider` object sent to AwStandardSublocation component.

Here, we are using the **AwTable** component and passing it the gridID created in the corresponding viewmodel.

In your src\viewmodel folder, add a file `issueTableViewModel.json` with following content:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "fetchIssueData": {
            "actionType": "JSFunctionAsync",
            "method": "fetchIssueData",
            "inputData": {
                "store": "issue"
            },
            "dataParsers": [{
                "id": "issueDataParser",
                "responseObjs": "data.result"
            }],
            "outputData": {
                "totalFound": "data.totalResults",
                "issueResults": "data.result"
            },
            "deps": "js/issueHelperService"
        }
    },
    "dataProviders": {
        "issueDataProvider": {
            "action": "fetchIssueData",
            "response": "{{data.issueResults}}",
            "totalFound": "{{data.totalFound}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            }
        }
    },
    "columnProviders": {
        "issueColumnProvider": {
            "frozenColumnIndex": 1,
            "columns": [{
                    "name": "icon",
                    "displayName": "",
                    "maxWidth": 30,
                    "minWidth": 30,
                    "width": 30,
                    "enableColumnMenu": false,
                    "pinnedLeft": true
                },
                {
                    "name": "Title",
                    "displayName": "Title",
```

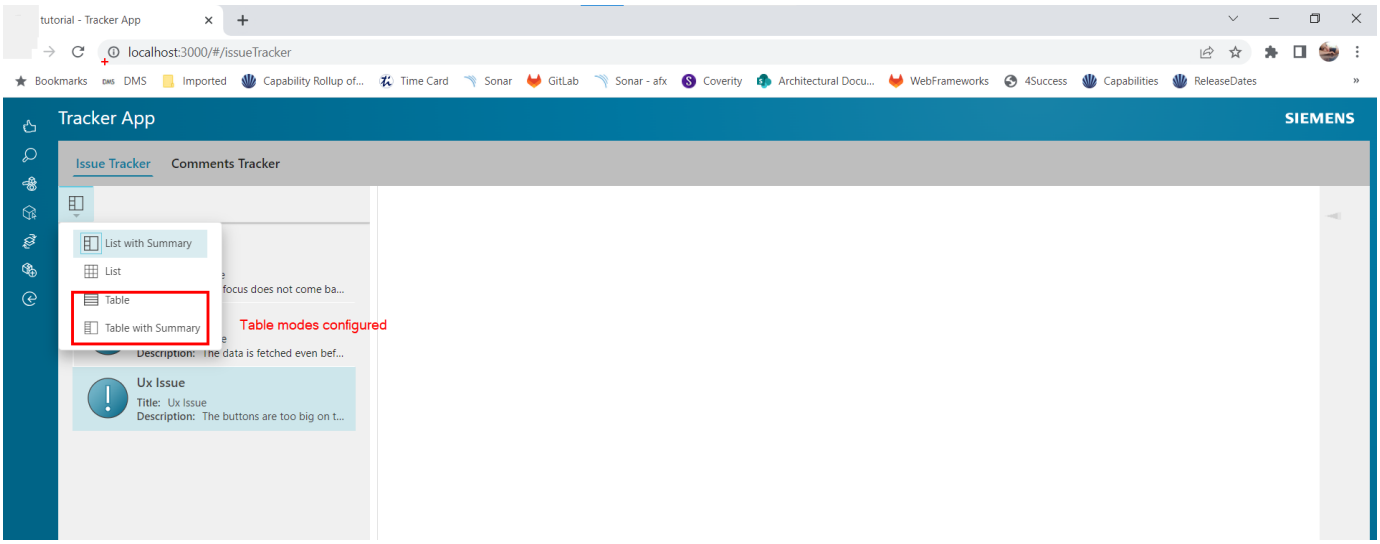
```
                    "minWidth": 100,
                    "width": 150,
                    "enableColumnMenu": false,
                    "pinnedLeft": true
                },
                {
                    "name": "Description",
                    "displayName": "Description",
                    "minWidth": 250,
                    "width": 400,
                    "enableColumnMenu": false,
                    "pinnedLeft": true
                }
            ]
        }
    },
    "functions": {},
    "grids": {
        "issueTable": {
            "dataProvider": "issueDataProvider",
            "columnProvider": "issueColumnProvider",
            "addIconColumn": true,
            "gridOptions": {
                "enablePinning": true,
                "enableSorting": false
            }
        }
    },
    "dataParseDefinitions": {
        "issueDataParser": {
            "identifier": "{{response.uid}}",
            "typeIconURL": "{{response.TypeIcon}}",
            "props": {
                "Title":{
                    "displayName": "{{response.Title}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Description":{
                    "displayName": "{{response.Description}}",
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                }
            }
        }
    }
}
```

Here, **DataParseDefinition** have been used to create the **ViewModelObjects** corresponding to the data received in the response.

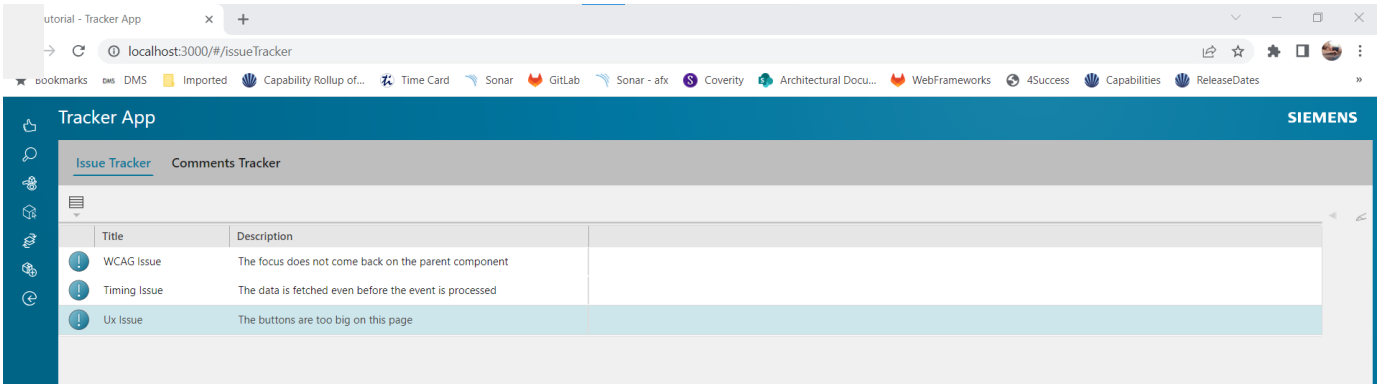

Siemens Web Framework
Unpublished work. © 2025 Siemens

Here, **selectionModel** received/managed by the SubLocation component has been re-used and sent to the dataProvider of the list. This is an important step which will be required even in our later tutorials to have sync between primary workarea list and details view that we will be adding in secondary workarea.

When you build your application, refresh the browser and hit the `url` specified in your states.json; you should see:




On switching the view mode to table:




Select something in table and switch to list mode; you will see that the selection is retained/shared between the two views. This is something that you get in-built from from Fx.

## What comes for free from Fx components (so far)

1. The commands for switching the display modes
2. The layout for Primary view, moveable Splitter and Secondary view (summary)
3. The sharing of selection model i.e. when you select something in one view (say list) and switch to another (say table), the selection is not lost.

### What you can override (so far)

1. The view modes that you want to configure
2. The default view mode (if not specified, the first view mode is picked up)
3. The content to render in these views

## Building reusable pages using Standard sublocation

Here is an example to demonstrate how to build reusable pages by using Standard Sublocation:

**Tip** This example assumes that the **previous example** is already executed which means we have a tracker location with two sub-locations - issueTracker and commentsTracker. We have a list and table added in issueTracker sublocation. We have empty views added in commentsTracker sublocation.

In current example, we will reuse the views created for "issueTracker" sublocation to populate the "commentsTracker" sublocation.

**1. Consuming AwStandardSublocation**

In your `commentsTrackerPageView.html` consume **AwStandardSublocation**

```
<aw-standard-sublocation provider="data.provider"></aw-standard-sublocation>
```

In your `commentsTrackerPageViewModel.json` add the provider object containing the infomration about **viewBase and viewModes**

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "provider": {
            "nameToken": "CommentsTracker",
            "viewBase": "issue",
            "pwaViewModes":  [ "list", "table" ],
            "secondaryWorkarea": true,
            "dataSource": "comments"
        }
    }
}
```

Here, we are **reusing** the same `viewBase` which was defined for earlier sublocation. This example shows how generic views can be re-used across various sublocations and only the variable information can be parameterized.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Here, we have additionally defined a custom property called `dataSource` to parameterize our list and table components so that the ones added for issueTracker can be re-used even for showing the information for comments.

**1. Adding parameter in list/table components to achieve re-use**

In your `issueListViewModel.json` modify the existing action to fetch the data to remove the hardcoding with parameterized data.

For this specific example, this is the **before**

```
...
"actions": {
    ...
    "fetchIssueData": {
        "actionType": "JSFunctionAsync",
        "method": "fetchIssueData",
        "inputData":{
            "store": "issue"
        },
        "dataParsers": [{
            "id": "issueDataParser",
            "responseObjs": "data.result"
        }],
        "outputData": {
            "totalFound": "data.totalResults",
            "issueResults": "data.result"
        },
        "deps": "js/issueHelperService"
    }
}
...
```

and this is the **after** with paraparameterized "dataSource":

```
...
"actions": {
    ...
    "fetchIssueData": {
        "actionType": "JSFunctionAsync",
        "method": "fetchIssueData",
        "inputData":{
            "store": "{{props.subPanelContext.provider.dataSource}}"
        },
        "dataParsers": [{
            "id": "issueDataParser",
            "responseObjs": "data.result"
        }],
        "outputData": {
            "totalFound": "data.totalResults",
            "issueResults": "data.result"
```

```
            },
            "deps": "js/issueHelperService"
        }
    }
    ...
```

Here, the name of the store has been changed from a hardcoded name to a parameterized name which is getting resolved via the custom dataSource property that we added in the provider object defined at the page level component.

Exact same change will be needed in the `issueTableViewModel.json` to have the custom table component also function as a reusable component.

**Tip** Do not forget to send `dataSource` information from `issueTrackerPageViewModel.json` as well:

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "provider": {
            "nameToken": "IssueTracker",
            "viewBase": "issue",
            "pwaViewModes": [ "list", "table" ],
            "secondaryWorkarea": true,
            "dataSource": "issue"
        }
    }
}
```

Similarly, the glue code in `issueHelperService.js` needs to be modified to accept a parameter instead of hardcoded data source name:

**before**

```
import { getDB } from 'js/indexDb';

export const fetchIssueData = async() => {
    let result = await getDB().getAllFromStore( 'issue' );
    return Promise.resolve( { data: { result: result } } );
};
```

**after**

```
import { getDB } from 'js/indexDb';

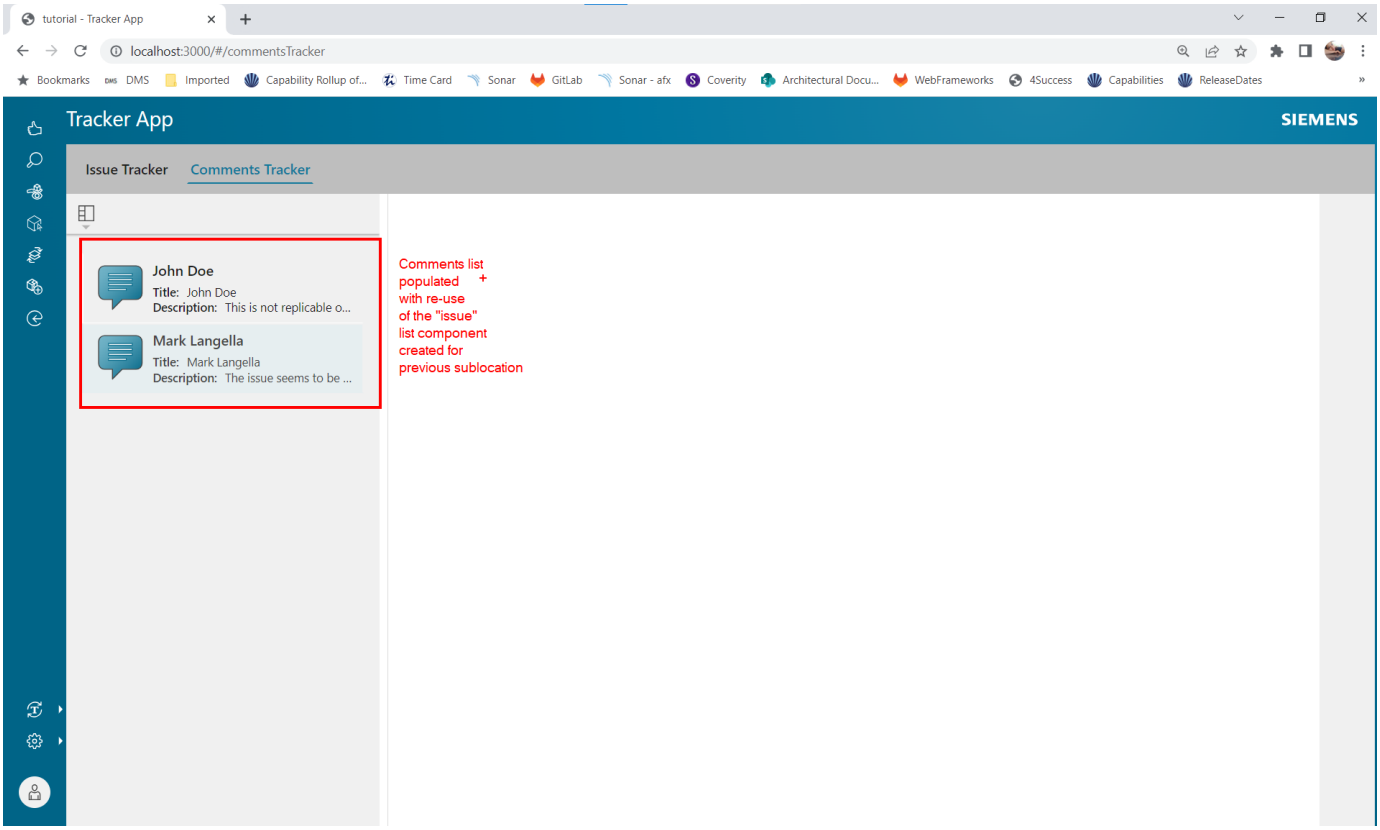
export const fetchIssueData = async( store ) => {
    let result = await getDB().getAllFromStore( store );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

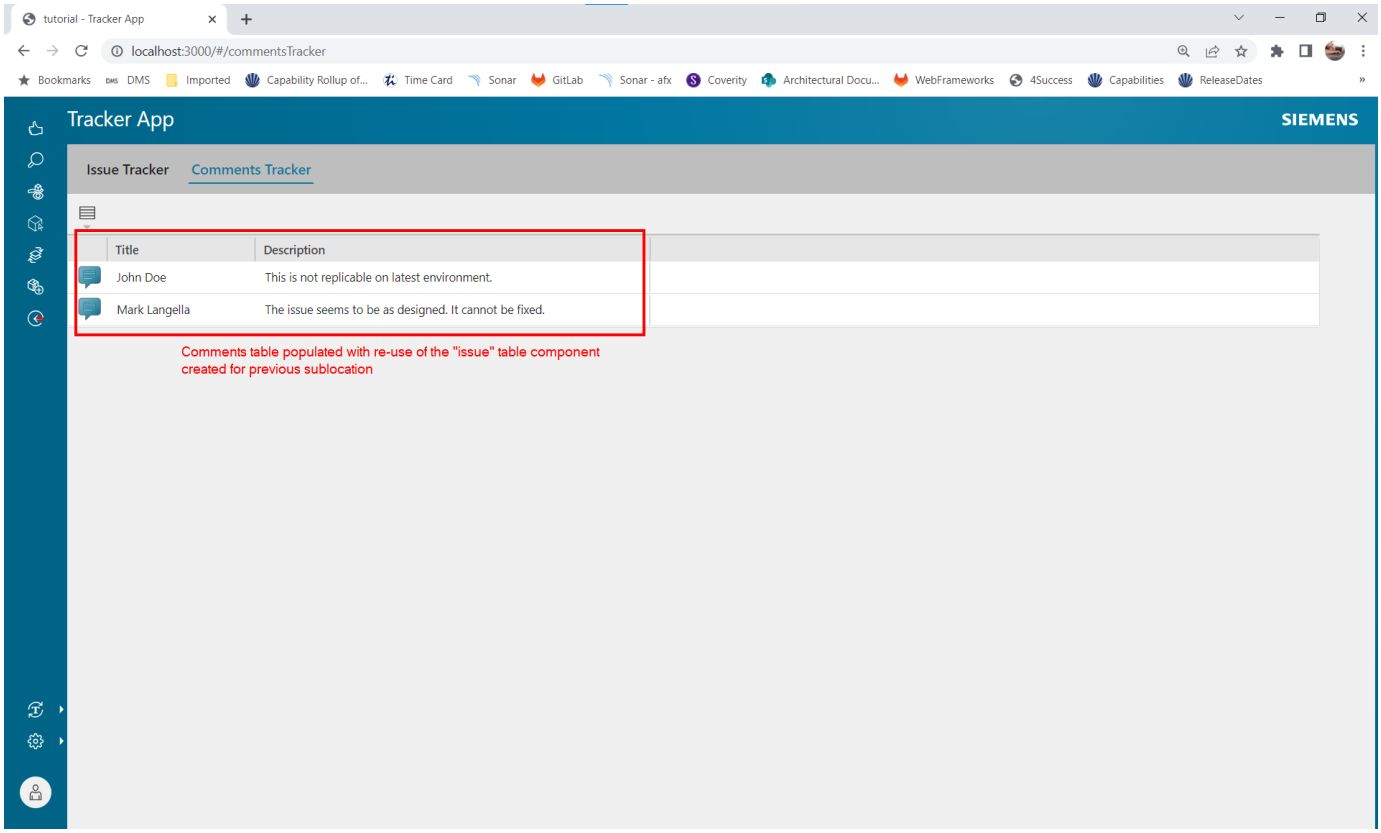
```
    return Promise.resolve( { data: { result: result } } );
};
```

**Tip** Please note that the imported file `indexDb` and everything which is required to mock the server response is available in the **MockServerForTrackingApp**

When you build your application, refresh the browser and hit the `url` specified in your states.json; you should see:




On switching the view mode to table:




### What comes for free from Fx components (so far)

1. The commands for switching the display modes
2. The layout for Primary view, moveable Splitter and Secondary view (summary)
3. The sharing of selection model i.e. when you select something in one view (say list) and switch to another (say table), the selection is not lost.
4. The re-usability of viewModes across various sub-pages

### What you can override (so far)

1. The view modes that you want to configure
2. The default view mode (if not specified, the first view mode is picked up)
3. The content to render in these views

## Building details pages using Standard sublocation

Here is an example to demonstrate how to build details pages by using Standard Sublocation:


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Tip** This example assumes that the **previous example** is already executed which means we have a tracker location with two sub-locations - issueTracker and commentsTracker. We have a list and table added in both the sublocations.

In current example, we will add a details view which reflects to the selection change being done in the primary views configured in "issueTracker" sublocation.

### 1. Adding SecondaryWorkarea Tabs

In your module, add a `secondaryWorkareaTabs.json` parallel to your `module.json` file. This will help us in configuring the **SecondaryWorkarea** which acts as a Details view for the master being selected in the **PrimaryWorkarea**

```
{
    "schema": "1.0.0.0",
    "tabs": {
        "issueDetailsTabTitle": {
            "name": {
                "source": "/i18n/TrackerAppMessages",
                "key": "issueDetailsTabTitle"
            },
            "priority": 10,
            "view": "issueDetails",
            "visibleWhen": "context.nameToken === 'IssueTracker' &&
selection.length === 1"
        },
        "emptyWorkarea": {
            "name": "",
            "priority": 20,
            "view": "emptyWorkarea",
            "visibleWhen": "context.nameToken === 'IssueTracker' &&
selection.length === 0"
        }
    }
}
```

Here, we are creating two tabs:

- "issueDetailsTabTitle" - this has a title localized and defined in the corresponding messages file referred via `name`. This has the `priority` to identify the order in which multiple tabs if configured should show. This has the `view` which represents the name of the component which should render when this tab is selected. This has `visibleWhen` which basically defines the **declarative condition** under which this tab should show up.

In this example, when the `nameToken` matches with the specified value i.e. we are on IssueTracker page and the length of the `selection` coming from PrimaryWorkarea is 1; then only this tab should be rendered.

- "emptyWorkarea" - this is the tab which will get rendered when there is no selection. This is wherein you can define a view of your choice to guide the user to next action when there is no selection in PrimaryWorkarea and hence nothing to show in the SecondaryWorkarea.

**1. Adding the components which should render for the added tabs**

Create a component with name `issueDetailsView.html` and `issueDetailsViewMode.json`.

For this specific example, we are choosing the view which simply prints some of the properties of the object selected in PrimaryWorkarea.

```
<aw-flex-column width="12">
    <aw-flex-row height="fill">
        <aw-flex-column width="12">
            <aw-widget prop="data.selectedObj.props.IssueID"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Title"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Type"></aw-widget>
            <aw-break></aw-break>
            <aw-widget prop="data.selectedObj.props.Description"></aw-
widget>
        </aw-flex-column>
    </aw-flex-row>
</aw-flex-column>
```

The code above uses **AwWidget** to render the properties available on the input object.

Here, is the content of the ViewModel:

```
{
    "actions": {
        "fetchDetails": {
            "actionType": "JSFunctionAsync",
            "method": "fetchDetails",
            "inputData": {
                "store": "issue",
                "issueId": "{{props.subPanelContext.selection[0].uid}}"
            },
            "dataParsers": [
                {
                    "id": "issueDetailsParser",
                    "responseObjs": "data.result"
                }
            ],
            "outputData": {
                "selectedObj": "data.result"
            },
            "deps": "js/issueHelperService"
        }
    },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "dataParseDefinitions": {
        "issueDetailsParser": {
            "identifier": "{{response.uid}}",
            "props": {
                "IssueID": {
                    "name": "IssueID",
                    "displayName": "Issue ID",
                    "value": "{{response.IssueID}}",
                    "displayValue": "{{response.IssueID}}",
                    "type": "String"
                },
                "Title": {
                    "name": "Title",
                    "displayName": "Title",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Description": {
                    "name": "Description",
                    "displayName": "Description",
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                },
                "Type": {
                    "name": "Type",
                    "displayName": "Type",
                    "value": "{{response.Type}}",
                    "displayValue": "{{response.Type}}",
                    "type": "String"
                }
            }
        }
    },
    "lifecycleHooks": {
        "onUpdate": {
            "observers": [ "props.subPanelContext.selection" ],
            "action": "fetchDetails"
        }
    }
}
```

Here, `onUpdate` **lifecycleHook** has been used to observe any change in the `props.subPanelContext.selection`. This is nothing but the PrimaryWorkarea selection. Basically, we are saying, whenever there is a change in this selection, invoke the action `fetchDetails` which uses some glue code to fetch the information related to the selected issueId and populates it on `data.selectedObj` which is what is referred in the corresponding view.html.

The unique ID of the input selection is available on the path `props.subPanelContext.selection[0].uid` which is what has been used in the custom action

to fetch the data for this unique object. Here is how the glue code (`issueHelperService.js`) for this example code looks like:

```
import { getDB } from 'js/indexDb';

export const fetchDetails = async( store, issueId ) => {
    let result = await getDB().getFromStore( store, issueId );
    return Promise.resolve( { data: { result: result } } );
};
```

**Tip** Please note that the imported file `indexDb` and everything which is required to mock the server response is available in the **MockServerForTrackingApp**

Similarly, the view configured for second tab `emptyWorkarea` can be added. For our example, we are using an empty state view: `emptyWorkareaView.html`:

```
<aw-empty-workarea image='graEmptySearchSummary'></aw-empty-workarea>
```

`emptyWorkareaViewModel.json`:

```
{
    "schemaVersion": "1.0.0"
}
```

But overall, any other view with desired contents can be configured here.

**Tip** If there is a single tab visible in Secondary Workarea, the tab title is hidden by default. The title shows up when there is more than one visible tab configured.

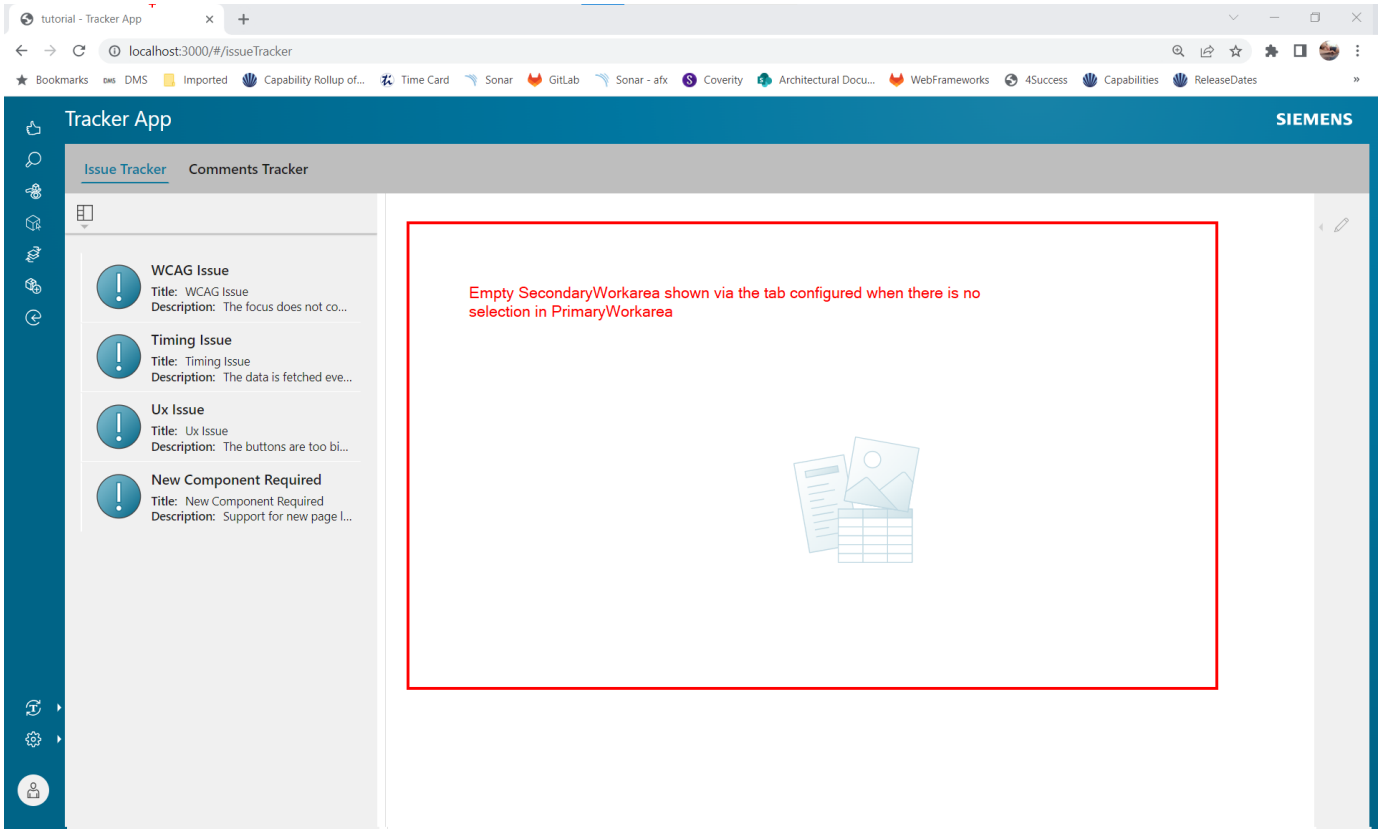
**1. Adding the localization**

In your src\i18n folder, in the existing `TrackerAppMessages.json`, make an entry for the tab title:

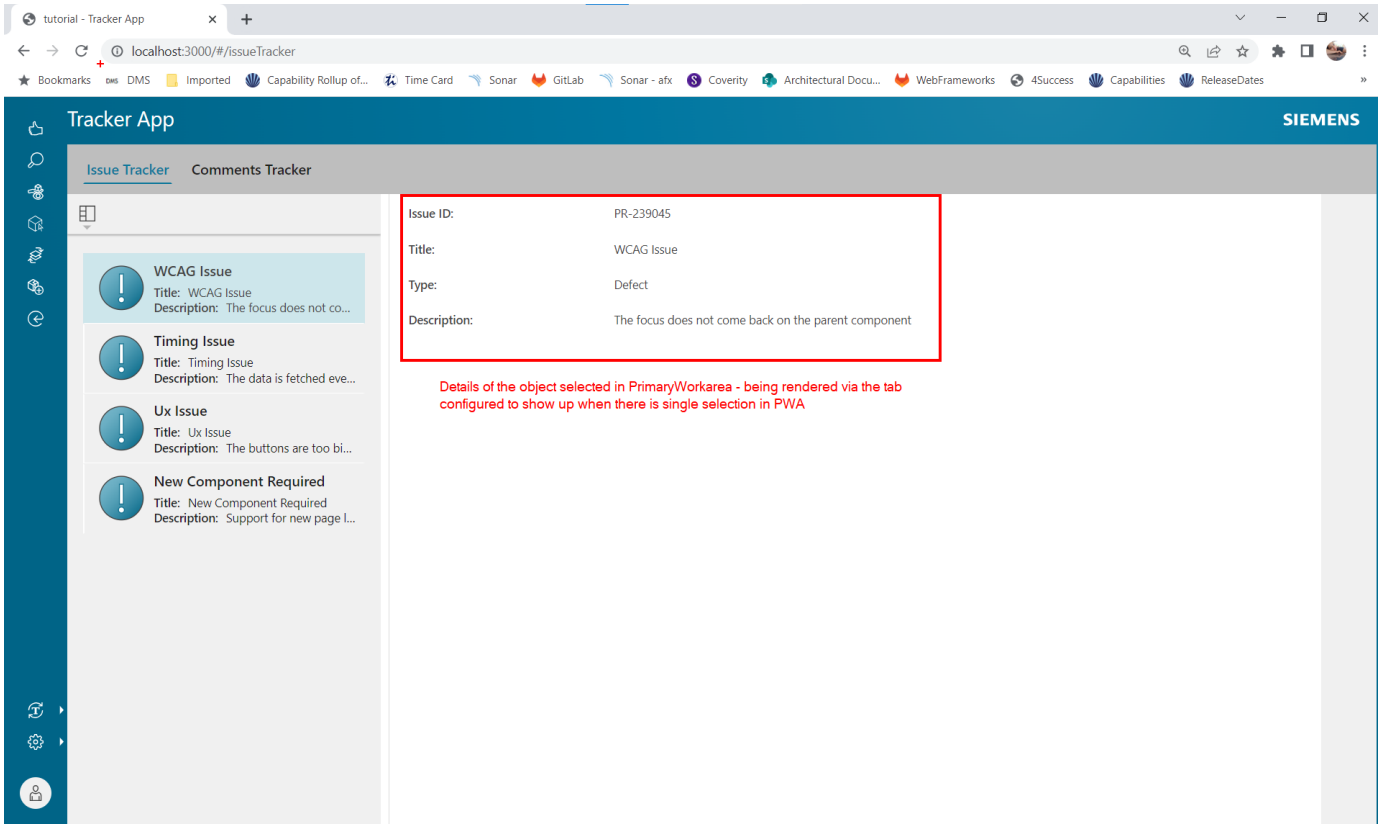
```
{
    ...
    "issueDetailsTabTitle": "Issue Details"
}
```

When you build your application, refresh the browser and hit the `url` specified in your states.json; you should see:


Siemens Web Framework
Unpublished work. © 2025 Siemens

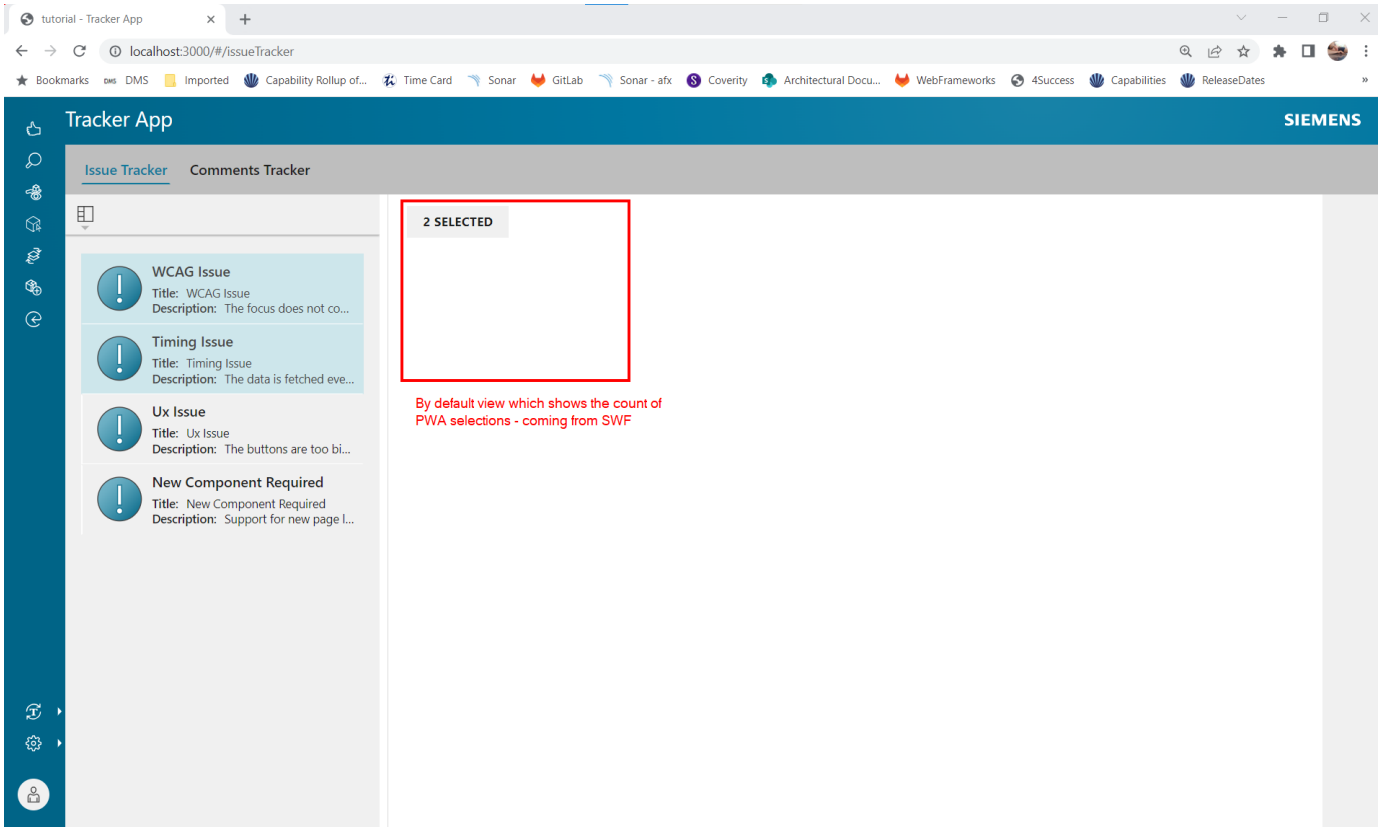



On selecting something in the PrimaryWorkarea, the details of the selected object would show up:




On doing multi-selection in the PrimaryWorkarea, the default view configured in SWF to show the selection count will show up:


Siemens Web Framework
Unpublished work. © 2025 Siemens




## What comes for free from Fx components (so far)

1. The commands for switching the display modes
2. The layout for Primary view, moveable Splitter and Secondary view (summary)
3. The sharing of selection model i.e. when you select something in one view (say list) and switch to another (say table), the selection is not lost.
4. The re-usability of viewModes across various sub-pages
5. The selection synchronization between PWA and SWA
6. The default view which shows up on multi-selection in PWA

## What you can override (so far)

1. The view modes that you want to configure
2. The default view mode (if not specified, the first view mode is picked up)
3. The content to render in these views
4. The tabs that you want to render on no/single/multi selections in PWA
5. The order in which these tabs should show
6. The views which should render when these tabs are selected

## Building an alternate primary workarea pages using standard sublocation

Here is an example to demonstrate how to build alternate primary workarea pages by using Standard Sublocation:

**Tip** This example assumes that the **previous example** is already executed which means we have a tracker location with two sub-locations - issueTracker and commentsTracker. We have a list, table added in both the sublocations and details page is configured.

In current example, we will add a alternate primary workarea view which reflects the data in primary workarea in different format for "issueTracker" sublocation.

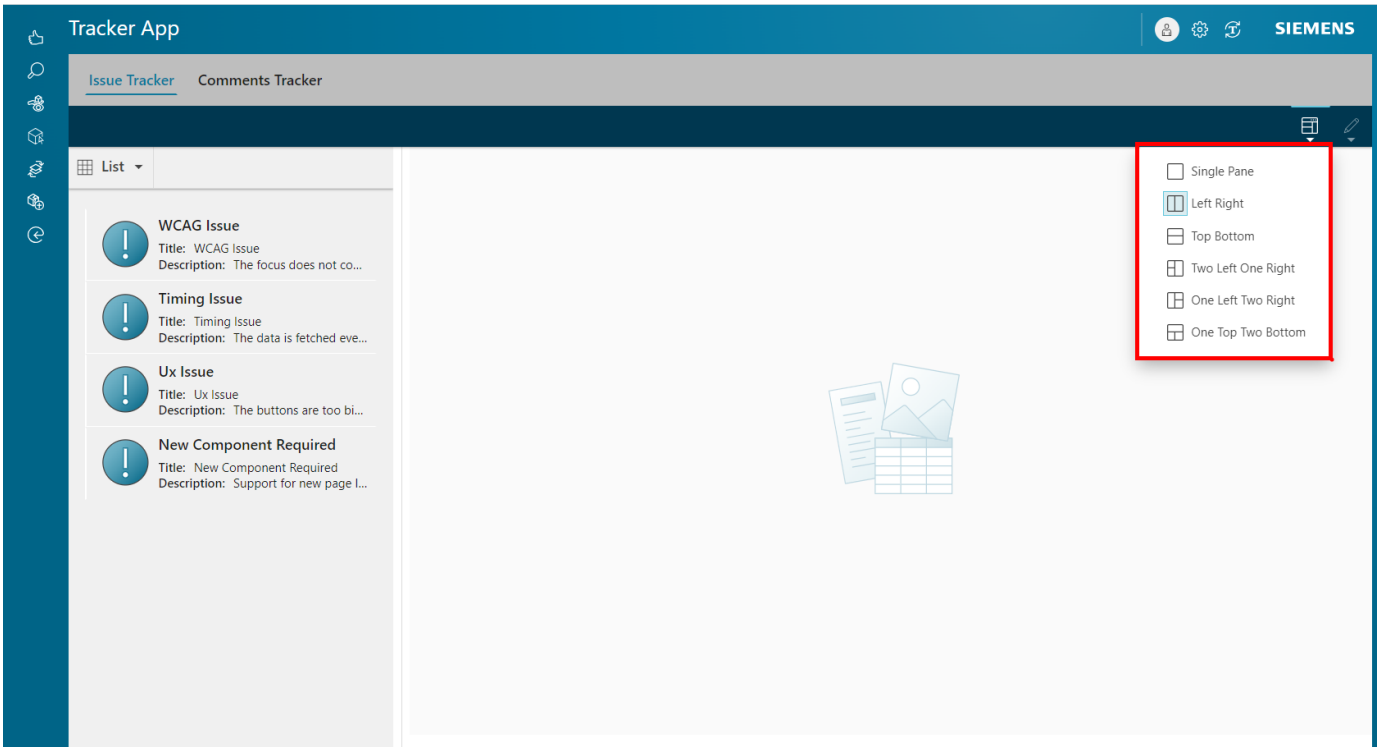
**1. Enable three view layouts**

To enable three view layouts, set supportedLayouts attribute in the provider of viewModel `issueTrackerPageViewModel.json`.

```
{
    "data":{
        "provider":{
            ...,
            "supportedLayouts": [ "Awp0SinglePaneLayout",
"Awp0TopBottomLayout","Awp0LeftRightLayout", "Awp0TwoLeftOneRightLayout",
"Awp0OneLeftTwoRightLayout", "Awp0OneTopTwoBottomLayout" ]
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

When you build your application, you should see:




### 1. Create alternate primary workarea view-viewModel

Create view-viewModel for alternate primary workarea e.g. `chartPageView.html` and `chartPageViewModel.json`. One toolbar needs to be added with anchors as mentioned below.

```
<div class='sw-column w-12'>
    <aw-toolbar class="aw-layout-workareaCommandbar aw-theme-toolbar"
first-anchor="aw_altPWADisplay" orientation="HORIZONTAL" overflow="true"
context="props.subPanelContext"></aw-toolbar>
    <aw-panel>
        <aw-panel-body>
            <aw-chart chart-
provider="data.chartProviders.lineChartProvider"></aw-chart>
        </aw-panel-body>
    </aw-panel>
</div>
```

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "getChartDataActionForMultipleSeries": {
            "actionType": "JSFunction",
            "method": "createChartFromArrayOfSeries",
            "inputData": {
                "input": ""
            },
```

```
            "outputData": {
                "chartProviders.lineChartProvider.chartPoints": ""
            },
            "deps": "js/trackerAppService"
        },
        "setSelectPoint": {
            "actionType": "JSFunction",
            "method": "setSelectPoint",
            "deps": "js/trackerAppService",
            "inputData": {
                "eventData":"{{parameters.eventData}}"
            }
        }
    },
    "chartProviders": {
        "lineChartProvider": {
            "title": "",
            "chartType": "line",
            "loadDataAction": "getChartDataActionForMultipleSeries",
            "chartConfig": {
                "isChartZoomable": true,
                "isYAxisLinearOrLogarithmic": "linear",
                "xAxisLabel": "Story Points Expected",
                "yAxisLabel": "Story Points Spent",
                "tooltip": {
                    "useHTML": true,
                    "headerFormat": "",
                    "pointFormat": "Story Points Expected: {point.x}<br>
Story Points Spent : {point.y}<br>"
                }
            }
        }
    },
    "onEvent": [
        {
            "eventId": "lineChartProvider.selected",
            "action": "setSelectPoint",
            "inputArgs": {
                "eventData": "{{eventData}}"
            }
        }
    ]
}
```

```
Note :

First anchor for the toolbar should be set to "aw_altPWADisplay" in view
file. first-anchor="aw_altPWADisplay"
```

**1. Contribute command for alternatePWA**


Siemens Web Framework
Unpublished work. © 2025 Siemens

Contribute command in `commandsViewModel.json` of module to switch to alternatePWA. Same command needs to be contributed twice in placement to support switching between view in three view layouts.

```
{
    "commands": {
        ...,
        "ChartPageCmd": {
            "iconId": "cmdListView",
            "title": "{{i18n.chartPageCmdTitle}}",
            "description": "{{i18n.chartPageCmdDescription}}"
        }
    },
    "commandHandlers": {
        ...,
        "chartPageHandler": {
            "id": "ChartPageCmd",
            "action": "setChartView",
            "activeWhen": true,
            "visibleWhen": true,
            "selectWhen": {
                "condition": "conditions.isChartViewSelected"
            }
        }
    },
    "commandPlacements": {
        ...,
        "chartPagePlacement": {
            "id": "ChartPageCmd",
            "parentGroupId": "Awp0DisplayToggles",
            "priority": 20
        },
        "ChartPageSummaryPlacement": {
            "id": "ChartPageCmd",
            "parentGroupId": "Awp0DisplayTogglesSummary",
            "priority": 20
        }
    },
    "actions": {
        ...,
        "setChartView":{
            "actionType": "JSFunction",
            "method": "changeViewMode",
            "deps": "js/pageLayoutService",
            "inputData": {
                "viewTupleState": "{{commandContext.viewTuple}}",
                "viewTuple": { "alternatePWA": "chartPage" },
                "sublocationState":
"{{commandContext.pageContext.sublocationState}}",
                "title": "Chart"
            }
        }
    },
    "conditions":{
```

```
        "isChartViewSelected":{
            "expression": "commandContext.viewTuple.alternatePWA ===
'chartPage'"
        }
    },
    "i18n": {
        ...,
        "chartPageCmdTitle": [ "TrackerAppMessages" ],
        "chartPageCmdDescription": [ "TrackerAppMessages" ]
    }
}
```

update i18n file `TrackerAppMessages.json` for newly added keys.

```
{
    ...,
    "chartPageCmdTitle": "Chart",
    "chartPageCmdDescription": "Switch to see data in chart view"
}
```

**Note**

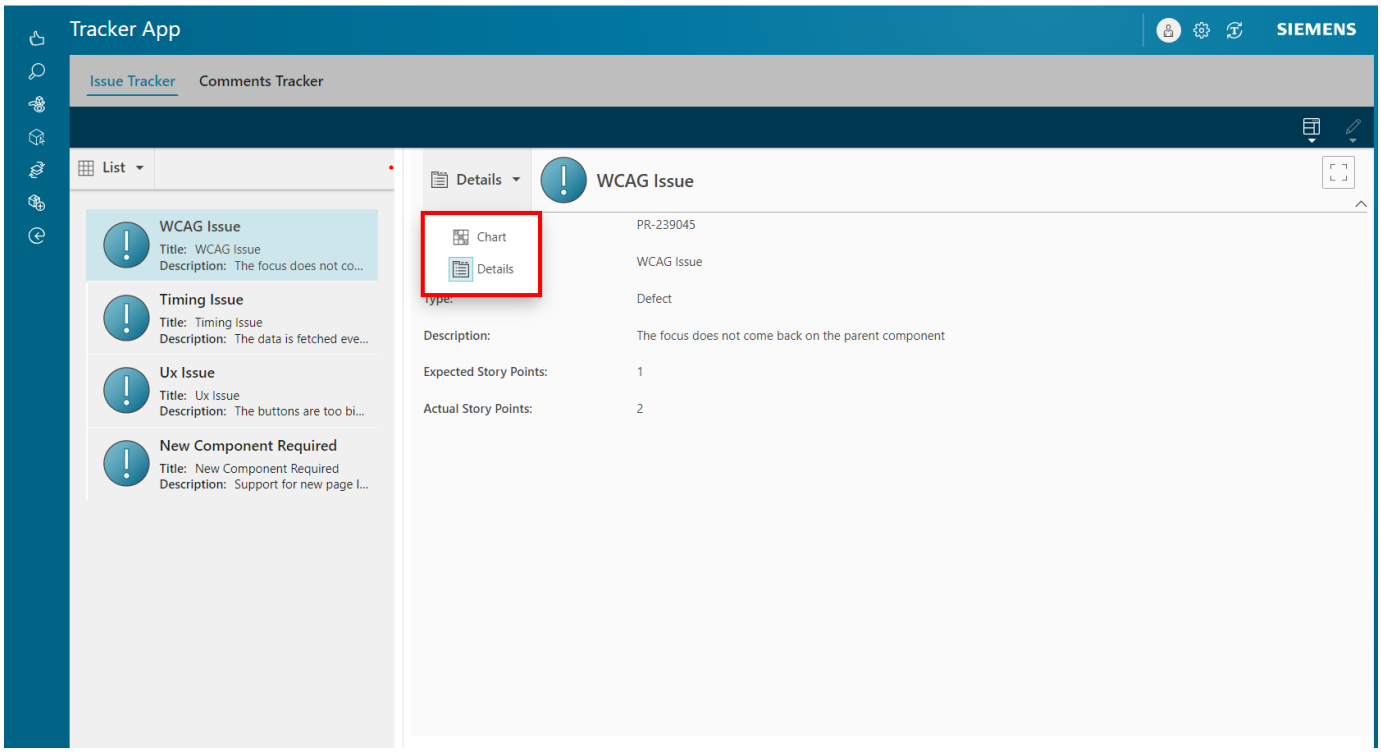
In the action to switch to alternatePWA, in the inputData viewTuple needs to be provided as below. Value for alternatePWA should be same as name of view-viewModel.

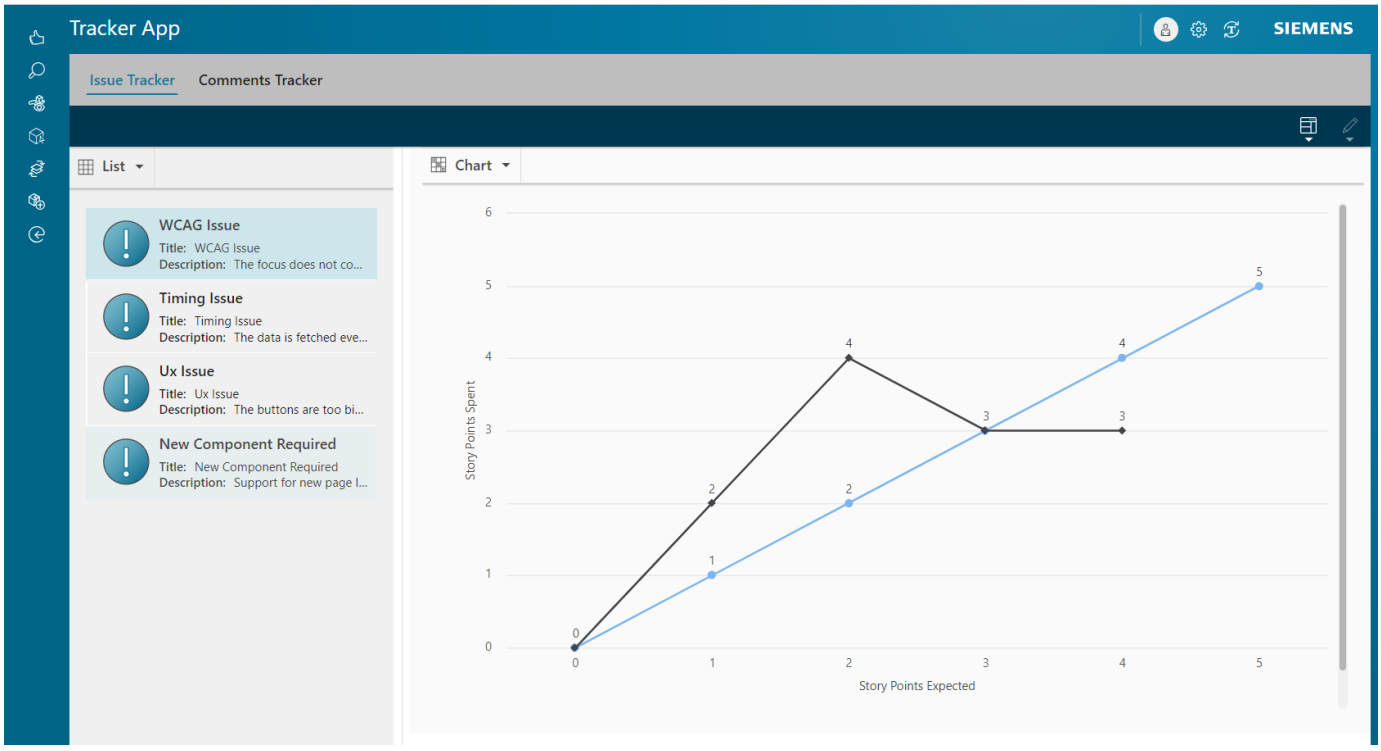
```
"inputData": {
   ...,
   "viewTuple": { "alternatePWA": "chartPage" },
}
```

With these steps you will see chart command on details command in left top of secondary workarea like following, which can be used to switch between the secondary workarea and alternate primary

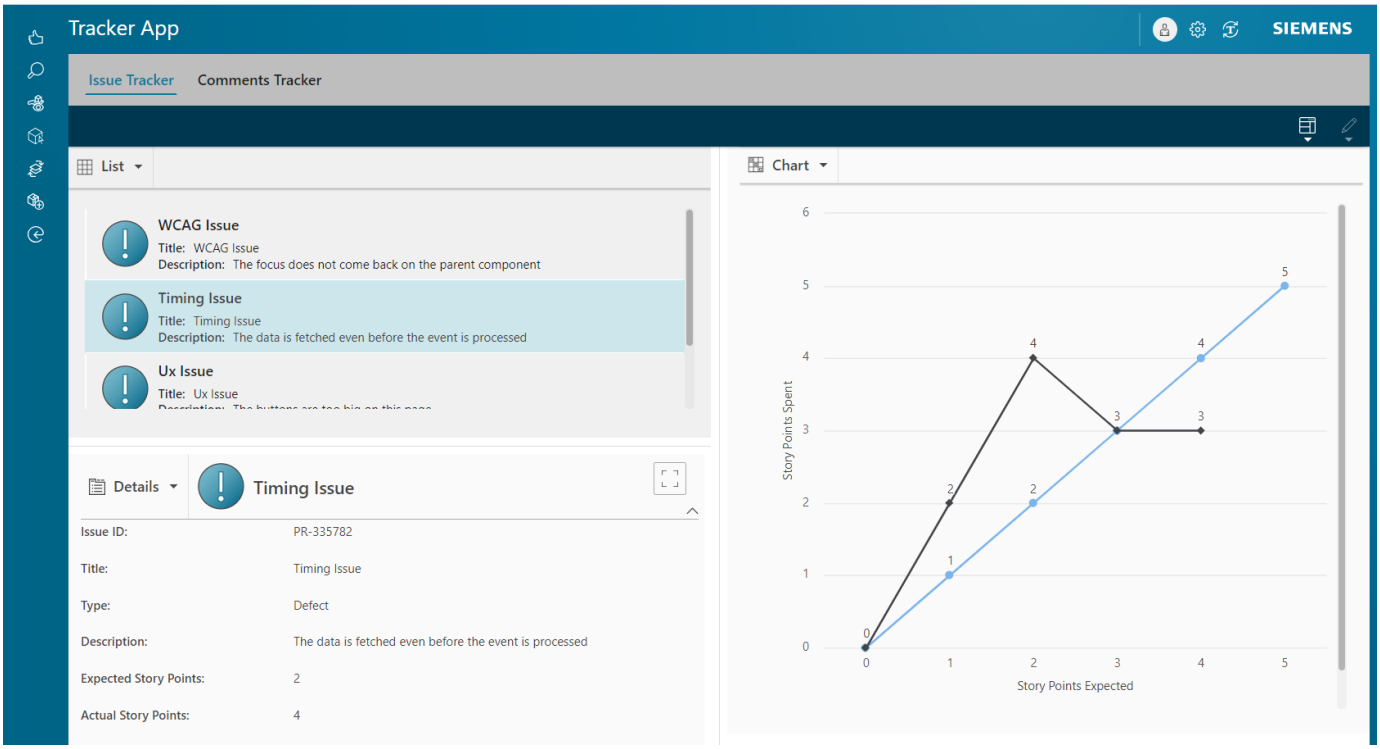

Siemens Web Framework
Unpublished work. © 2025 Siemens

workarea.


Tracker App
SIEMENS
Issue Tracker
Comments Tracker
List
WCAG Issue
Title: WCAG Issue
Description: The focus does not co...
Timing Issue
Title: Timing Issue
Description: The data is fetched eve...
Ux Issue
Title: Ux Issue
Description: The buttons are too bi...
New Component Required
Title: New Component Required
Description: Support for new page l...
Details
WCAG Issue
Chart
Details
PR-239045
WCAG Issue
Type:
Defect
Description:
The focus does not come back on the parent component
Expected Story Points:
1
Actual Story Points:
2


Tracker App
SIEMENS
Issue Tracker
Comments Tracker
List
WCAG Issue
Title: WCAG Issue
Description: The focus does not co...
Timing Issue
Title: Timing Issue
Description: The data is fetched eve...
Ux Issue
Title: Ux Issue
Description: The buttons are too bi...
New Component Required
Title: New Component Required
Description: Support for new page l...
Chart
Story Points Spent
Story Points Expected
0
1
2
3
4
5
6

One can switch to three view layout with layout chooser command on top and it will look like following




### 1. Selection synchronization

`s_uid` needs be configured under params in `states.json` to have sync in selection between alternate primary workarea and primary workarea. as following

```
{
   ...,
   "issueTracker": {
           ...,
           "params": {
               "s_uid": null
           }
   }
}
```

With this one can click on the point from chart and selection from primary workarea will get updated. For more details regarding **s_uid**

## What comes for free from Fx components (so far)

1. Commands for specified supportedLayouts
2. Swap functionality in three views for secondary workarea and alternate primary workarea.
3. Layout persistence if you move between locations or sublocations.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### What you can override (so far)

1. SupportedLayouts can be overridden as per need.
2. The views which should render when switched to alternate primary workarea .

## Configuring Commands

A **Command** is a UI component that enables users to trigger specific actions or operations within the application.

Siemens Web Framework supports declarative commands where user gets full flexibility to define the appearance, behavior, action and position of these commands.

The following tutorials provide some simple but useful examples on various types of commands which can be used to enrich your application:

- **Configuring command placement, visibility, and action**
- **Configuring an extended tooltip for a command**
- **Configuring context menu commands**
- **Reusing the same command in different places**
- **Configuring cell commands**
- **Configuring different behaviors for the same command**

> **Tip**
>
> These examples assume that the **previous tutorials** have been followed already and a **module** called **trackerApp** is already available in your example application.

## Configuring command placement, visibility, and action

Here is an example to demonstrate the management of commands:

### Adding a command

In your sample "trackerApp" module, add a **commandsViewModel.json** file at module level.

In `commandsViewModel.json``, add a section called "commands" and define a unique command object like this:

```
{
    "commands": {
        "Tkr0AddCmd": {
            "iconId": "cmdAdd",
            "title": "{{i18n.tkr0AddCmdTitle}}",
            "description": "{{i18n.tkr0AddCmdDescription}}"
        }
    }
}
```

Here, `Tkr0AddCmd` is the command definition for the example command that we are trying to add. It's appearance has been defined via iconId (the example uses an existing icon called `cmdAdd`), the localized title of the command as well as its description.

## Adding localization

Next, we have to add the localization for the title and description of the command defined in the command definition. For that, we need to do two things, first define the localization keys in the `commandsViewModel` in the `i18n` section. And second, define the actual localization in the corresponding message file (in this Tracker App example, we have already introduced "TrackerAppMessages.json" which we will continue to use, otherwise create a new one under the `i18n` folder and refer the same in the `commandsViewModel.json`).

This is the i18n section which needs to be added in commandsViewModel.json:

```
{
    "commands": {
        …
    },
    "i18n": {
        "tkr0AddCmdTitle": [ "TrackerAppMessages" ],
        "tkr0AddCmdDescription": [ "TrackerAppMessages" ]
    }
}
```

This is how the localization file `TrackerAppMessages.json`` would look like this:

```
{
    …,
    "tkr0AddCmdTitle": "Add",
    "tkr0AddCmdDescription": "Add a new object"
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Managing placement of the command

Now, we need to add the command to the page in the application. You can choose to add it to a predefined commandBars coming from SWF or a custom CommandBar component used in your application. This is governed by defining the commandPlacements as shown below:

```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        "Tkr0AddCmdPlacement": {
            "id": "Tkr0AddCmd",
            "uiAnchor": "aw_primaryWorkArea",
            "priority": 1000
        }
    }
}
```

Here, "Tkr0AddCmdPlacement" is a unique user-defined placement for the command defined earlier with unique Id "Tkr0AddCmd". This example is placing the command in one of the command anchors available from SWF in the **PrimaryWorkarea** called "aw_primaryWorkarea". The priority value defines the order at which the command is to be placed.

## Managing visibility of the command

Next, we need to add the behavior to the command i.e. when should the command be visible and what should happen when the command is clicked. We will first add the visibility conditions for the command by using standard **declarative conditions** in the "visibleWhen" clause of the handler that we will define for our newly added command.

```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        …
    },
    "commandHandlers": {
        "Tkr0AddCmdHandler": {
            "id": "Tkr0AddCmd",
            "action": "addTkrObject",
            "activeWhen": true,
            "visibleWhen": {
```

```
            "condition": "conditions.shouldShowAddCommand"
        }
    },
    "conditions":{
        "shouldShowAddCommand":{
            "expression": "!commandContext.selectionData.selected ||
commandContext.selectionData.selected.length === 0"
        }
    }
}
```

Here, **Tkr0AddCmdHandler** is a unique user-defined handler for the command added earlier with unique Id "Tkr0AddCmd". This example is using `true` in the `activeWhen` clause which means this handler for the command should be always active. We will learn more about defining multiple handlers for same command where we will revisit this clause to define different conditions under which each of these handlers should be active in **subsequent tutorials.**

In the `visibleWhen` clause, a declarative condition has been defined which controls the visibility of the command i.e. the command will be visible only when the specified condition is evaluated to true.

The condition **shouldShowAddCommand** used here is using `commandContext` which is the contextual object sent by commandBar to all the commands placed in it. As an application user, you have full control to choose what should be passed down to commandBar (and eventually commands) in this object and then later use it for any purpose say condition evaluation.

The example condition makes the command show up only when there is no explicit selection in the PrimaryWorkarea. But this is totally in application control and depends on business requirement.

## Managing action of the command

In the previous step in the command handler, against `actions` we referred something with name **addTkrObject**. This is the **declarative action** which should be executed when the command is invoked/ clicked.

Now, we are going to define this action using the standard actions schema as shown below:

```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        …
    },
    "commandHandlers": {
        …
    },
    "conditions":{
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        …
    },
    "actions": {
        "addTkrObject": {
            "actionType": "dialog",
            "inputData": {
                "options": {
                    "view": "tkr0AddDialog",
                    "caption": "Add",
                    "draggable": true,
                    "width": "STANDARD",
                    "height": "LARGE",
                    "parent": ".aw-layout-workareaMain",
                    "placement": "right"
                }
            }
        }
    }
}
```

Here, **addTkrObject** is a unique user-defined action which has been referred back in the command handler defined for the command added earlier.

This example is using a pre-defined "actionType" called `dialog` which launches a **dialog** on the click of the command and it renders "tkr0AddDialog" view in it. The standard **dialog options** have been used.

The example is rendering the view **tkr0AddDialog** in the dialog launched. Here are the sample contents of this view and viewModel but this is again an application code which can totally be modified as per business needs.

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textbox prop="data.description"></aw-textbox>
        <aw-listbox prop="data.types"></aw-listbox>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">Add</aw-button>
    </aw-panel-footer>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
```

```
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        }
    },
    "actions": {
        "add": {
            "actionType": "batchJob",
            "steps": [
                {
                    "action": "addData"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                },
                {
                    "action": "closeDialog"
                }
            ]
        },
        "addData": {
            "actionType": "JSFunction",
            "method": "addIssueData",
            "deps": "js/issueHelperService",
            "inputData": {
                "title": "{{data.title.dbValue}}",
                "description": "{{data.description.dbValue}}",
                "type": "{{data.types.dbValue}}"
            }
        },
        "reloadPrimaryWorkarea": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [
                    {
                        "name": "primaryWorkarea.reset"
                    }
                ]
            }
        },
        "closeDialog": {
            "actionType": "JSFunction",
            "method": "closeDialog",
            "deps": "js/dialogService",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

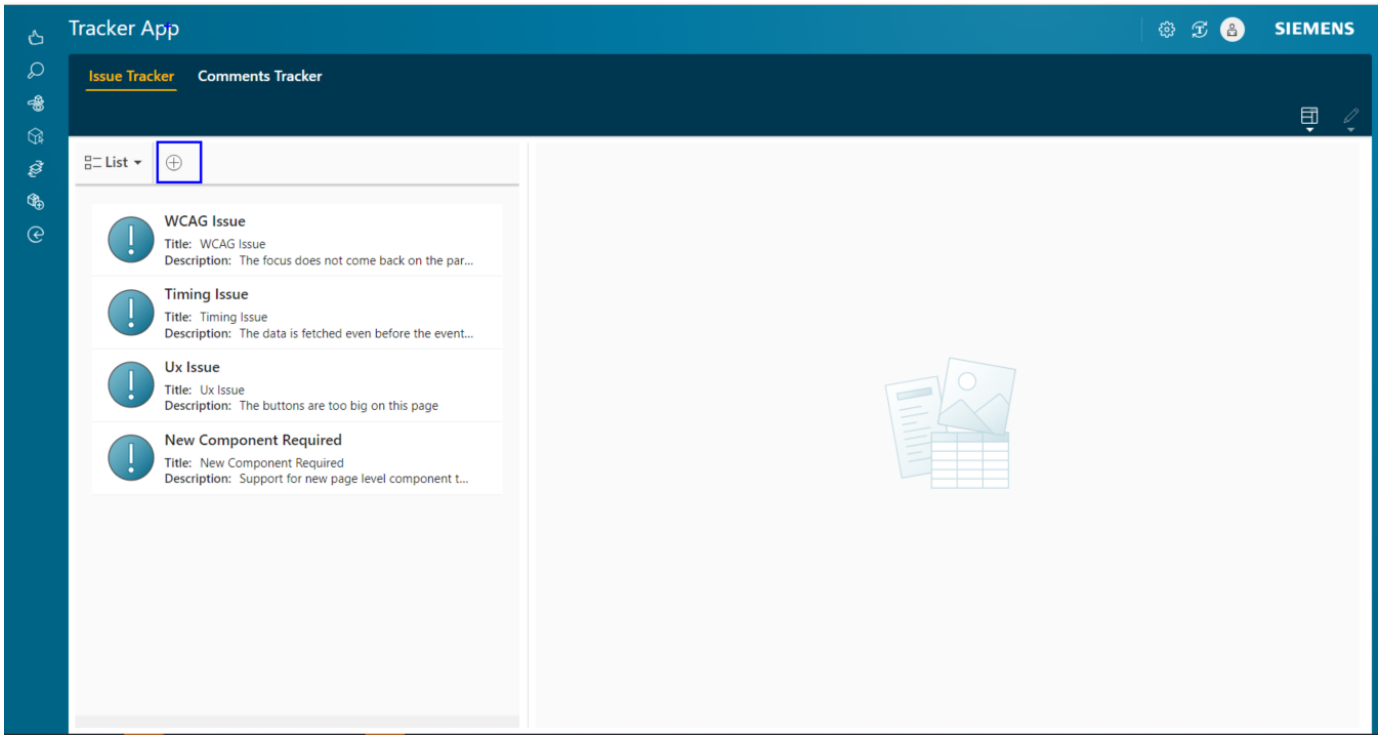
```
            "inputData": {
                "editContext": null,
                "popupId": "{{subPanelContext.popupOptions.popupId}}"
            }
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [
                {
                    "propDisplayValue": "Feature",
                    "dispValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "dispValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "dispValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "dispValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        }
    }
}
```

Here is the sample code for **issueHelperService.js** referred in code above. This is using IndexedDB, which is leveraged by the whole Tracker App to manage sample data on the client side.

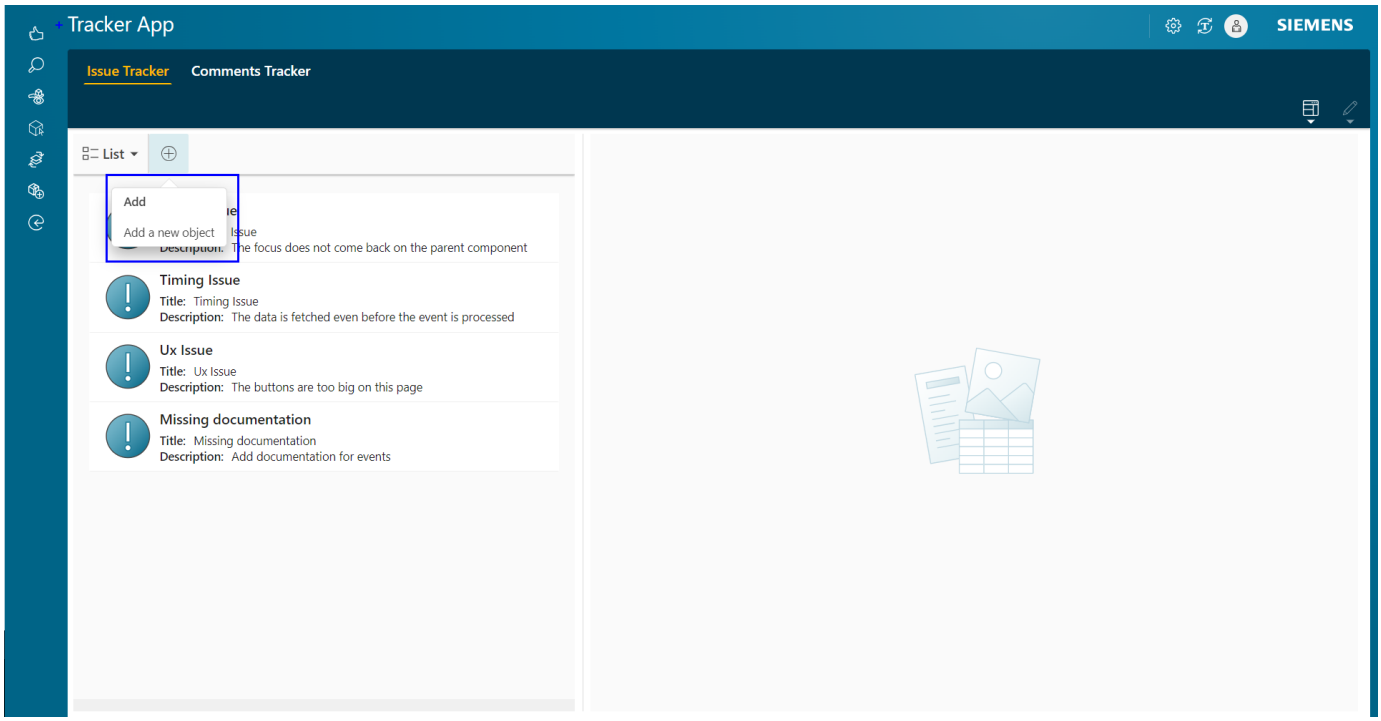
The logic here is just an implementation and is completely out of scope for our current discussion on commands.

```
export const addIssueData = async ( title, description, type ) => {
    let result = await getDB().addToStore( 'issue', {
        IssueID: 'PR-' + Math.floor( Math.random() * 1000000 ),
        Title: title,
        Description: description,
        TypeIcon: 'assets/images/typeIssue48.svg',
        Type: type,
        ExpectedStoryPoints: Math.floor( Math.random() * 10 ),
        ActualStoryPoints: Math.floor( Math.random() * 10 )
    } );
    return Promise.resolve( { data: { result: result } } );
};
```

When you build your application, refresh the browser and hit the url of your application; you can see the command got added with the specified `iconId`, at the specified command bar anchor:



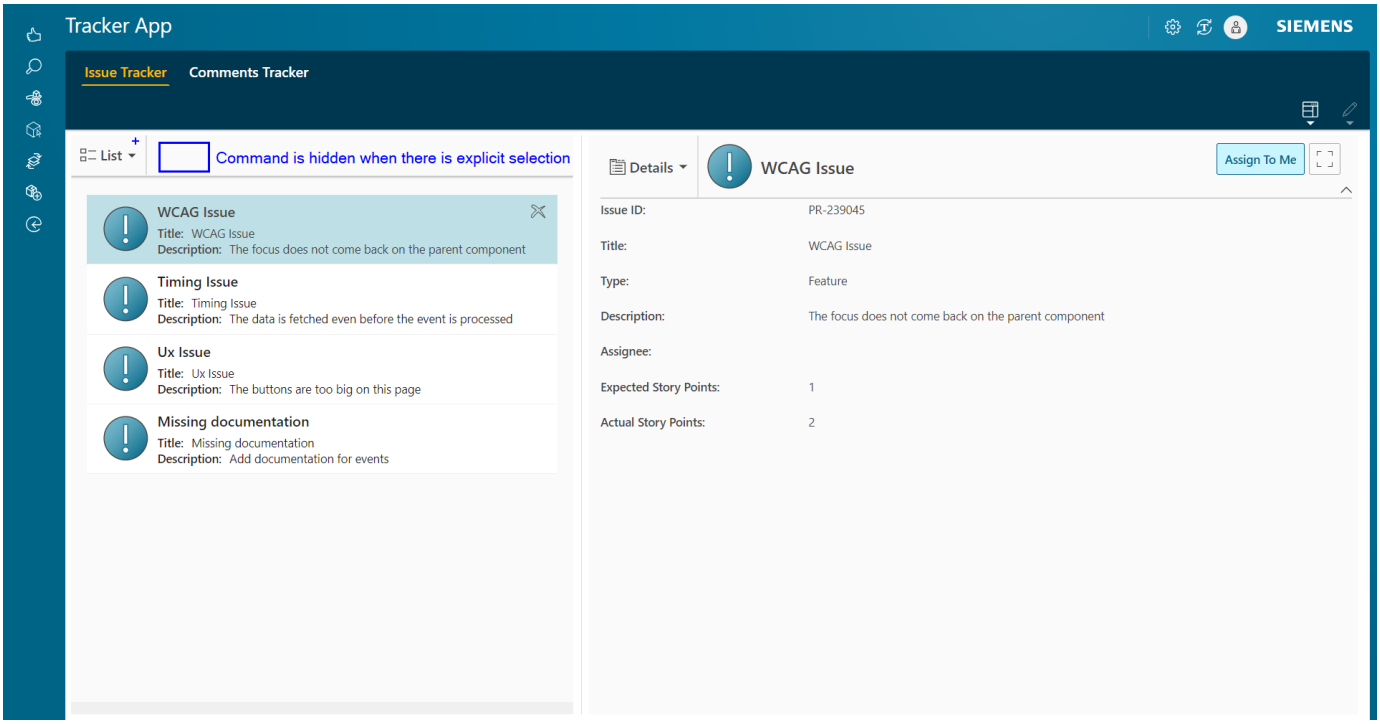

It shows the specified tooltip (made up of title and description) on hover:



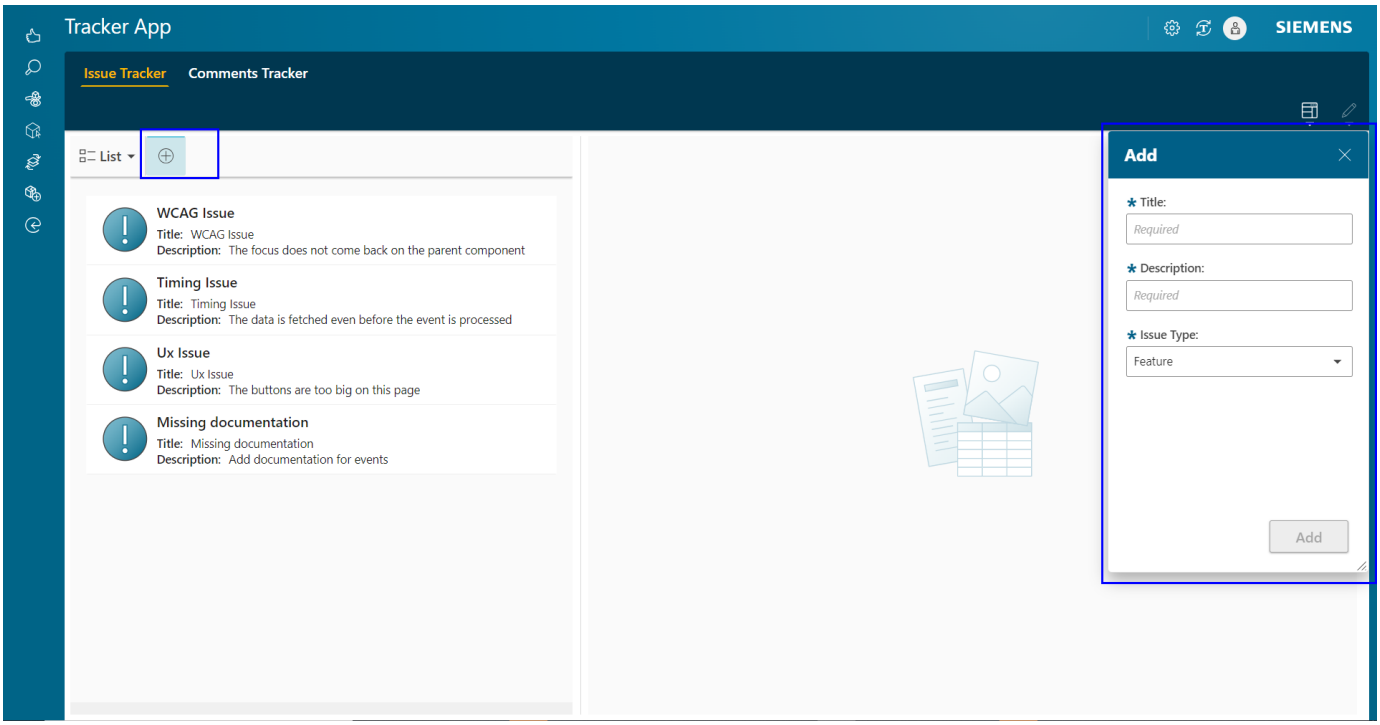


Siemens Web Framework
Unpublished work. © 2025 Siemens

It is visible only when there is nothing selected in the PWA and hides itself when something is selected from PWA:




When the command is clicked, you should see the dialog with specified dialog options launched rendering the view configured via "tkr0AddDialog":



The behavior of what should happen when the **Add** button in this view is clicked is out of scope for commands discussion. This example shows one implementation but there could be many more.

## Configuring an extended tooltip for a command

By default, the `title` and the `description` defined on the command defines its tooltip which is shown when user hovers over the command.

However, if you want to show more details and information about the command, you can leverage **extended tooltips**.

Here is an example to demonstrate the addition of extended tooltip on a command.

### Adding an extended tooltip

In the `commands` section of the **commandsViewModel.json**, add an extended tooltip as shown below:

```
{
    "commands": {
        "Tkr0AddCmd": {
            "iconId": "cmdAdd",
            "title": "{{i18n.tkr0AddCmdTitle}}",
            "description": "{{i18n.tkr0AddCmdDescription}}",
            "extendedTooltip": {
                "view": "tkr0AddTooltip"
            }
        }
    }
}
```

Here, **tkr0AddTooltip** is the declarative view which will get rendered on command hover instead of the regular tooltip.

### Adding the tooltip component

The **tkr0AddTooltip** is the name of the custom component that can be written as per the business requirement. For the tutorial example, following is the content added in the view and viewModel:

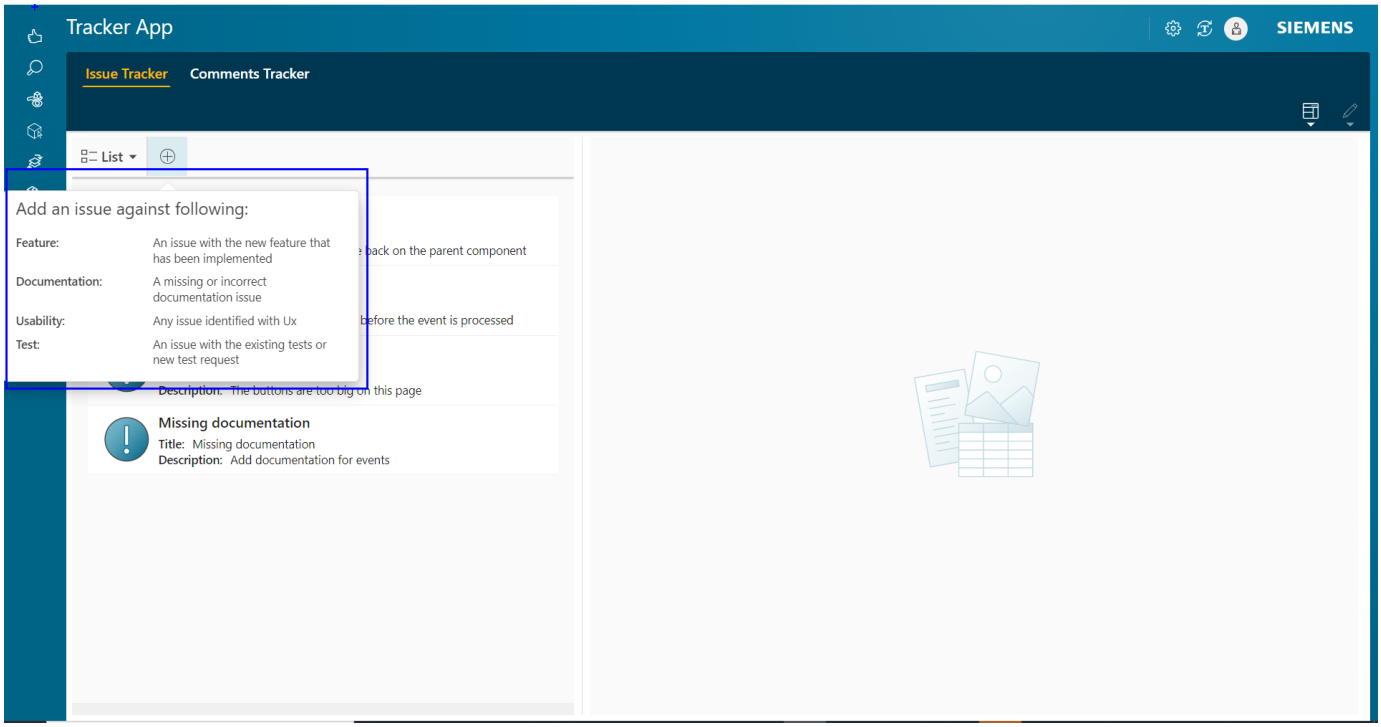
```
<h4><b>Add an issue against following:</b></h4>
<br>
<aw-label prop="data.feature"></aw-label>
<aw-label prop="data.documentation"></aw-label>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
<aw-label prop="data.usability"></aw-label>
<aw-label prop="data.test"></aw-label>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "feature": {
            "displayName": "Feature",
            "type": "STRING",
            "isRequired": "false",
            "dbValue": "Feature",
            "dispValue": "An issue with the new feature that has been
implemented"
        },
        "documentation": {
            "displayName": "Documentation",
            "type": "STRING",
            "isRequired": "false",
            "dbValue": "Documentation",
            "dispValue": "A missing or incorrect documentation issue"
        },
        "usability": {
            "displayName": "Usability",
            "type": "STRING",
            "isRequired": "false",
            "dbValue": "Usability",
            "dispValue": "Any issue identified with Ux"
        },
        "test": {
            "displayName": "Test",
            "type": "STRING",
            "isRequired": "false",
            "dbValue": "Test",
            "dispValue": "An issue with the existing tests or new test
request"
        }
    }
}
```

After rebuilding, refresh the browser and hit the url of your application; you can see the command with the newly added extended tooltip:




The content of what needs to be shown in the extended tooltip is as per the business need.

## Configuring context menu commands

The commands can be optionally added in the **context menu** of list or table components such as that, once configured, they will show up when the user performs a right-click operation.

Here is an example to demonstrate the addition of commands in the context menu of the list that we have already created in the Tracker App application.

### Enabling the context menu on list

Enable the context menu by adding the prop called `show-context-menu` on the list/table component. In our example, the following change needs to be made in the issueListView.html:

```
<aw-panel>
    <aw-list dataprovider="data.dataProviders.getIssues" command-
context="props.subPanelContext" show-context-menu="true">
        <aw-default-cell vmo="item"></aw-default-cell>
    </aw-list>
</aw-panel>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Defining the context-menu anchor

The context menu anchor needs to be defined in the `dataProviders` of the list or table component. In our example, the following change needs to be made in the `issueListViewModel.json` file:

```
{
    "schemaVersion": "1.0.0",
    "dataProviders": {
        "getIssues": {
             …
            "contextMenuCommandsAnchor": "issue_ContextMenu"
        }
    }
}
```

Here **issue_ContextMenu** is the name of the example anchor that has been added to the dataProvider being used for rendering the issue example list.

## Adding command to the Context-menu anchor

Add the command of your choice to the Context-menu anchor defined in previous steps (**issue_ContextMenu**). All the steps are same except that the anchor name which should be same as what has been defined above. The `commandsViewModel.json` would look like this:

```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        "Tkr0AssignCmdPlacement": {
            "id": "Tkr0AssignCmd",
            "uiAnchor": "issue_ContextMenu",
            "priority": 1000
        }
    },
    "commandHandlers": {
        …
    },
    "conditions":{
        …
    },
    "actions": {
        …
    }
}
```

Here, is the full code for **Tkr0AssignCmd** to see it function in the Tracker App.

```
{
    "commands": {
        "Tkr0AssignCmd": {
            "title": "{{i18n.tkr0AssignCmdTitle}}",
            "description": "{{i18n.tkr0AssignCmdDescription}}"
        }
    },
    "i18n": {
        "tkr0AssignCmdTitle" : [ "TrackerAppMessages" ],
        "tkr0AssignCmdDescription" : [ "TrackerAppMessages" ]
    },
    "commandPlacements": {
        "Tkr0AssignCmdPlacement": {
            "id": "Tkr0AssignCmd",
            "uiAnchor": "issue_ContextMenu",
            "priority": 1000
        }
    },
    "commandHandlers": {
        "Tkr0AssignCmdHandler": {
            "id": "Tkr0AssignCmd",
            "action": "assignTkrObject",
            "activeWhen": true,
            "visibleWhen": {
                "condition": "conditions.shouldShowAssignCommand"
            }
        }
    },
    "conditions":{
        "shouldShowAssignCommand":{
            "expression":
"commandContext.selection[0].props.Assignee.dbValue !== 'Ed' &&
commandContext.selectionData.selected[0].props.Assignee.dbValue !== 'Ed'"
        }
    },
    "actions": {
        "assignTkrObject": {
            "actionType": "batchJob",
            "steps": [
                {
                    "action": "assignTkrObjectToMe"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                }
            ]
        },
        "assignTkrObjectToMe":{
            "actionType": "JSFunction",
            "method": "assignIssue",
            "deps": "js/issueHelperService",
            "inputData": {
            "issueId": "{{commandContext}}"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            }
        },
        "reloadPrimaryWorkarea":{
            "actionType": "Event",
            "method": "Event",
            "inputData": {
            "events": [ {
                "name": "primaryWorkarea.reset"
                } ]
            }
        }
    }
}
```

This example assumes that a user called *Ed* is the currently logged in user which is why **Assign to Me** assigns the issue to user with name "Ed".

This example assumes that the **Assignee** property is available on the items in the list. If it is not available already, make sure, you make it available by adding this on the dataParserDefinition defined in `issueListViewModel.json`:

```
{
    …
    "dataParseDefinitions": {
        "issueDataParser": {
            "identifier": "{{response.uid}}",
            "cellHeader1": "{{response.Title}}",
            "typeIconURL": "{{response.TypeIcon}}",
            "cellProperties": {
                "Title": {
                    "key": "Title",
                    "displayName": "Title",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Description": {
                    "key": "Description",
                    "displayName": "Description",
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                }
            },
            "props": {
                "Title":{
                    "displayName": "{{response.Title}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                },
                "Description":{
                    "displayName": "{{response.Description}}",
```

```
                    "value": "{{response.Description}}",
                    "displayValue": "{{response.Description}}",
                    "type": "String"
                },
                "IssueID":{
                    "displayName": "{{response.IssueID}}",
                    "value": "{{response.IssueID}}",
                    "displayValue": "{{response.IssueID}}",
                    "type": "String"
                },
                "Assignee":{
                    "displayName": "{{response.Assignee}}",
                    "value": "{{response.Assignee}}",
                    "displayValue": "{{response.Assignee}}",
                    "type": "String"
                },
                "object_string":{
                    "displayName": "{{response.Title}}",
                    "value": "{{response.Title}}",
                    "displayValue": "{{response.Title}}",
                    "type": "String"
                }
            }
        }
    }
    …
}
```

Here is the sample code for `issueHelperService.js` referred in code above. This is using IndexedDB which is leveraged by Tracker App to manage sample client-side data.

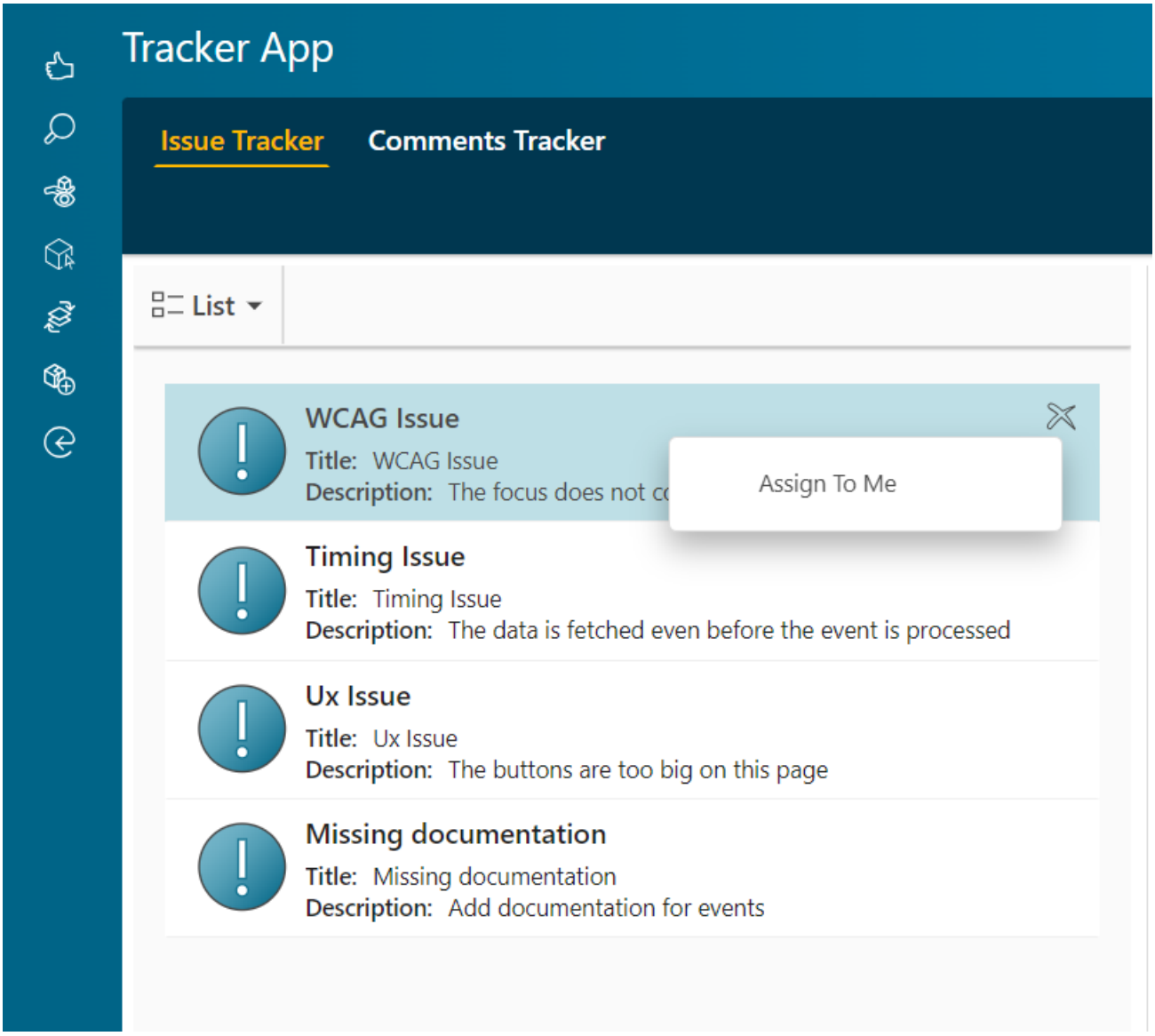
The logic here is just an implementation and is completely out of scope for our current discussion on commands.

```
const handleAssignments = async ( commandContext, assignee ) => {
    let issueId = commandContext?.selection?.[0]?.uid ||
commandContext.selectionData?.selected[0]?.uid;
    let object = await getDB().getFromStore( 'issue', issueId );
    object.Assignee = assignee;
    let result = await getDB().updateStore( 'issue', object );
    return Promise.resolve( { data: { result: result } } );
};

export const assignIssue = ( commandContext ) => {
    return handleAssignments( commandContext, 'Ed' );
};
```

After rebuilding, refresh the browser and hit the url of your application; you can see the context menu command on doing right click on the items in the list:


Siemens Web Framework
Unpublished work. © 2025 Siemens




The business logic to determine the visibility of the command and action of the command is as per the business need.

## Reusing the same command in different places

The commands can be optionally placed at different places in the same page. This can be simply achieved by adding more command placements.

Here is an example to demonstrate the second placement of the **Assign to Me** command added in previous tutorial in the context menu of the list. The example shows how to reuse the same command in the summary header of the details section of the application.

If same command needs to be exposed at multiple places in the application in different commandBars, the only configuration required to achieve this is to add another placement for the same command:

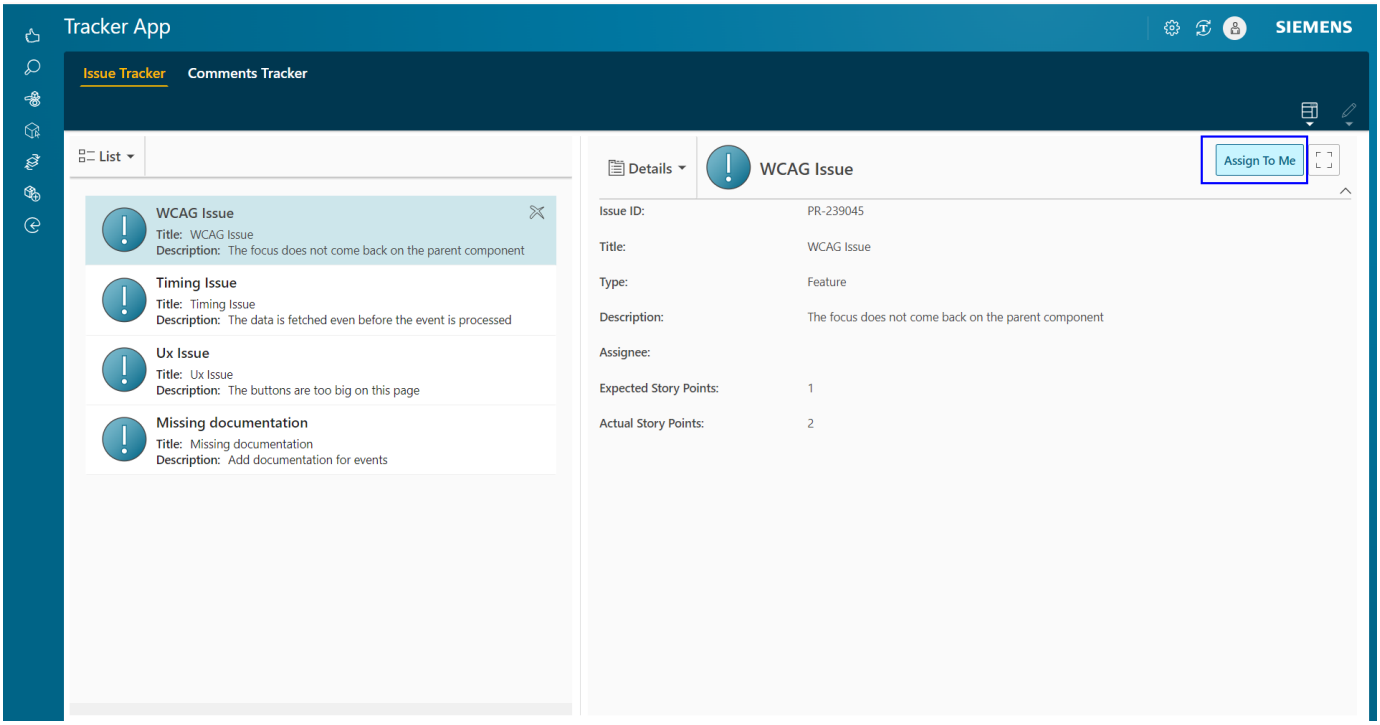
```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        "Tkr0AssignCmdPlacement": {
            "id": "Tkr0AssignCmd",
            "uiAnchor": "issue_ContextMenu",
            "priority": 1000
        },
        "Tkr0AssignCmdPlacementOnSummaryHeader": {
            "id": "Tkr0AssignCmd",
            "uiAnchor": "aw_summaryHeader",
            "priority": 1000,
            "style": "primary",
            "showTitle": "always"
        }
    },
    "commandHandlers": {
        …
    },
    "conditions":{
        …
    },
    "actions": {
        …
    }
}
```

This example adds the **Tkr0AssignCmd** to another existing placement called `aw_summaryHeader` which shows up in the Details section of the SecondaryWorkarea. By this simple addition of the placement, the command should be available and functional in different commandBar too. As a user, make sure to pass the right `commandContext` to the command such that it gets the correct information at correct path to evaluate visibility and perform action correctly.

This example also demonstrates additional command features like `style` which renders the command with the Ux of a button and `showTitle` which renders the command with label always shown. These additional configurations are available for all types of commands.

When you build your application, refresh the browser and hit the `url` of your application; you can see the additional placement of **Assign to Me** command on the summary header:


Siemens Web Framework
Unpublished work. © 2025 Siemens




# Configuring cell commands

The commands can be optionally used in the cells of list/table; such commands are called **cell commands**.

Here is an example to demonstrate the configuration of a cell command called **Delete** in the list used in our Tracker App:

## Defining the cell command anchor

The cell command anchor needs to be defined in the dataProvider of the list or table component. In our example, the following change needs to be made in the `issueListViewModel.json`.

```
{
    ...
    "dataProviders": {
        "getIssues": {
            …
            "commandsAnchor": "issue_cellCommands"
        }
    }
}
```

Here **issue_cellCommands** is the name of the example anchor that has been added to the dataProvider being used for rendering the issue example list.

### Adding the command to the cell command anchor

Add the command of your choice to the Cell Command anchor defined in previous steps (**issue_cellCommands**). All the steps are the same except that the anchor name which should be same as what has been defined above. The `commandsViewModel.json` would look like this:

```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        "Tkr0DeleteCmdPlacement":{
            "id": "Tkr0DeleteCmd",
            "uiAnchor": "issue_cellCommands",
            "priority": 1000,
            "cellDisplay": {
            "hover": true,
                "position": "TOP_RIGHT",
                "selected": true
            }
        }
    },
    "commandHandlers": {
        …
    },
    "conditions":{
        …
    },
    "actions": {
        …
    }
}
```

Here, the section `cellDisplay` in the command placement defines the extra properties like the fact that the command is supposed to show up on hover and it is supposed to show up on top right of the cell.

Here, is the full code for **Tkr0DeleteCmd** to see it function in the Tracker App.

```
{
    "commands": {
        "Tkr0DeleteCmd": {
            "iconId": "cmdDelete",
            "title": "{{i18n.tkr0DeleteCmdTitle}}",
            "description": "{{i18n.tkr0DeleteCmdDescription}}"
        }
    },
    "commandHandlers": {
        "Tkr0DeleteIssueCmdHandler": {
            "id": "Tkr0DeleteCmd",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "action": "handleIssueDeletion",
            "activeWhen": true
            "visibleWhen": true
        }
    },
   "commandPlacements": {
        "Tkr0DeleteCmdPlacement":{
            "id": "Tkr0DeleteCmd",
            "uiAnchor": "issue_cellCommands",
            "priority": 1000,
            "cellDisplay": {
                "hover": true,
                "position": "TOP_RIGHT",
                "selected": true
            }
        }
    },
    "onEvent": [ {
        "eventId": "confirmIssueDeletionEvent",
        "message": "confirmIssueDeletion"
    } ],
    "actions": {
        "handleIssueDeletion": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "confirmIssueDeletionEvent"
                } ]
            }
        },
        "deleteIssueObject": {
            "actionType": "batchJob",
            "steps": [
                {
                    "action": "deleteIssue"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                }
            ]
        },
        "deleteIssue":{
            "actionType": "JSFunction",
            "method": "deleteObject",
            "deps": "js/issueHelperService",
            "inputData": {
                "store": "issue",
                "objectId": "{{commandContext.vmo.uid}}"
            }
        },
    "reloadPrimaryWorkarea":{
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
```

```
                    "name": "primaryWorkarea.reset"
                } ]
            }
        }
    },
    "messages": {
        "confirmIssueDeletion": {
            "messageType": "WARNING",
            "messageText": "{{i18n.deleteIssueConfirmation}}",
            "navigationOptions": [ {
                    "option": "cancel",
                    "text": "{{i18n.cancel}}",
                    "action": ""
                },
                {
                    "option": "delete",
                    "text": "{{i18n.delete}}",
                    "action": "deleteIssueObject"
                }
            ]
        }
    },
    "conditions":{
    },
    "i18n": {
        "tkr0DeleteCmdTitle": [ "TrackerAppMessages" ],
        "tkr0DeleteCmdDescription": [ "TrackerAppMessages" ],
        "deleteIssueConfirmation": [ "TrackerAppMessages" ],
        "cancel": [ "TrackerAppMessages" ],
        "delete": [ "TrackerAppMessages" ]
    }
}
```

This example is using the data available in `commandContext`. The business logic assumed in this example is that clicking on the Delete command will first show a confirmation of whether or not to really Delete the selected item. The actual deletion happens only when the delete confirmation is provided. This is again only an implementation and has been included here just as an example.

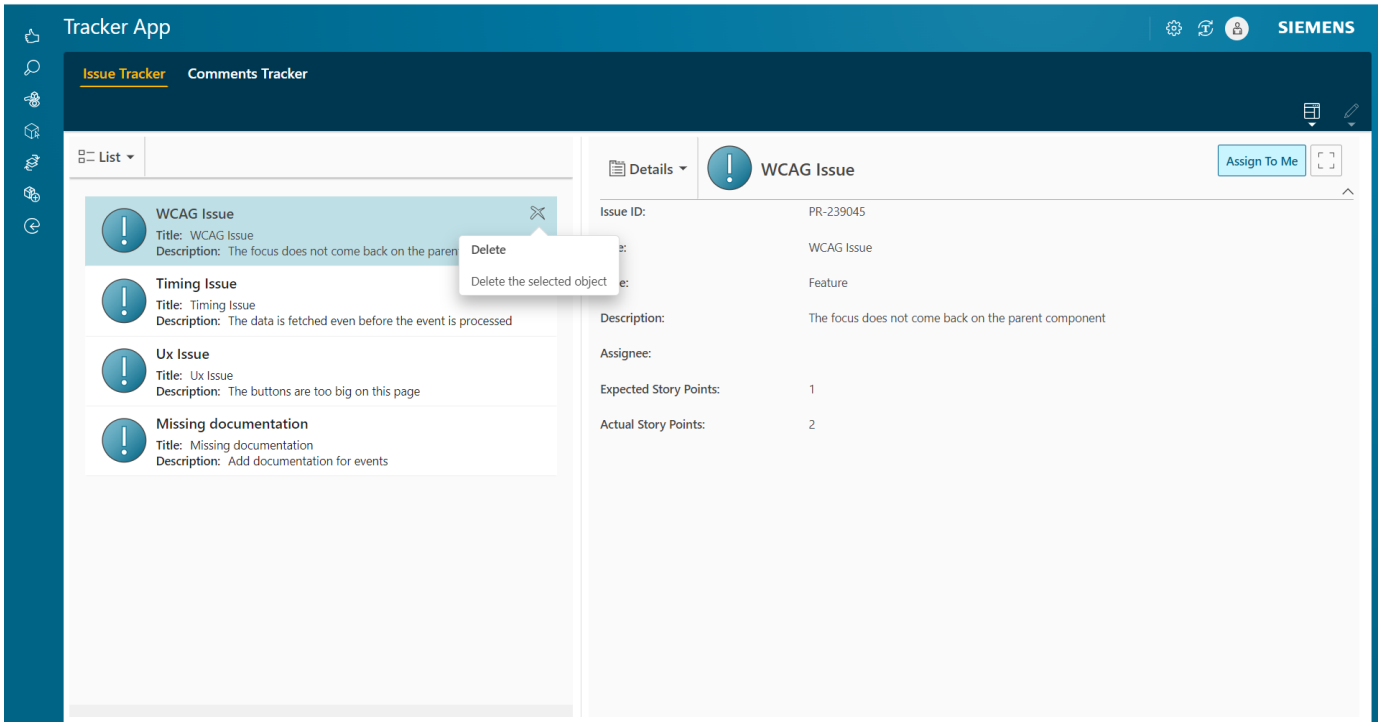
Here is the sample code for "issueHelperService.js" referred in code above. This is using IndexedDB which is leveraged by Tracker App to manage sample client-side data.

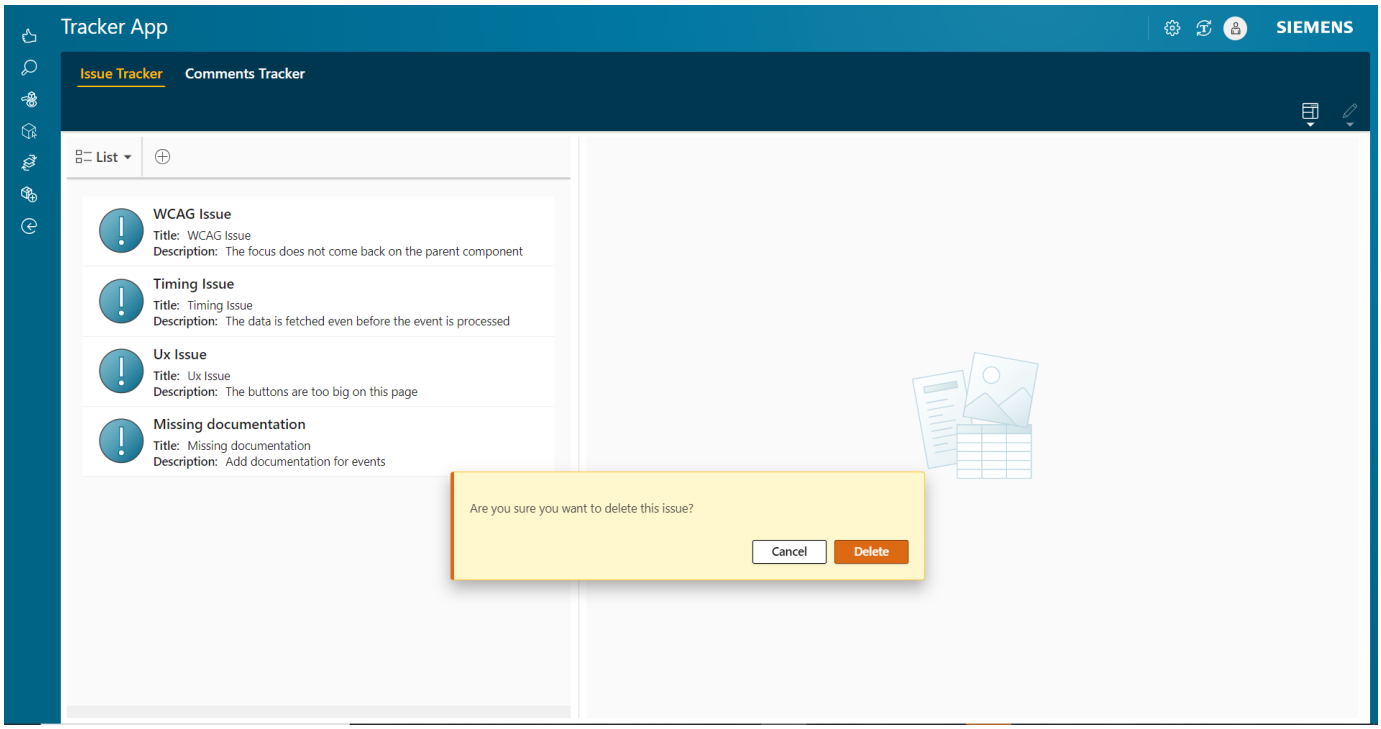
The logic here is just an implementation and is completely out of scope for our current discussion on commands.

```
export const deleteObject = async ( store, objectId ) => {
    let result = await getDB().deleteFromStore( store, objectId );
    return Promise.resolve( { data: { result: result } } );
};
```

After rebuilding, refresh the browser and hit the url of your application; you can see the Cell command on the items in the list:


Siemens Web Framework
Unpublished work. © 2025 Siemens




The business logic to determine the visibility of the command and action of the command is as per the business need.

# Configuring different behaviors for the same command

Depending on the business needs, you may want to reuse the same command but have them perform different behaviors under different circumstances. The same can be achieved via overriding the command handlers defined for the command.

The example here expands the one taken in previous one where we added a **Delete** command on cell for deleting an issue. This was done in the **IssueTracker** sublocation where the expected behavior was to show a confirmation before the actual deletion can happen.

Let's assume we need the same command (i.e same appearance, same placement) but a slightly different behavior in the **CommentsTracker** sublocation. Here, maybe we don't need to show the confirmation and direct delete is the business need.

This can be achieved by simply defining a separate command handler for this sublocation which should become active only when user is under **CommentsTracker** sublocation.

Here is an example to demonstrate managing the same command with different behaviors:

## Defining a new command handler

A new command handler needs to be added to the same command like shown below:

```
{
    "commands": {
        …
    },
    "i18n": {
        …
    },
    "commandPlacements": {
        …
    },
    "commandHandlers": {
        "Tkr0DeleteCommentsCmdHandler": {
            "id": "Tkr0DeleteCmd",
            "action": "deleteCommentObject",
            "activeWhen": {
                "condition": "conditions.isCommentsSublocation"
            },
            "visibleWhen": true
        }
    },
    "conditions":{
        "isCommentsSublocation":{
            "expression": " commandContext.context.nameToken ===
'CommentsTracker'"
        },
    },
    "actions": {
```



Unpublished work. © 2025 Siemens

```
        …
    }
}
```

Here, a new command handler has been added for the same command which had a previously defined handler which does behavior i.e. of showing a confirmation before the actual delete can happen. This new handler has been provided a condition such that it instructs the system clearly when should this new behavior (of direct delete) should apply.

## Adding conditions to the handler

We have to ensure that the `activeWhen` conditions for these handlers have to be mutually exclusive to ensure there is no ambiguity in resolving which handler is active at a certain point in time. This is why, it is important to add the `activeWhen` condition to the previous handler as well:

```
{
    "commands": {
        …
    },
    "i18n": {
    …
    },
    "commandPlacements": {
        …
    },
    "commandHandlers": {
        "Tkr0DeleteIssueCmdHandler": {
            "id": "Tkr0DeleteCmd",
            "action": "handleIssueDeletion",
            "activeWhen": {
                "condition": "conditions.isIssueSublocation"
            },
            "visibleWhen": true
        },
        "Tkr0DeleteCommentsCmdHandler": {
            "id": "Tkr0DeleteCmd",
            "action": "deleteCommentObject",
            "activeWhen": {
                "condition": "conditions.isCommentsSublocation"
            },
            "visibleWhen": true
        }
    },
    "conditions":{
        "isIssueSublocation":{
            "expression": "commandContext.context.nameToken ===
'IssueTracker5'"
        },
        "isCommentsSublocation":{
        "expression": " commandContext.context.nameToken ===
'CommentsTracker'"
        },
```

```
    },
    "actions": {
        …
    }
}
```

Here both the handlers have full flexibility to define their visibility conditions as well as the action that is to be performed when the command is invoked under different contexts.

Here, is the full code for **Tkr0DeleteCmd** to see it function in the Tracker App.

```
{
    "commands": {
        "Tkr0DeleteCmd": {
            "iconId": "cmdDelete",
            "title": "{{i18n.tkr0DeleteCmdTitle}}",
            "description": "{{i18n.tkr0DeleteCmdDescription}}"
        }
    },
    "commandHandlers": {
        "Tkr0DeleteIssueCmdHandler": {
            "id": "Tkr0DeleteCmd",
            "action": "handleIssueDeletion",
            "activeWhen": {
                "condition": "conditions.isIssueSublocation"
            },
            "visibleWhen": true
        },
        "Tkr0DeleteCommentsCmdHandler": {
            "id": "Tkr0DeleteCmd",
            "action": "deleteCommentObject",
            "activeWhen": {
                "condition": "conditions.isCommentsSublocation"
            },
            "visibleWhen": true
        }
    },
    "commandPlacements": {
        "Tkr0DeleteCmdPlacement":{
            "id": "Tkr0DeleteCmd",
            "uiAnchor": "issue_cellCommands",
            "priority": 1000,
            "cellDisplay": {
                "hover": true,
                "position": "TOP_RIGHT",
                "selected": true
            }
        }
    },
    "onEvent": [ {
        "eventId": "confirmIssueDeletionEvent",
        "message": "confirmIssueDeletion"
    } ],
```

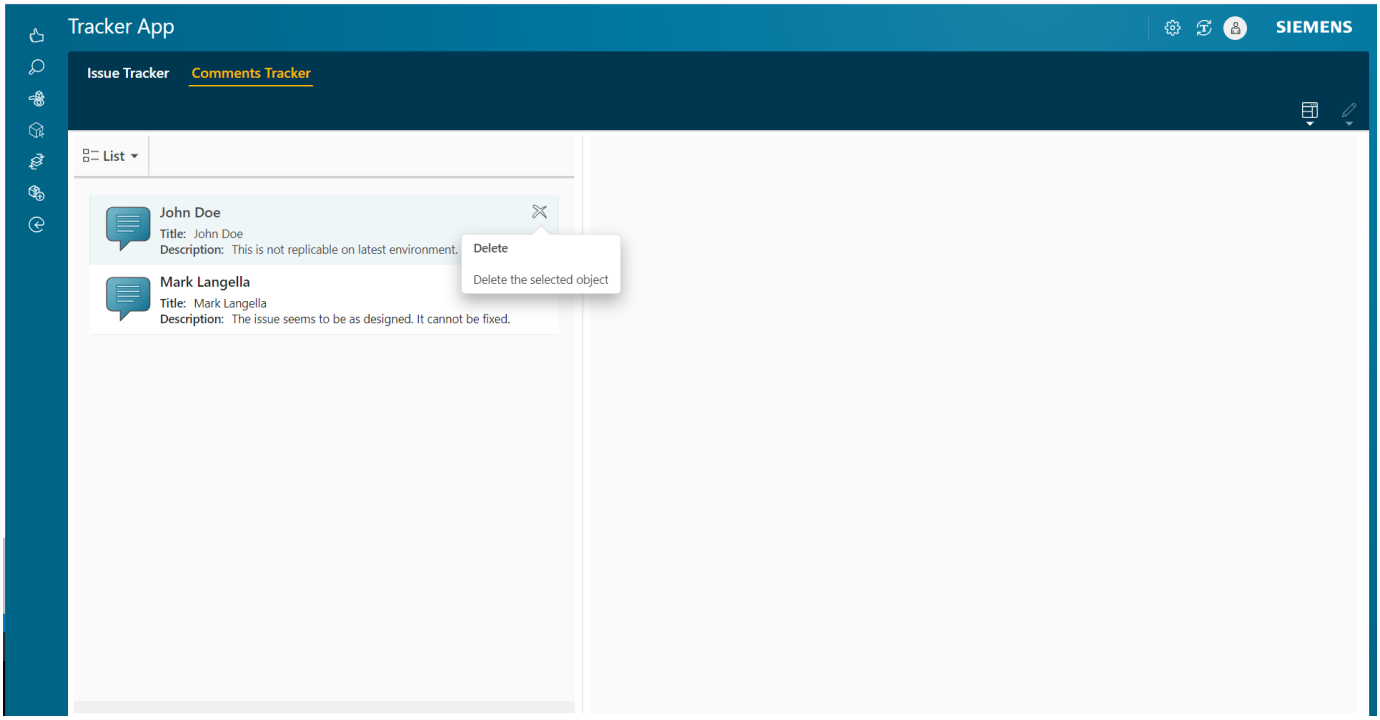

Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "actions": {
        "handleIssueDeletion": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "confirmIssueDeletionEvent"
                } ]
            }
        },
        "deleteIssueObject": {
            "actionType": "batchJob",
            "steps": [
                {
                    "action": "deleteIssue"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                }
            ]
        },
        "deleteIssue":{
            "actionType": "JSFunction",
            "method": "deleteObject",
            "deps": "js/issueHelperService",
            "inputData": {
                "store": "issue",
                "objectId": "{{commandContext.vmo.uid}}"
            }
        },
    "reloadPrimaryWorkarea":{
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "primaryWorkarea.reset"
                } ]
            }
        },
        "deleteCommentObject":{
            "actionType": "batchJob",
            "steps": [
                {
                    "action": "deleteComment"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                }
            ]
        },
        "deleteComment":{
            "actionType": "JSFunction",
            "method": "deleteObject",
            "deps": "js/issueHelperService",
            "inputData": {
                "store": "comments",
```

```
                "objectId": "{{commandContext.vmo.uid}}"
            }
        }
    },
    "messages": {
        "confirmIssueDeletion": {
            "messageType": "WARNING",
            "messageText": "{{i18n.deleteIssueConfirmation}}",
            "navigationOptions": [ {
                    "option": "cancel",
                    "text": "{{i18n.cancel}}",
                    "action": ""
                },
                {
                    "option": "delete",
                    "text": "{{i18n.delete}}",
                    "action": "deleteIssueObject"
                }
            ]
        }
    },
    "conditions":{
        ...
    },
    "i18n": {
        "tkr0DeleteCmdTitle": [ "TrackerAppMessages" ],
        "tkr0DeleteCmdDescription": [ "TrackerAppMessages" ],
        "deleteIssueConfirmation": [ "TrackerAppMessages" ],
        "cancel": [ "TrackerAppMessages" ],
        "delete": [ "TrackerAppMessages" ]
    }
}
```

After rebuilding, refresh the browser and hit the url of your application; you can see the **Delete** command perform different behavior when operated on Issue vs Comments:


Siemens Web Framework
Unpublished work. © 2025 Siemens




The business logic to determine the visibility of the command and action of the command is as per the business need.

## Managing Dialogs, Property Widgets, Forms and Validations

Siemens Web Framework supports **Dialogs** which can be easily configured in the application to define the required user interface of the command panels.

These **Panels** can internally use the required combinations of **property widgets** to create the user defined forms. There are a lot of **property and form level validations** that the framework provides.

On top of these framework provided validations, **custom validations** can be added. There are form level APIs available to perform programmatic update of property widgets and even reset/clear the form.

The following tutorials provide some simple and useful examples on various types of property widgets and validations which can be used to enrich the dialogs in your application:

- **Dialogs**
- **Property Widgets, Forms and Validations**
- **Dynamic Widgets and Custom Validations**

**Tip**

These examples assume that the **previous tutorials** have been followed already and a **module** called **trackerApp** is already available in your example application.

## Managing Dialogs

Here is an example to demonstrate the configuration of dialogs:

In your sample "trackerApp" module, open the **commandsViewModel.json** file at module level.

Look for the `Tkr0AddCmd` command added in **previous tutorials**. Check the command handler defined for that command.

```
        "addTkrObject": {
            "actionType": "dialog",
            "inputData": {
                "options": {
                    "view": "tkr0AddIssue",
                    "caption": "{{i18n.addIssue}}",
                    "width": "STANDARD",
                    "height": "FULL",
                    "parent": ".aw-layout-workareaMain",
                    "placement": "right"
                }
            }
        }
```

Let's understand what this means.

### Dialog Action

**actionType** - `dialog` - This is indicating the system that a **Dialog** needs to be launched with certain **DialogOptions**.

### Dialog Options

### View

The **view** indicates the name of the component that has to be shown inside the Dialog. Remember, Dialog is just a container which provides certain behavior like Dock/Undock/Reset for free. The view which needs to be rendered inside is configurable via this dialogOption called `view`.

For our example, we would need to create a view and viewModel with name `tkr0AddIssue`. We can create them as empty for now:


Siemens Web Framework
Unpublished work. © 2025 Siemens

tkr0AddIssueView.html

```
<div> </div>
```

tkr0AddIssueViewModel.json

```
{
    "schemaVersion": "1.0.0"
}
```

## Caption

The **caption** of the dialog indicates the title of the panel that you would like to show. It can be localized as shown below.

```
 "caption": "{{i18n.addIssue}}"
```

where standard i18n definition can be specified in the commandsViewModel.json

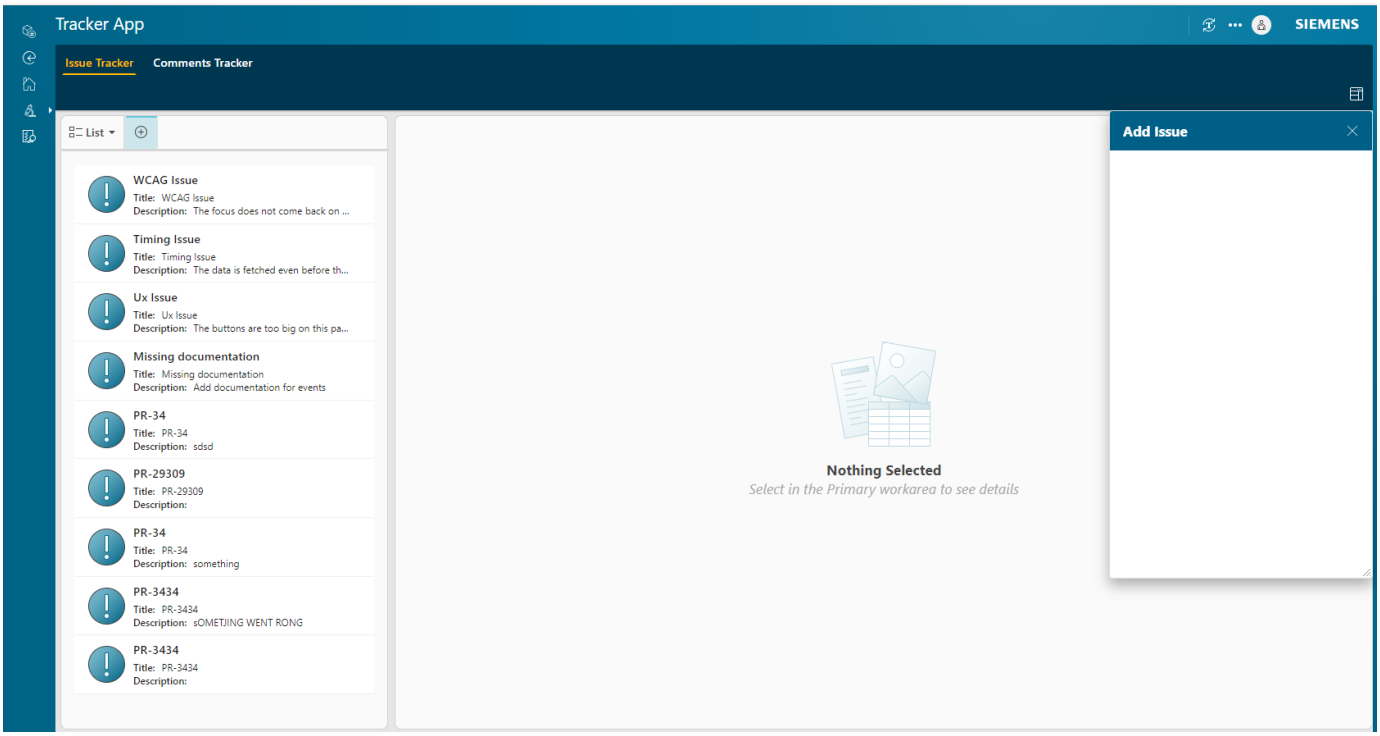
```
"i18n": {
    ...
    "addIssue": [ "TrackerAppMessages" ]
}
```

And define the i18n key in the message file TrackerAppMessages as:

```
 "addIssue": "Add Issue"
```

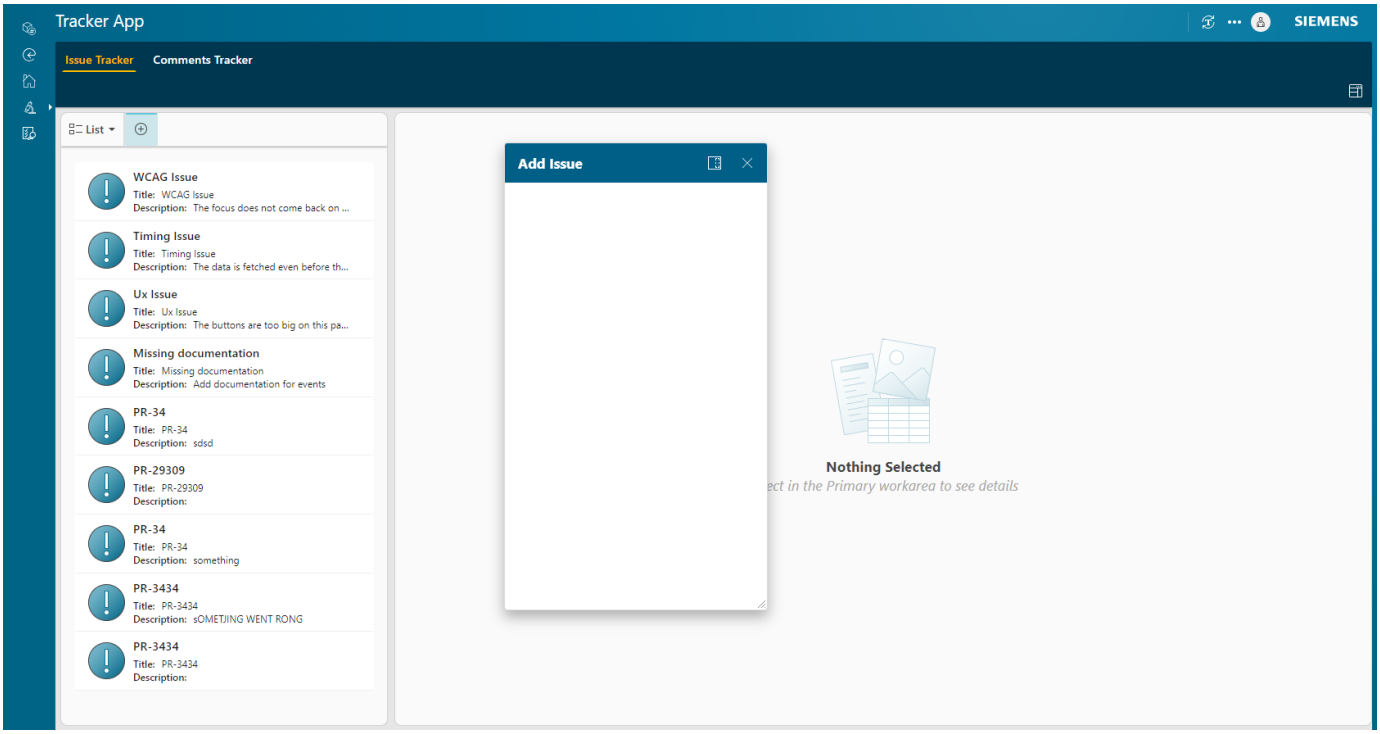
The other dialogOptions are self explanatory which provide the width, height and decide where should the dialog be placed.

With this Dialog configuration, here is what your compiled application should show on clicking the `AddIssue` command added in previous tutorials:




## Close/Reset Commands

Notice the styling and **Close** command coming for free from Dialog. Try to drag it around, and you will see its position will be persisted after drag. And a **Reset** command will show up for you to reset it.



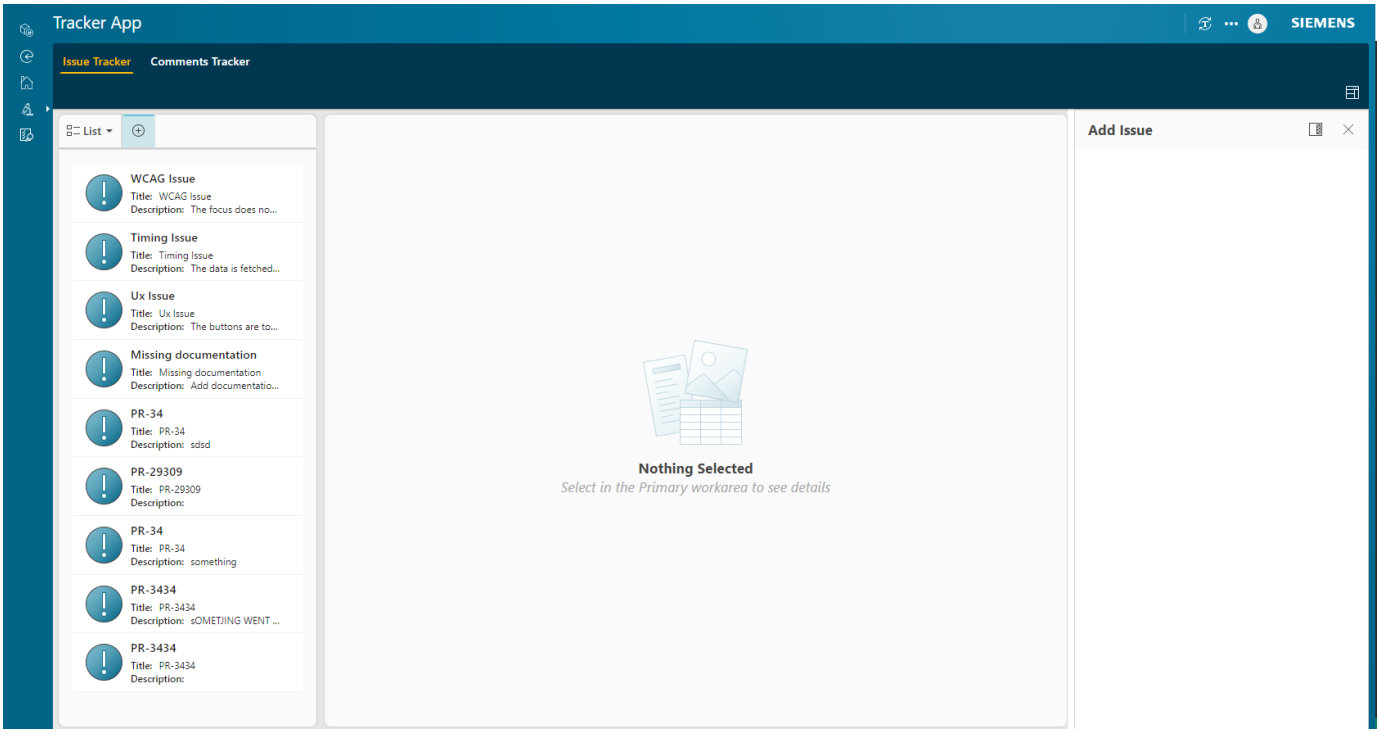


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Push Dialog - Dock/Undock Commands

Now, let's add a DialogOption called `push` and specify the value as `true`.

The Dialog is now changed to `push` type which means now it can be pushed in and made to look as part of the workareas. It comes with **Dock/Undock** capabilities.




The dialogs can be launched/invoked via using the actionType as **DialogAction**. For closing the dialog, either the **Close** command on the Dialog could be used or there is a **published API** called `closeDialog` which can be used either declaratively or programmatically.

In the **next tutorial**, we will be using some property widgets to develop a form and learn about property and form validations provided by Siemens Web Framework.

## Property Widgets, Forms and Validations

Siemens Web Framework offers a variety of property widgets that can be used to design the required UI. There are some in-built property validations that we will see in the current tutorial.

Let us use the tkr0AddIssueView.html that we created in **last tutorial** and add some meaningful content to it.

Continuing with our example, we need to create a form which lets the user create an Issue which can then be seen in our `trackerApp`.

## Property Widgets

Let's add the following property widgets to the View:

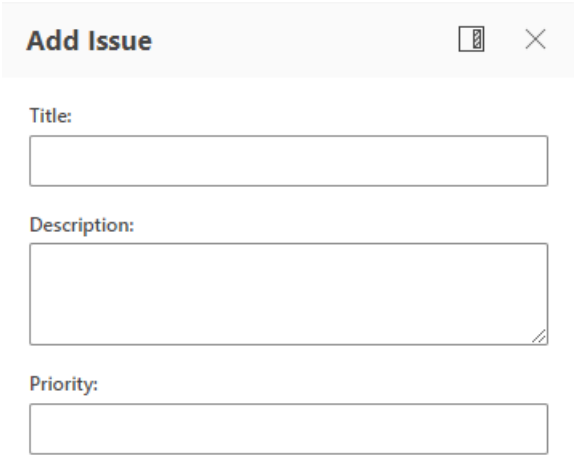
```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-numeric prop="data.priority"></aw-numeric>
    </aw-panel-body>
</aw-panel>
```

And define the corresponding properties in ViewModel.json

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": ""
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": ""
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": ""
        }
    }
}
```

After building and running the code, the dialog should look like:


Siemens Web Framework
Unpublished work. © 2025 Siemens




Here, we have used the property widgets from SWF like **aw-textbox**, **aw-textarea** and **aw-numeric** and we have created the **ViewModelProperties** specifying the required Datatypes.

## Property Validations

Let us now, add some property validations by adding them to the viewModelProperty definitions:

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
        }
    }
}
```

The type based property validation will kick in automatically. Like, the priority field, since it is defined with type 'Integer', it will accept only integers as inputs.

After building and running the code, the dialog should look like:

Here, the `isRequired` flag has ensured that the title property is marked as Required. The specified `dbValue` on priority has ensured that the widget shows up with default value as populated.

## Drop down list

Let's now add a dropdown to this form.

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-numeric prop="data.priority"></aw-numeric>
    </aw-panel-body>
</aw-panel>
```

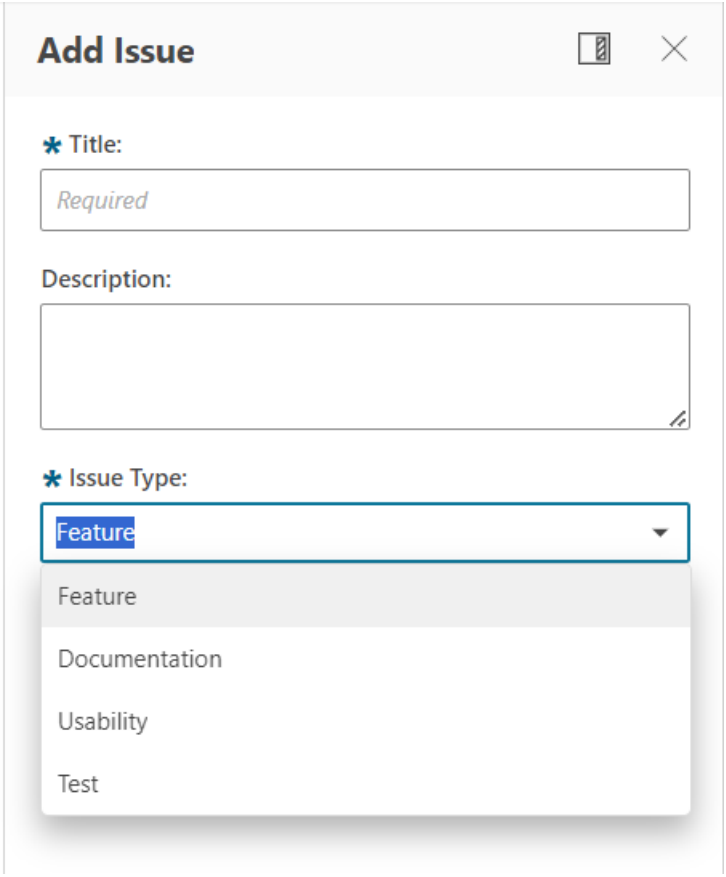
```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
```



Unpublished work. © 2025 Siemens

```
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        }
    }
}
```

After building and running the code, the dialog should look like:




Here, we have added a simple drop down with some values coming from JSON configuration. These values can be fetched from any source like JS, REST, SOA etc using the standard **dataProviders**. Associating the dataProvider to the widget `type` has instructed the system to render it is a list of values or a drop down list. The specified value in the dbValue is the default value with which the dropdown list gets rendered. Please also note that this is marked as Required via `isRequired` property set on the VMP.

## Universal Property Widget - Listbox example

The property widget used for rendering this dropdown is a **listbox**. However, an universal widget called **AwWidget** can also be used for rendering any type of properties. This universal widget works with the help of a property called `hint` - this when specified instructs the system to render the corresponding widget. Here is an example -

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

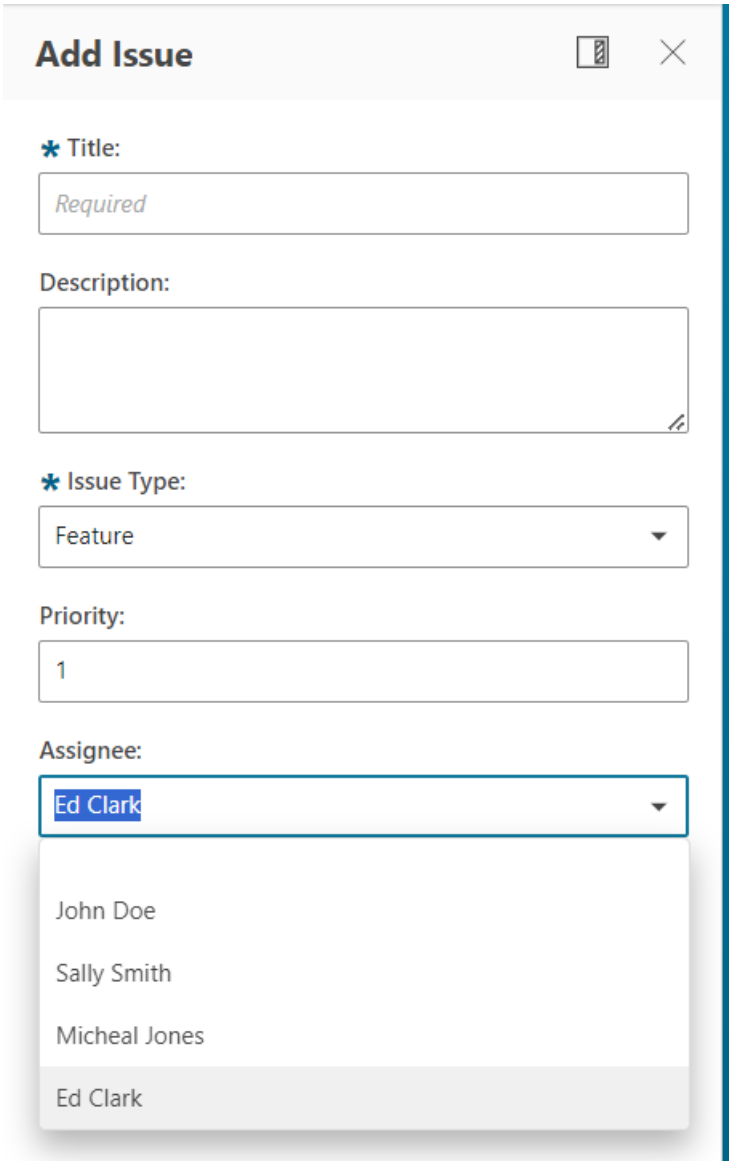
```
    </aw-panel-body>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
```

```
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    }
}
```

After building and running the code, the dialog should look like:


Siemens Web Framework
Unpublished work. © 2025 Siemens




Here, we have rendered a very similar dropdown but via using the universal widget AwWidget which when specified with dataProvider knows that a list needs to be rendered.

## Universal Property Widget - Checkbox example

Here is another example for this universal widget using the `hint` property to identify which type of widget to render.

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
```

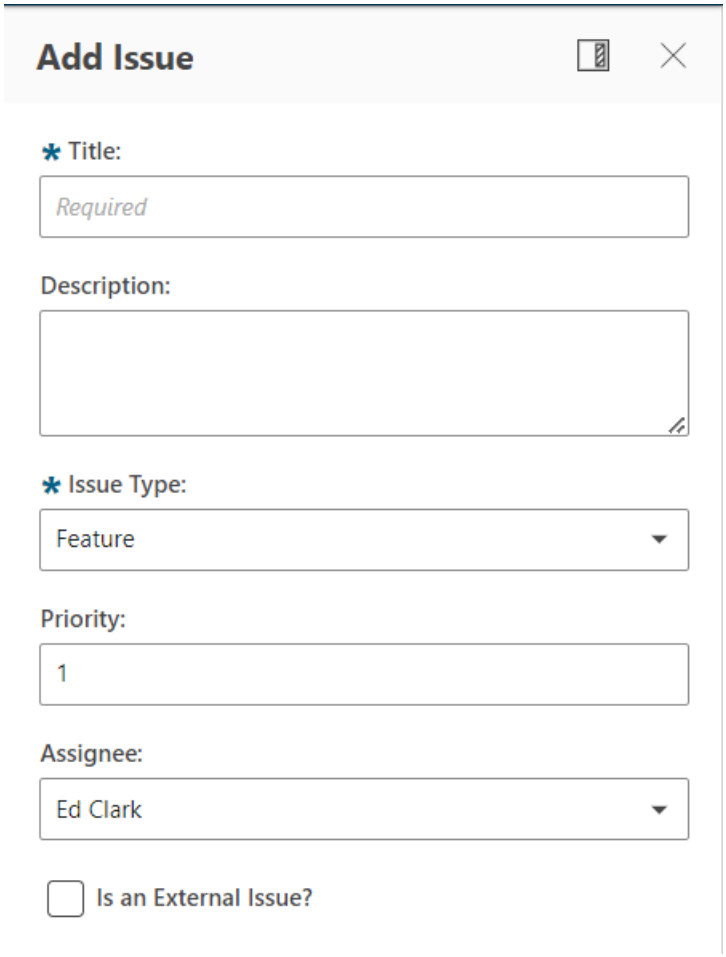
```
        <aw-widget prop="data.isExternal"></aw-widget>
    </aw-panel-body>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        },
        "isExternal": {
            "displayName": "Is an External Issue?",
            "type": "BOOLEAN",
            "isRequired": "false",
            "isEditable": "true",
            "hint": "checkbox"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    }
}
```

After building and running the code, the dialog should look like:




Here, a checkbox has been rendered due to the `hint` property specified on the definition of `isExternal`.

## Form Attributes and Validations - Valid State on Form

Let's now add the footer to this panel with some buttons - add.

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"></aw-widget>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add">{{i18n.add}}</aw-button>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    </aw-panel-footer>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        },
        "isExternal": {
            "displayName": "Is an External Issue?",
            "type": "BOOLEAN",
            "isRequired": "false",
            "isEditable": "true",
            "hint": "checkbox"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
```

```
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    },
    "actions": {
        "add": {
            "actionType": "batchJob",
            "steps": [ {
                    "action": "addData"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                },
                {
                    "action": "closeDialog"
                }
            ]
        },
        "addData": {
            "actionType": "JSFunction",
            "method": "addIssueData",
            "deps": "js/issueHelperService",
            "inputData": {
                "title": "{{data.title.dbValue}}",
                "description": "{{data.description.dbValue}}",
                "type": "{{data.types.dbValue}}",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "assignee": "{{data.assignee.dbValue}}"
            }
        },
        "reloadPrimaryWorkarea": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "primaryWorkarea.reset"
                } ]
            }
        },
        "closeDialog": {
            "actionType": "JSFunction",
            "method": "closeDialog",
            "deps": "js/dialogService",
            "inputData": {
                "editContext": null,
                "popupId": "{{subPanelContext.popupOptions.popupId}}"
            }
        }
    },
    "i18n": {
        "add": [ "TrackerAppMessages" ]
    }
}
```

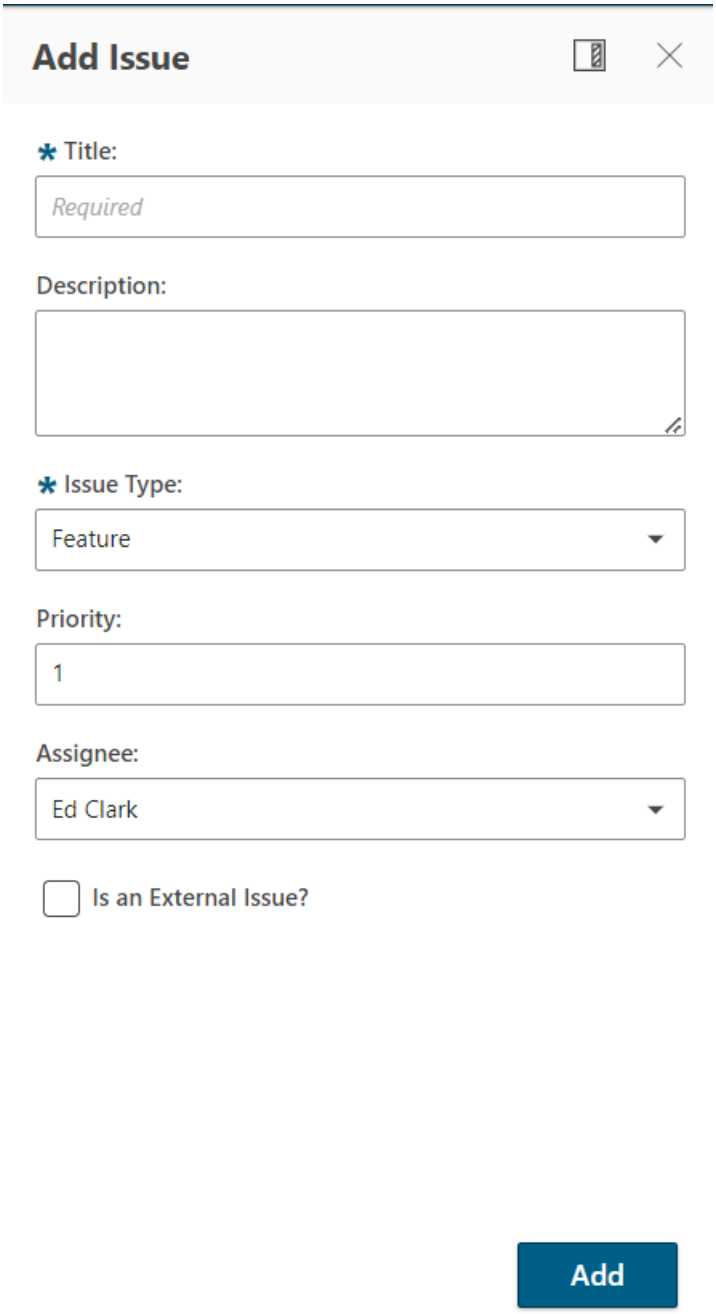
The i18n key needs to be defined in TrackerAppMessages:

```
{
    ...
    "add": "Add
}
```

The `add` action was added in the previous tutorial, if not - here is the code in the issueHelperService.js

```
export const addIssueData = async ( title, description, type, assignee ) =>
{
    let result = await getDB().addToStore( 'issue', {
        IssueID: 'PR-' + Math.floor( Math.random() * 1000000 ),
        Title: title,
        Description: description,
        TypeIcon: 'assets/images/typeIssue48.svg',
        Type: type,
        Assignee: assignee,
        ExpectedStoryPoints: Math.floor( Math.random() * 10 ),
        ActualStoryPoints: Math.floor( Math.random() * 10 )
    } );
    return Promise.resolve( { data: { result: result } } );
};
```

After building and running the code, the dialog should look like:




Here, we have added a simple `Add` button with some declarative actions to perform the actual add operation and close the dialog using the `closeDialog` API. What is of interest is the visibility of this button.

Since, we have some required properties which are empty on the form; you would expect this button to be rendered as disabled first and it should get enabled only when all the required property validations have been passed i.e. the complete form is valid.

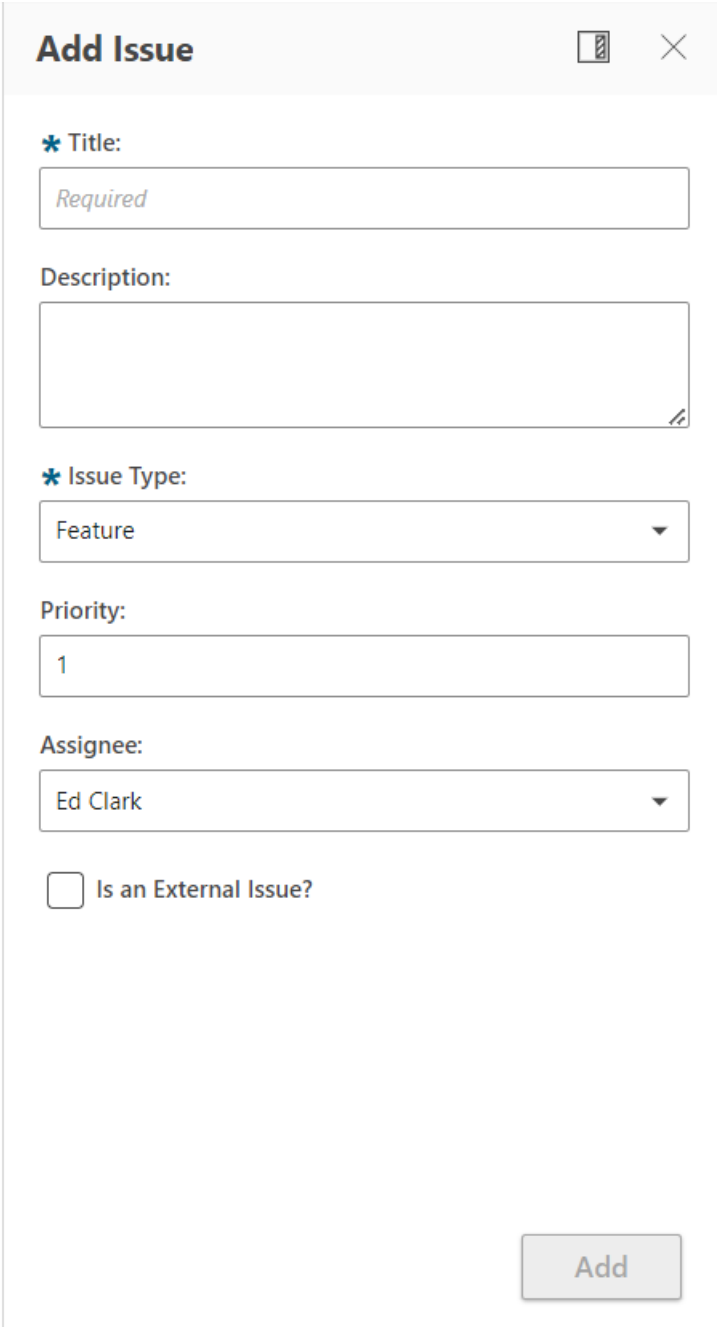

Unpublished work. © 2025 Siemens

The same can be achieved by adding a simple enable-when clause to this button and using the **formProp.attributes.valid** property available on the component. Here's how:

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"></aw-widget>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">{{i18n.add}}</aw-button>
    </aw-panel-footer>
</aw-panel>
```

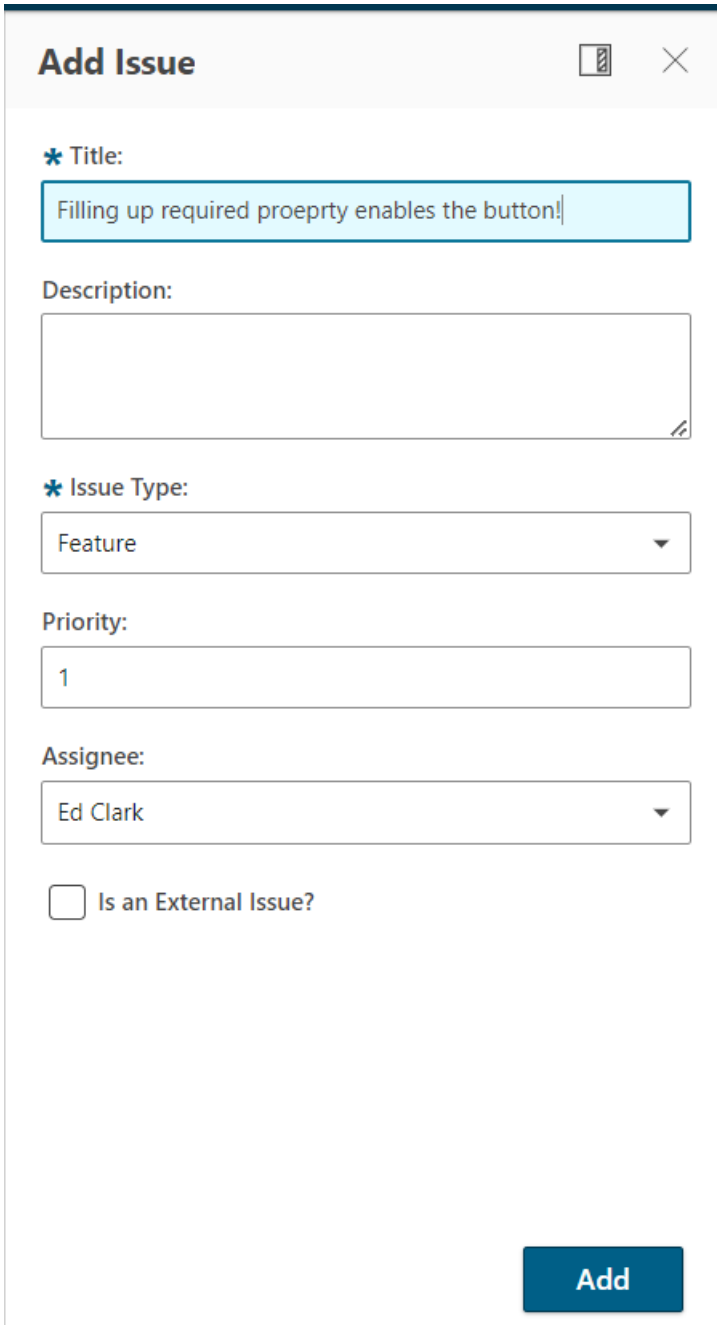
After building and running the code, the dialog should look like:


Unpublished work. © 2025 Siemens




Here, Add is disabled because the form is not valid. One of the required properties has not been filled in; once it is specified, here is how form would look like:


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Form Attributes and Validations - Dirty State on Form - Clear API

Let's now take a look at the use case where you need to clear the form. The requirement would be to clear all the filled in fields and this button to clear should be visible only when the form has been modified by the user i.e. marked dirty. The same can be achieved by defining a clear button which will be enabled only

when form has been made dirty i.e. formProp.attributes.dirty has been set. And on its click, we can simply re-use the form API available on this formProp object called clear(). Here's how:

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"></aw-widget>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">{{i18n.add}}</aw-button>
        <aw-button action="clear" enable-
when="formProp.attributes.dirty">{{i18n.clear}}</aw-button>
    </aw-panel-footer>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        },
        "isExternal": {
            "displayName": "Is an External Issue?",
            "type": "BOOLEAN",
            "isRequired": "false",
            "isEditable": "true",
            "hint": "checkbox"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    },
    "actions": {
        "add": {
            "actionType": "batchJob",
            "steps": [ {
```

```
                    "action": "addData"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                },
                {
                    "action": "closeDialog"
                }
            ]
        },
        "addData": {
            "actionType": "JSFunction",
            "method": "addIssueData",
            "deps": "js/issueHelperService",
            "inputData": {
                "title": "{{data.title.dbValue}}",
                "description": "{{data.description.dbValue}}",
                "type": "{{data.types.dbValue}}",
                "assignee": "{{data.assignee.dbValue}}"
            }
        },
        "reloadPrimaryWorkarea": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "primaryWorkarea.reset"
                } ]
            }
        },
        "closeDialog": {
            "actionType": "JSFunction",
            "method": "closeDialog",
            "deps": "js/dialogService",
            "inputData": {
                "editContext": null,
                "popupId": "{{subPanelContext.popupOptions.popupId}}"
            }
        },
        "clear": {
            "actionType": "JSFunction",
            "method": "clearData",
            "inputData": {
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        }
    },
    "i18n": {
        "add": [ "TrackerAppMessages" ],
        "clear": [ "TrackerAppMessages" ]
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

TrackerAppMessages.json:

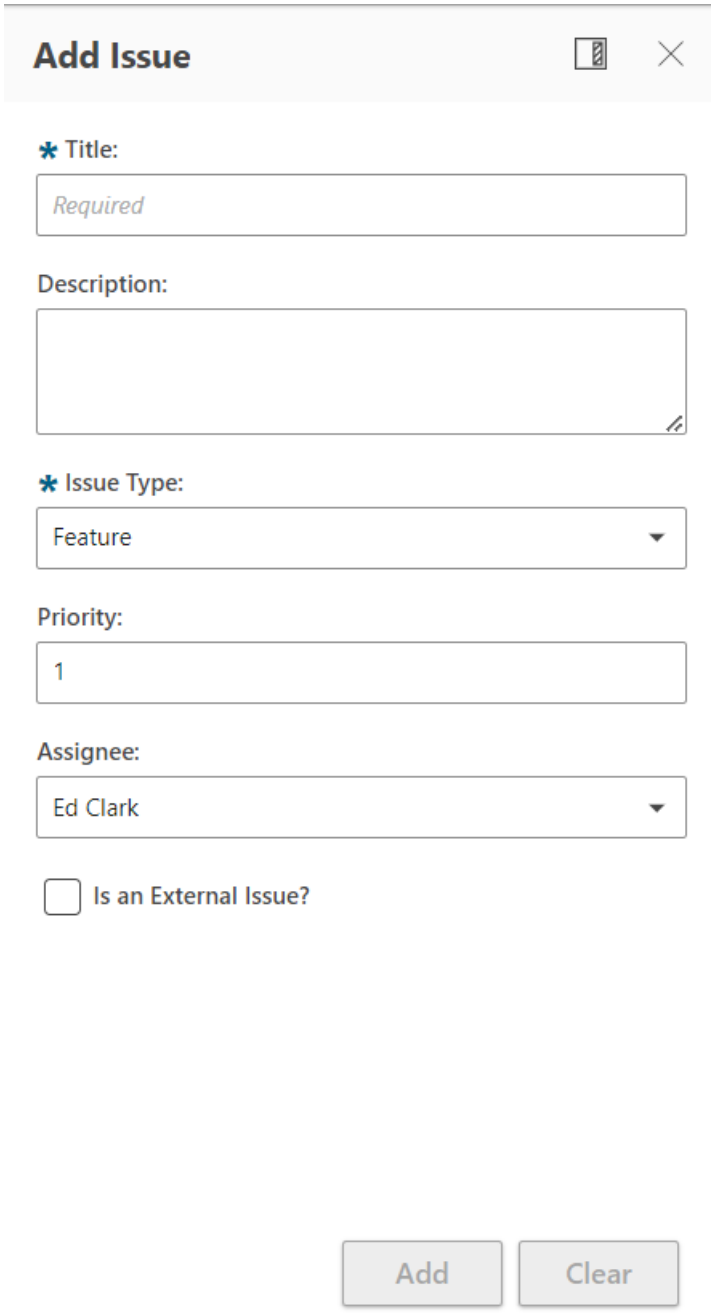
```
{
    ...
    "add": "Add",
    "clear": "Clear"
}
```

issueHelperService.js

```
export const clearData = function( formProp ) {
    return formProp.clear();
};
```

After building and running the code, the dialog should look like:

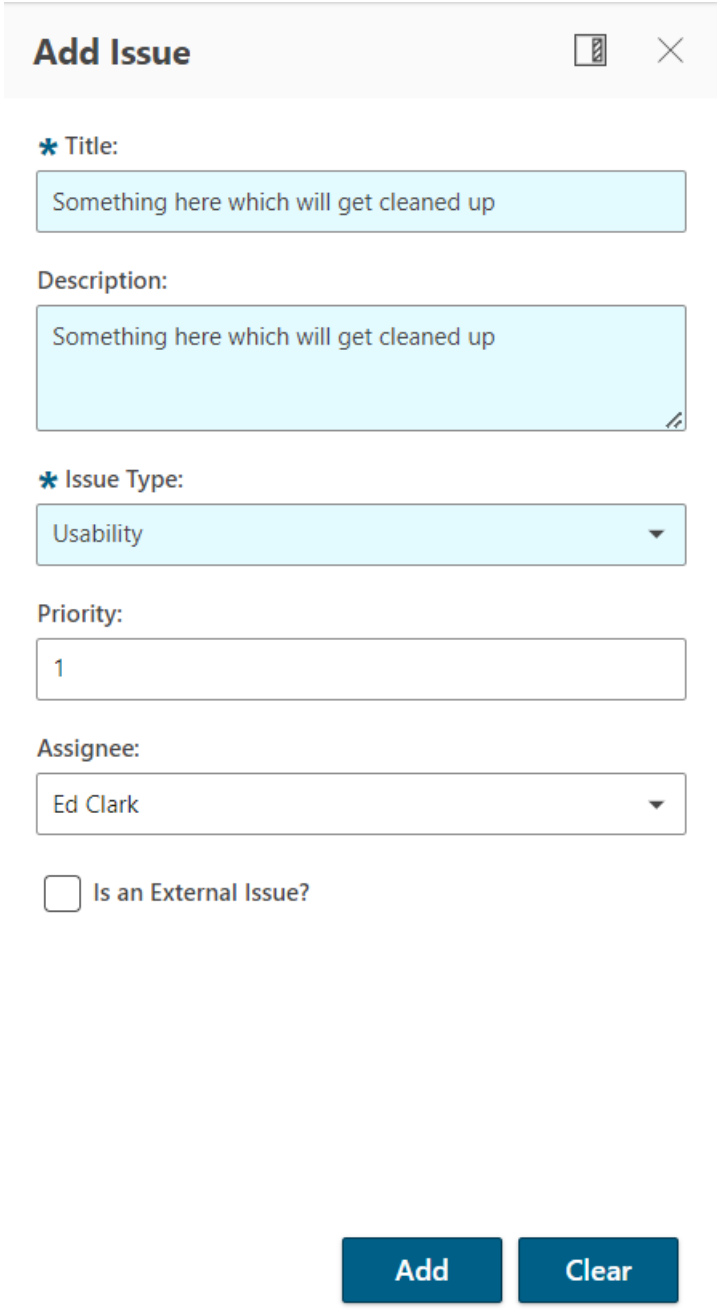

Unpublished work. © 2025 Siemens



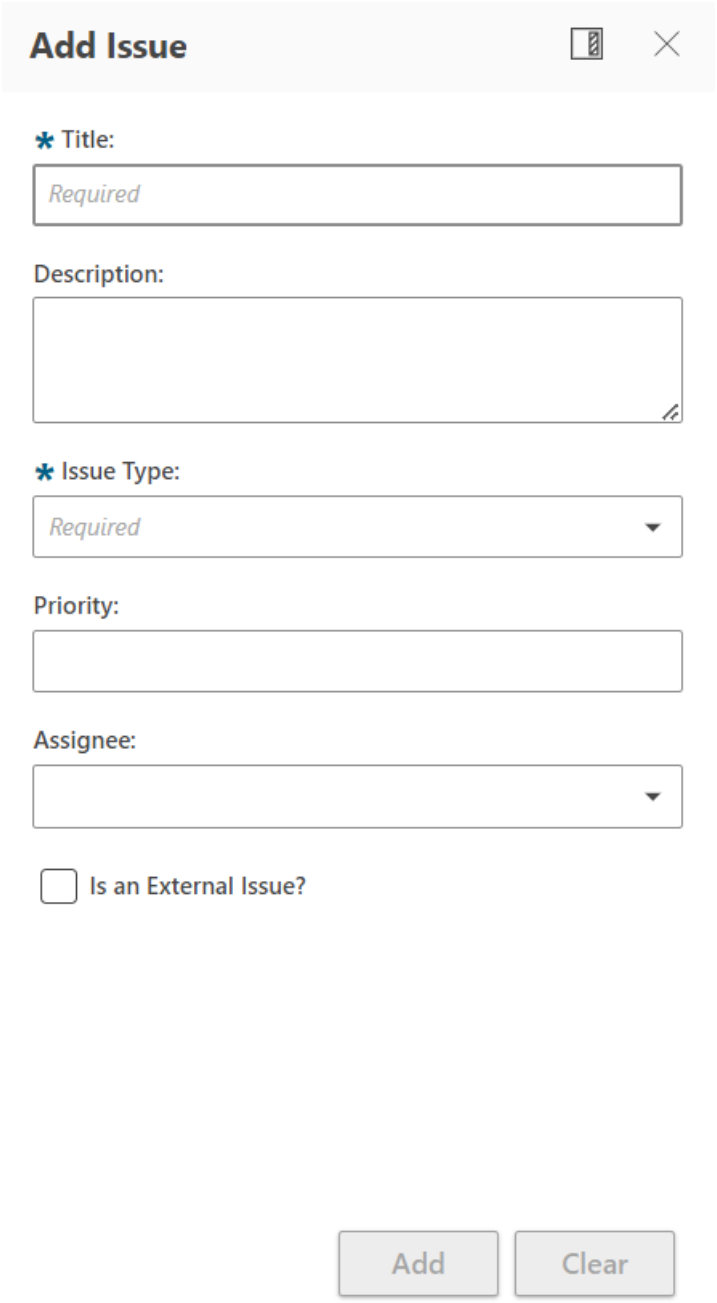

Look at the disabled clear button due to formProp.attributes.dirty not being set i.e. the form is pristine and not dirty.

Let's dirty the form by making some edits.


Siemens Web Framework
Unpublished work. © 2025 Siemens




The clear button is enabled now. And on clicking, it is going to invoke the **formProp.clear()** API and clear all the fields on the form like:




## Form Attributes and Validations - Dirty State on Form - Reset API

Now, here you see all the properties have been cleared out; in case you want to reset the form to its default values; you can instead use the Reset capabilities. Here is an example:

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"></aw-widget>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">{{i18n.add}}</aw-button>
        <aw-button action="clear" enable-
when="formProp.attributes.dirty">{{i18n.clear}}</aw-button>
        <aw-button action="reset" enable-
when="formProp.attributes.dirty">{{i18n.reset}}</aw-button>
    </aw-panel-footer>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        },
        "isExternal": {
            "displayName": "Is an External Issue?",
            "type": "BOOLEAN",
```

```
            "isRequired": "false",
            "isEditable": "true",
            "hint": "checkbox"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    },
    "actions": {
        "add": {
            "actionType": "batchJob",
            "steps": [ {
                    "action": "addData"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                },
                {
                    "action": "closeDialog"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                }
            ]
        },
        "addData": {
            "actionType": "JSFunction",
            "method": "addIssueData",
            "deps": "js/issueHelperService",
            "inputData": {
                "title": "{{data.title.dbValue}}",
                "description": "{{data.description.dbValue}}",
                "type": "{{data.types.dbValue}}",
                "assignee": "{{data.assignee.dbValue}}"
            }
        },
        "reloadPrimaryWorkarea": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "primaryWorkarea.reset"
                } ]
            }
        },
        "closeDialog": {
            "actionType": "JSFunction",
            "method": "closeDialog",
            "deps": "js/dialogService",
            "inputData": {
                "editContext": null,
                "popupId": "{{subPanelContext.popupOptions.popupId}}"
            }
        },
        "clear": {
            "actionType": "JSFunction",
            "method": "clearData",
            "inputData": {
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        },
        "reset": {
            "actionType": "JSFunction",
            "method": "resetData",
            "inputData": {
                "data": "{{data}}",
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        }
    },
    "i18n": {
        "add": [ "TrackerAppMessages" ],
        "clear": [ "TrackerAppMessages" ],
        "reset": [ "TrackerAppMessages" ]
```

```
    }
}
```

TrackerAppMessages.json:

```
{
    ...
    "add": "Add",
    "clear": "Clear",
    "reset": "Reset",
}
```

issueHelperService.js

```
export const resetData = function( data, formProp ) {
    formProp.clear();
    let priority = { ...data.priority };
    priority.dbValue = priority.uiValue = '1';

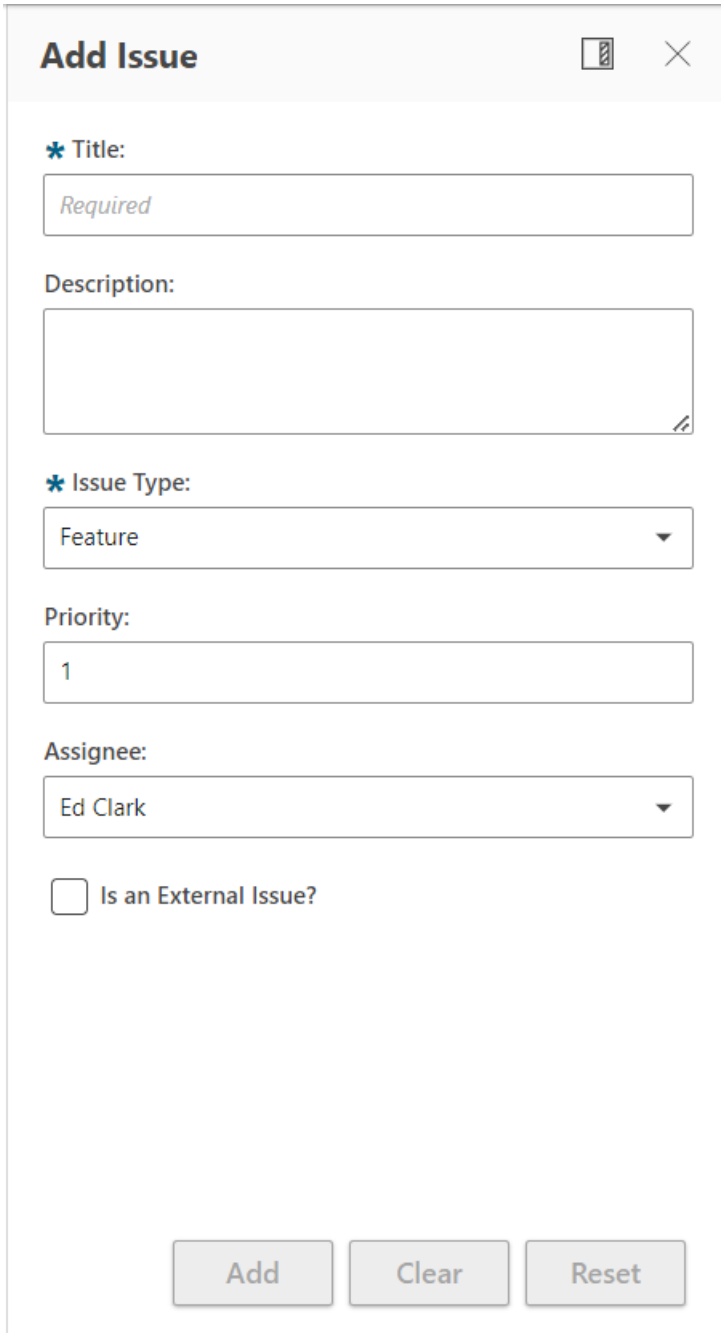
    let types = { ...data.types };
    types.dbValue = types.uiValue = 'Feature';

    let assignee = { ...data.assignee };
    assignee.dbValue = assignee.uiValue = 'Ed Clark';

    formProp.reset( {
        priority,
        types,
        assignee
    } );
};
```

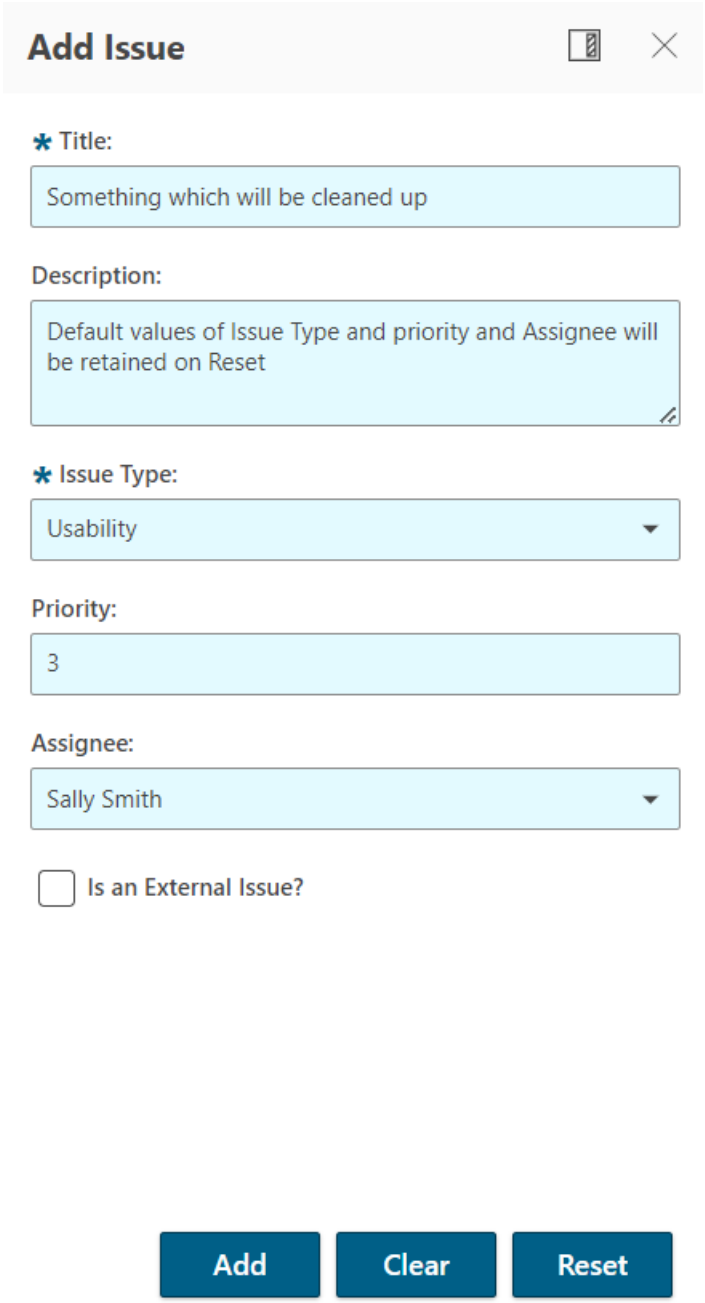
After building and running the code, the dialog should look like:


Siemens Web Framework
Unpublished work. © 2025 Siemens



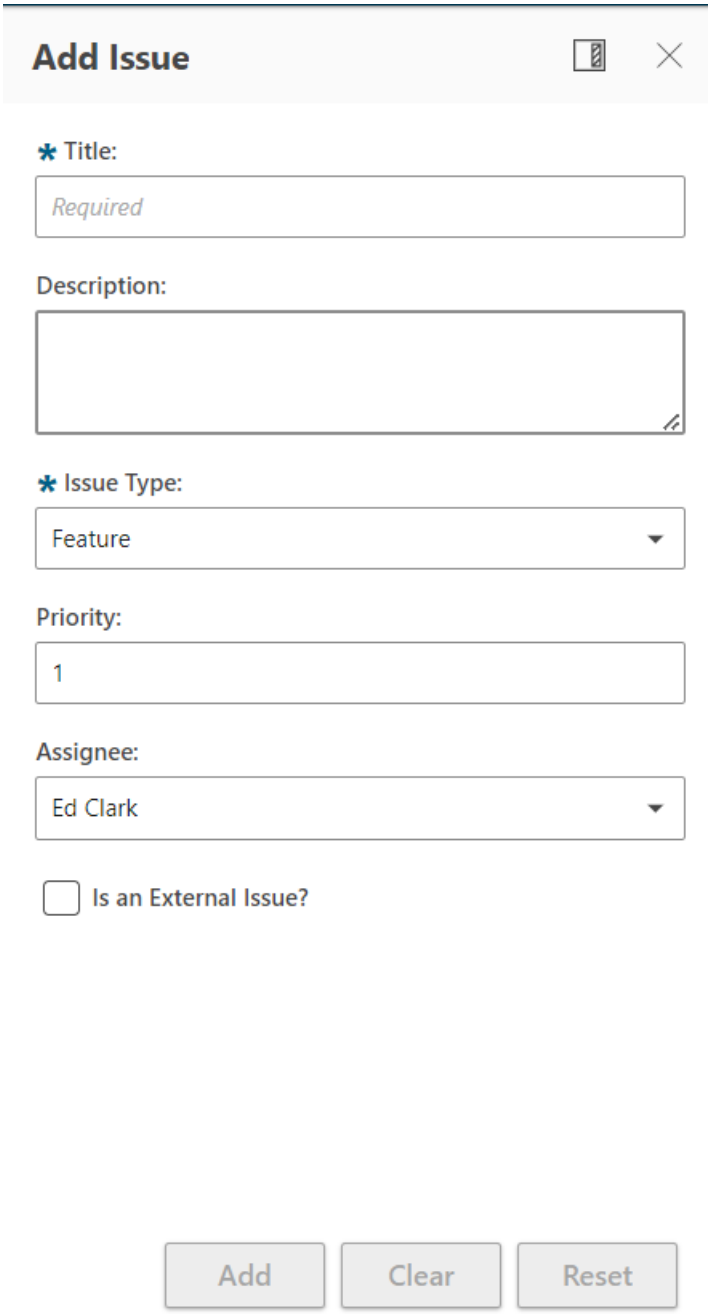

The default state of the form where Reset is rendered as disabled because the form is not dirty. Look at the enable-when specified on the reset button.

Once, the user has made some edits on the form, the reset button gets enabled



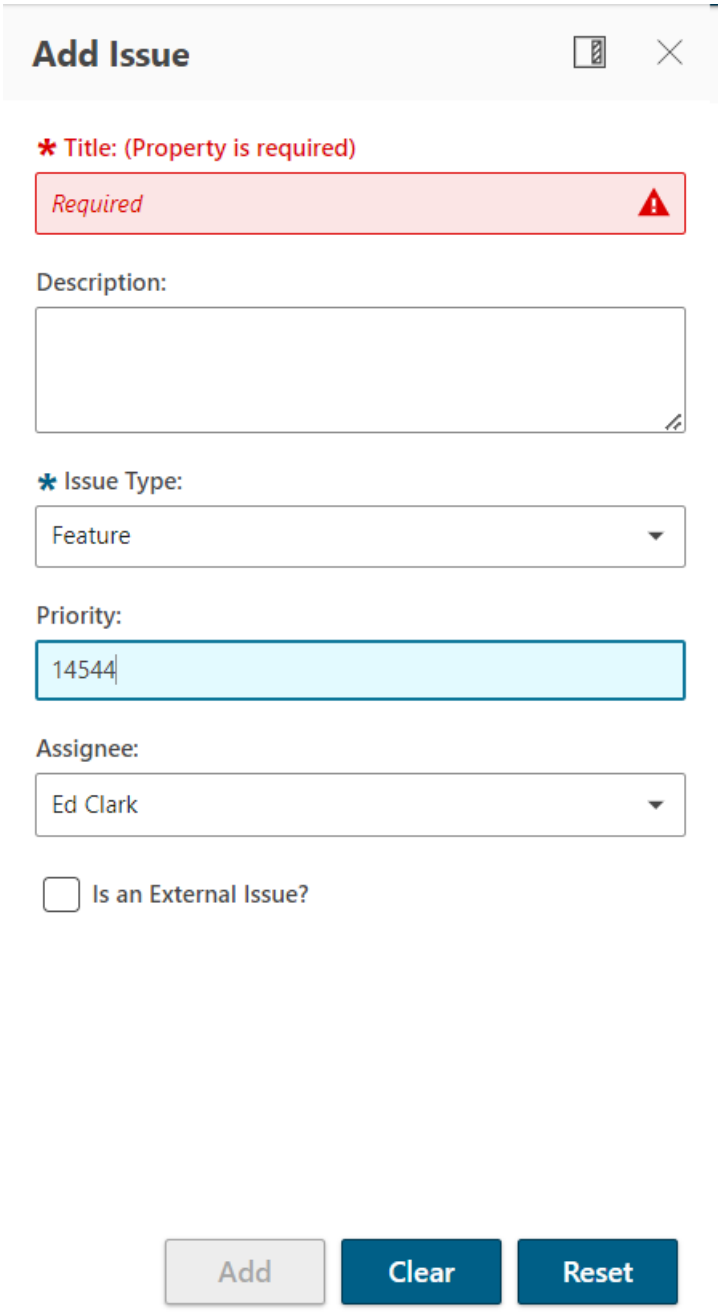

On clicking the Reset, note that the properties which had the default values, they have been retained back on the form as:


Siemens Web Framework
Unpublished work. © 2025 Siemens




The **formProp.reset()** API does the reset of the form and sets it to its original state but the default values that need to be reset, needs to be provided by the user. The framework does not and cannot have knowledge of these default values and hence it is the consumer's responsibility to use reset API with the original or the default values you need to the form to be reset to.

Also, notice the in-built property validation in this form. The properties which are marked as `isRequired` would need a value before the form can be considered as valid. And if the value is not specified (after the form has been marked dirty), the required validations would show up to indicate the same to the user.




This was about the property widgets, universal property widget, basic property level and form level validations, understanding the form state (valid/dirty) and form APIs (clear and reset) to help you achieve the required form related operations and validations.

Next, we will take a look at some **custom validations and dynamic widgets**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Dynamic Widgets and Custom Validations

In the **previous tutorial**, we saw the basic property and form validation available directly from Siemens Web Framework. Now, if there is a need to add custom validations, that can also be achieved via **Validation Criteria**.

## Custom Validations - Validation Criteria

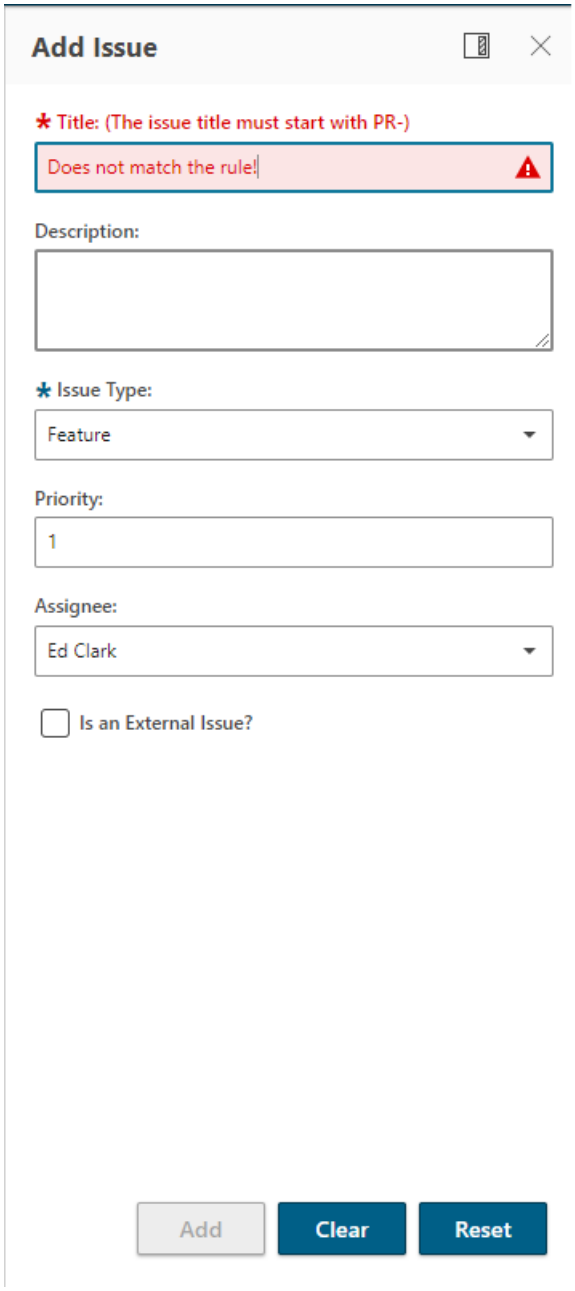
As an example, let's say the AddIssue form that we built, it needs a naming rule on the title i.e. the issues should start with `PR-`. And you need to form to ensure that only the values meeting this criteria are allowed into the form. Here's how to achieve that:

```
{
    ...
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true",
            "validationCriteria": [ {
                "condition": "conditions.isInvalidIssueTitle",
                "value": "{{i18n.issueNamingRuleError}}"
            } ]
        },
        ...
    },
    "conditions": {
        "isInvalidIssueTitle": {
            "expression": "data.title.dbValue.indexOf('PR-') !== 0"
        }
    },
     "i18n": {
        "add": [ "TrackerAppMessages" ],
        "clear": [ "TrackerAppMessages" ],
        "reset": [ "TrackerAppMessages" ],
        "issueNamingRuleError": [ "TrackerAppMessages" ]
    }
    ...
}
```

The i18n goes to the TrackerAppMessages.json:

```
{
    ...
    "add": "Add",
    "clear": "Clear",
    "reset": "Reset",
    "issueNamingRuleError": "The issue title must start with PR-",
}
```

After building and running the code, the dialog should look like:




Here, on the `title` property, we have specified a custom validation using `validationCriteria`. This criteria takes a **declarative condition** which defines the criteria to be met by the validation. And the localized error it should show if the criteria is not met is defined by the `value` property in the `validationCriteria`.

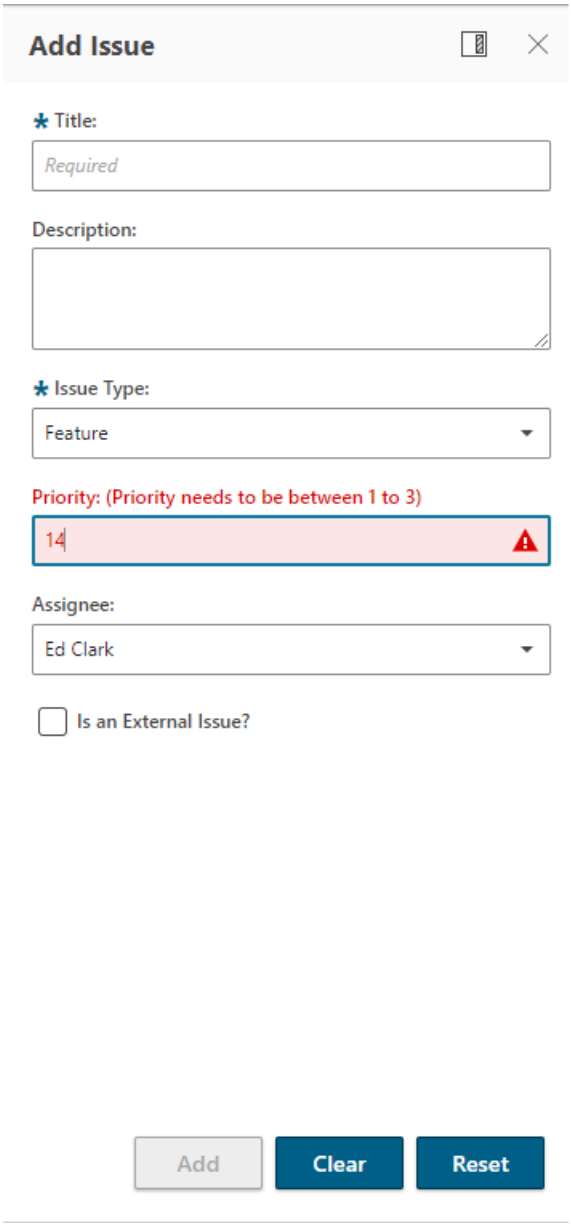

Siemens Web Framework
Unpublished work. © 2025 Siemens

## Custom Validations - Validation Criteria - another example

Let's add one more custom validation - say the `priority` has to be between 1 to 3. It can be added simply by defining this same criteria on `priority` property as:

```
{
    ...
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true",
            "validationCriteria": [ {
                "condition": "conditions.isInvalidIssueTitle",
                "value": "{{i18n.issueNamingRuleError}}"
            } ]
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "dbValue": "1",
            "validationCriteria": [ {
                "condition": "conditions.isInvalidPriorityRange",
                "value": "{{i18n.priorityRangeError}}"
            } ]
        },
        ...
    },
    "conditions": {
        "isInvalidIssueTitle": {
            "expression": "data.title.dbValue.indexOf('PR-') !== 0"
        },
        "isInvalidPriorityRange": {
            "expression": "data.priority.dbValue < 1 ||
data.priority.dbValue > 3"
        }
    },
     "i18n": {
        "add": [ "TrackerAppMessages" ],
        "clear": [ "TrackerAppMessages" ],
        "reset": [ "TrackerAppMessages" ],
        "issueNamingRuleError": [ "TrackerAppMessages" ],
        "priorityRangeError": [ "TrackerAppMessages" ]
    }
    ...
}
```

After building and running the code, the dialog should look like:




Here, the custom property validation of priority being in between 1 and 3 has been attached to the `priority` property. Any other value specified by the user is going to be errored out.

## Interdependent Widgets

Next, let us take a look at interdependent widgets. In this example, we are going to take up the requirement that based on the `type` of the issue selected in the dropdown that we added earlier, a new property called `teamName` should be automatically populated. Essentially, the value of team needs to be decided or driven by the type of the issue. This is an example of populating the value in one widget depending upon that of another widget.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Here's how it would look like:

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-textbox prop="data.team"></aw-textbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"></aw-widget>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">{{i18n.add}}</aw-button>
        <aw-button action="clear" enable-
when="formProp.attributes.dirty">{{i18n.clear}}</aw-button>
        <aw-button action="reset" enable-
when="formProp.attributes.dirty">{{i18n.reset}}</aw-button>
    </aw-panel-footer>
</aw-panel>
```

```
{
    "schemaVersion": "1.0.0",
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true",
            "validationCriteria": [ {
                "condition": "conditions.isInvalidIssueTitle",
                "value": "{{i18n.issueNamingRuleError}}"
            } ]
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
```

```
            "dbValue": "1",
            "validationCriteria": [ {
                "condition": "conditions.isInvalidPriorityRange",
                "value": "{{i18n.priorityRangeError}}"
            } ]
        },
        "isExternal": {
            "displayName": "Is an External Issue?",
            "type": "BOOLEAN",
            "isRequired": "false",
            "isEditable": "true",
            "hint": "checkbox"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        },
        "team": {
            "displayName": "Team Name",
            "type": "STRING",
            "dbValue": "Development Team",
            "isRequired": "true",
            "isEnabled": "false"
        }
    },
    "actions": {
        "add": {
            "actionType": "batchJob",
            "steps": [ {
                    "action": "addData"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                },
                {
                    "action": "closeDialog"
                }
            ]
        },
        "addData": {
            "actionType": "JSFunction",
            "method": "addIssueData",
            "deps": "js/issueHelperService",
            "inputData": {
                "title": "{{data.title.dbValue}}",
                "description": "{{data.description.dbValue}}",
                "type": "{{data.types.dbValue}}",
                "assignee": "{{data.assignee.dbValue}}"
            }
        },
        "reloadPrimaryWorkarea": {
            "actionType": "Event",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "primaryWorkarea.reset"
                } ]
            }
        },
        "closeDialog": {
            "actionType": "JSFunction",
            "method": "closeDialog",
            "deps": "js/dialogService",
            "inputData": {
                "editContext": null,
                "popupId": "{{subPanelContext.popupOptions.popupId}}"
            }
        },
        "clear": {
            "actionType": "JSFunction",
            "method": "clearData",
            "inputData": {
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        },
        "reset": {
            "actionType": "JSFunction",
            "method": "resetData",
            "inputData": {
                "data": "{{data}}",
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        },
        "selectAppropriateTeam": {
            "actionType": "JSFunction",
            "method": "selectAppropriateTeam",
            "inputData": {
                "selectedType": "{{data.types.dbValue}}"
            },
            "outputData": {
                "team.dbValue": "",
                "team.uiValue": ""
            },
            "deps": "js/issueHelperService"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
```

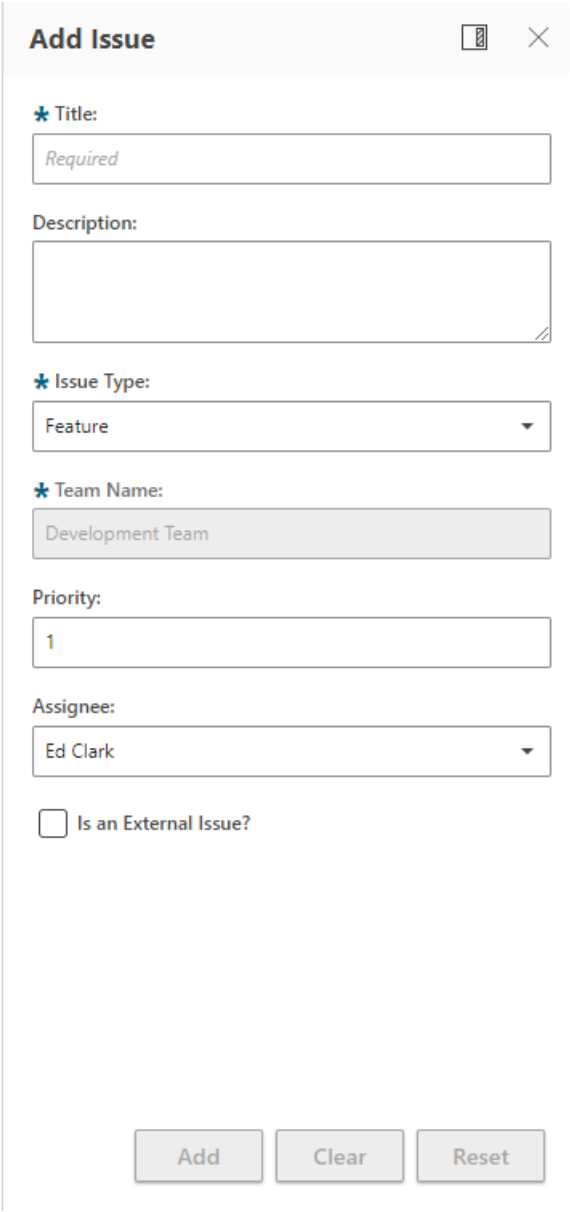
```
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    },
    "i18n": {
        "add": [ "TrackerAppMessages" ],
        "clear": [ "TrackerAppMessages" ],
        "reset": [ "TrackerAppMessages" ],
        "issueNamingRuleError": [ "TrackerAppMessages" ],
        "priorityRangeError": [ "TrackerAppMessages" ]
    },
    "conditions": {
        "isInvalidIssueTitle": {
            "expression": "data.title.dbValue.indexOf('PR-') !== 0"
        },
        "isInvalidPriorityRange": {
            "expression": "data.priority.dbValue < 1 ||
data.priority.dbValue > 3"
        }
    },
    "lifecycleHooks": {
        "onUpdate": [ {
            "action": "selectAppropriateTeam",
            "observers": [ "viewModel.types.dbValue" ]
        } ]
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    }
}
```

```
export const selectAppropriateTeam = function( selectedType ) {
    switch ( selectedType ) {
        case 'Feature':
            return 'Development Team';
        case 'Documentation':
            return 'Documentation Team';
        case 'Usability':
            return 'Ux Team';
        case 'Test':
            return 'Quality Assurance Team';
        default:
            return 'Development Team';
    }
};
```

After building and running the code, the dialog should look like:




Look at the `teamName` - it is disabled because in the VMP definition, we have marked the property as

```
"isEnabled": "false"
```

Also, we have defined a **lifecycle hook** which helps in building interdependence i.e. whenever the dbValue of `types` property changes i.e. user is updating the Type of the issue, we are invoking an action called `selectAppropriateTeam` which has the business logic of selecting the appropriate team name based on the input selected in Types. This is interdependence.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Observe, the outputData of this action:

```
        "selectAppropriateTeam": {
            "actionType": "JSFunction",
            "method": "selectAppropriateTeam",
            "inputData": {
                "selectedType": "{{data.types.dbValue}}"
            },
            "outputData": {
                "team.dbValue": "",
                "team.uiValue": ""
            },
            "deps": "js/issueHelperService"
        }
```

It basically declaratively updates the dbValue and uiValue of our dependent widget i.e. `team`. This is how interdependence between widget values can be achieved.

## Visibility of Widgets

If we need the complete widget to be visible or not depending upon the value selected in another widget, that can also be easily achieved. Let us take an example of another property called `customerName` which needs to be visible only when the issue being created is external i.e. the value of checkbox `isExternal` is set. This can be achieved by adding a visible-when clause on this new widget like:

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-textbox prop="data.team"></aw-textbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"></aw-widget>
        <aw-textbox prop="data.customerName" visible-
when="conditions.isAnExternalIssue"></aw-textbox>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">{{i18n.add}}</aw-button>
        <aw-button action="clear" enable-
when="formProp.attributes.dirty">{{i18n.clear}}</aw-button>
        <aw-button action="reset" enable-
when="formProp.attributes.dirty">{{i18n.reset}}</aw-button>
    </aw-panel-footer>
</aw-panel>
```

```
{
{
    "schemaVersion": "1.0.0",
```

```
    "data": {
        "title": {
            "displayName": "Title",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true",
            "validationCriteria": [ {
                "condition": "conditions.isInvalidIssueTitle",
                "value": "{{i18n.issueNamingRuleError}}"
            } ]
        },
        "description": {
            "displayName": "Description",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "types": {
            "displayName": "Issue Type",
            "name": "types",
            "type": "STRING",
            "isRequired": true,
            "dataProvider": "dataProviderForIssueTypes",
            "dbValue": "Feature",
            "dispValue": "Feature"
        },
        "priority": {
            "displayName": "Priority",
            "type": "INTEGER",
            "isRequired": "false",
            "isEditable": "true",
            "dbValue": 1,
            "dispValue": 1,
            "validationCriteria": [ {
                "condition": "conditions.isInvalidPriorityRange",
                "value": "{{i18n.priorityRangeError}}"
            } ]
        },
        "isExternal": {
            "displayName": "Is an External Issue?",
            "type": "BOOLEAN",
            "isRequired": "false",
            "isEditable": "true",
            "hint": "checkbox"
        },
        "customerName": {
            "displayName": "Customer Name",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        },
        "team": {
            "displayName": "Team Name",
            "type": "STRING",
            "dbValue": "Development Team",
            "isRequired": "true",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

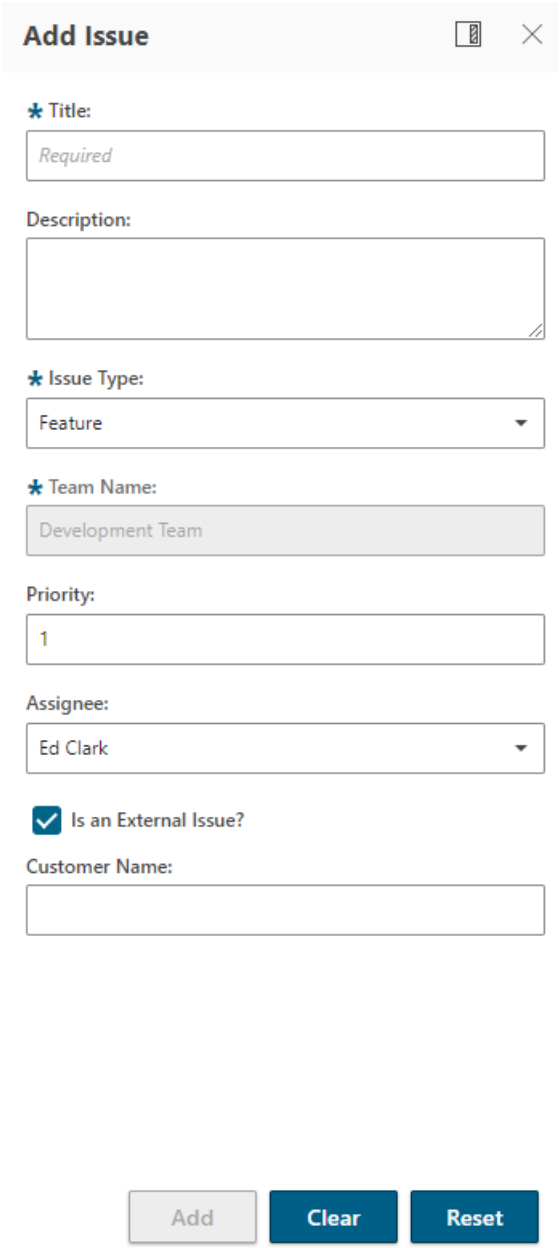
```
            "isEnabled": "false"
        },
        "assignee": {
            "displayName": "Assignee",
            "type": "STRING",
            "dbValue": "Ed Clark",
            "uiValue": "Ed Clark",
            "isRequired": "false",
            "dataProvider": "dataProviderForAssignees"
        }
    },
    "actions": {
        "add": {
            "actionType": "batchJob",
            "steps": [ {
                    "action": "addData"
                },
                {
                    "action": "reloadPrimaryWorkarea"
                },
                {
                    "action": "closeDialog"
                }
            ]
        },
        "addData": {
            "actionType": "JSFunction",
            "method": "addIssueData",
            "deps": "js/issueHelperService",
            "inputData": {
                "title": "{{data.title.dbValue}}",
                "description": "{{data.description.dbValue}}",
                "type": "{{data.types.dbValue}}",
                "assignee": "{{data.assignee.dbValue}}"
            }
        },
        "reloadPrimaryWorkarea": {
            "actionType": "Event",
            "method": "Event",
            "inputData": {
                "events": [ {
                    "name": "primaryWorkarea.reset"
                } ]
            }
        },
        "closeDialog": {
            "actionType": "JSFunction",
            "method": "closeDialog",
            "deps": "js/dialogService",
            "inputData": {
                "editContext": null,
                "popupId": "{{subPanelContext.popupOptions.popupId}}"
            }
        },
        "clear": {
            "actionType": "JSFunction",
```

```
            "method": "clearData",
            "inputData": {
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        },
        "reset": {
            "actionType": "JSFunction",
            "method": "resetData",
            "inputData": {
                "data": "{{data}}",
                "formProp": "{{formProp}}"
            },
            "deps": "js/issueHelperService"
        },
        "selectAppropriateTeam": {
            "actionType": "JSFunction",
            "method": "selectAppropriateTeam",
            "inputData": {
                "selectedType": "{{data.types.dbValue}}"
            },
            "outputData": {
                "team.dbValue": "",
                "team.uiValue": ""
            },
            "deps": "js/issueHelperService"
        }
    },
    "dataProviders": {
        "dataProviderForIssueTypes": {
            "response": [ {
                    "propDisplayValue": "Feature",
                    "propInternalValue": "Feature"
                },
                {
                    "propDisplayValue": "Documentation",
                    "propInternalValue": "Documentation"
                },
                {
                    "propDisplayValue": "Usability",
                    "propInternalValue": "Usability"
                },
                {
                    "propDisplayValue": "Test",
                    "propInternalValue": "Test"
                }
            ]
        },
        "dataProviderForAssignees": {
            "response": [ {
                    "propDisplayValue": "John Doe",
                    "propInternalValue": "John Doe"
                },
                {
                    "propDisplayValue": "Sally Smith",
                    "propInternalValue": "Sally Smith"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                },
                {
                    "propDisplayValue": "Micheal Jones",
                    "propInternalValue": "Micheal Jones"
                },
                {
                    "propDisplayValue": "Ed Clark",
                    "propInternalValue": "Ed Clark"
                }
            ]
        }
    },
    "i18n": {
        "add": [ "TrackerAppMessages" ],
        "clear": [ "TrackerAppMessages" ],
        "reset": [ "TrackerAppMessages" ],
        "issueNamingRuleError": [ "TrackerAppMessages" ],
        "priorityRangeError": [ "TrackerAppMessages" ]
    },
    "conditions": {
        "isAnExternalIssue": {
            "expression": "data.isExternal.dbValue === true"
        },
        "isInvalidIssueTitle": {
            "expression": "data.title.dbValue.indexOf('PR-') !== 0"
        },
        "isInvalidPriorityRange": {
            "expression": "data.priority.dbValue < 1 ||
data.priority.dbValue > 3"
        }
    },
    "lifecycleHooks": {
        "onUpdate": [ {
            "action": "selectAppropriateTeam",
            "observers": [ "viewModel.types.dbValue" ]
        } ]
    }
}
```

After building and running the code, the dialog should look like:




Here, we have added a new condition to determine if the checkbox with name `isExternal` has been selected or not.

```
    "conditions": {
        "isAnExternalIssue": {
            "expression": "data.isExternal.dbValue === true"
        }
        ...
    }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

We have added a new widget with name `customerName` and used the condition in its visibleWhen clause. This means, this widget should show up only when this condition is met or the checkbox `isExternal` is selected.

```
<aw-textbox prop="data.customerName" visible-
when="conditions.isAnExternalIssue"></aw-textbox>
```

This is a simple example to show how to control the visibility of a widget dynamically.

Now, the interesting part.

## Associating dynamic Validations to widgets

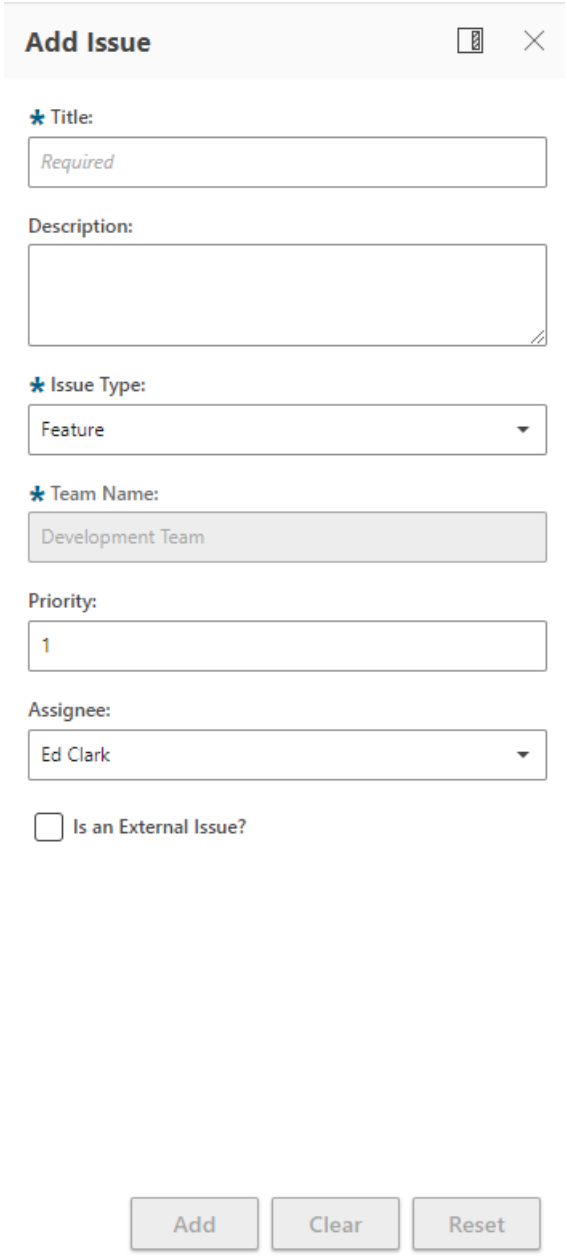
## The Requirement

Let's assume we have a requirement that when there is an external issue, the customer name is mandatory. In this case, what you would do is mark the `isRequired` property on `customerName` to be mandatory like:

```
{
    ...
    "data":{
        ...
        "customerName": {
            "displayName": "Customer Name",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "true"
        },
        ...
    }
}
```

After building and running the code, the dialog should look like:


Unpublished work. © 2025 Siemens

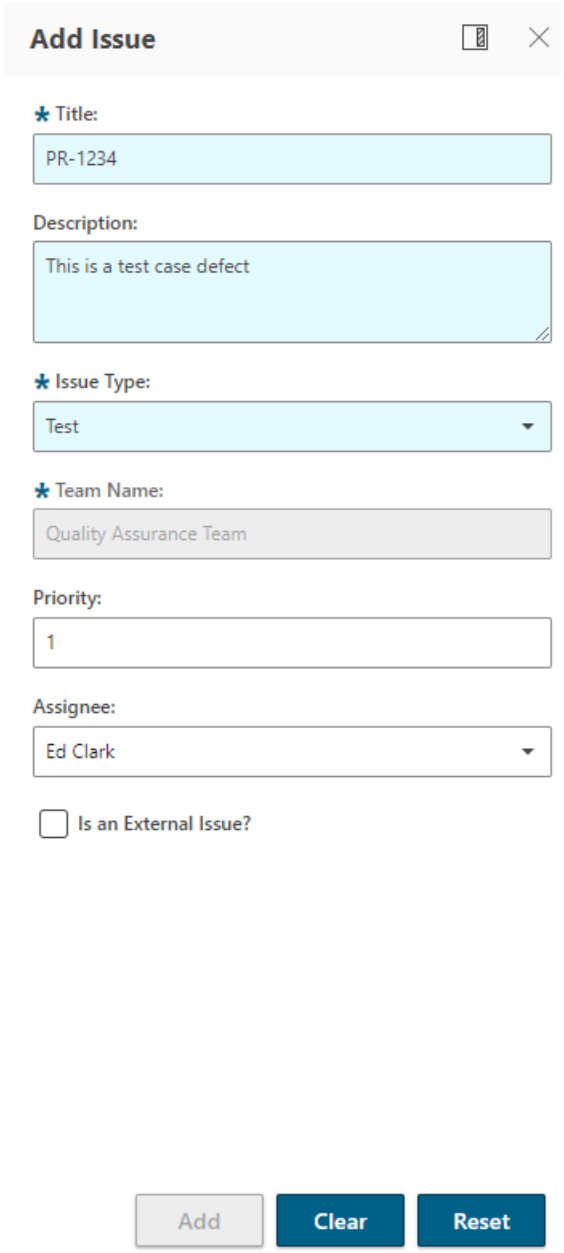



Okay - so far. Now, try to create an issue by filling up the required properties on this form that we have built.

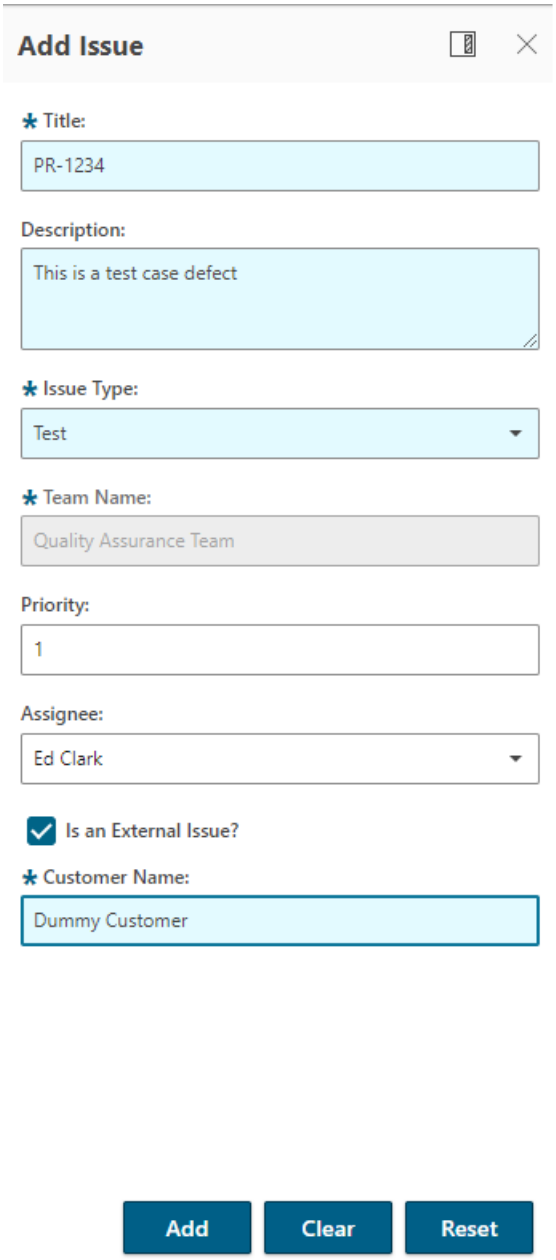
## The Problem

**What happens to the enablement of Add button?**


Siemens Web Framework
Unpublished work. © 2025 Siemens


Add Issue
* Title:
PR-1234
Description:
This is a test case defect
* Issue Type:
Test
* Team Name:
Quality Assurance Team
Priority:
1
Assignee:
Ed Clark
Is an External Issue?
Add
Clear
Reset

It is not getting enabled **unless** you select that this is an External issue and provide a customer name




**This was not expected!**

## The Analysis

Let us step back and see that we have marked this property `customerName` as Required. And have also rendered it conditionally. Now, irrespective of whether it is rendered or not, the form will not become valid unless *all* the required properties have been specified.

What we really want is that `customerName` should be mandatory *only when* the issue being created is an external one i.e. when the checkbox `isExternal` is selected, then change the `isRequired` of


Unpublished work. © 2025 Siemens

`customerName` to true else keep it as false or optional. This is where **dynamic validation** comes to rescue.

## The Solution

Let's add the code first; We will first add an action `markCustomerNameAsRequired` to the checkbox. This ensures that this action will be invoked every time the checkbox is checked/unchecked.

```
<aw-panel>
    <aw-panel-body>
        <aw-textbox prop="data.title"></aw-textbox>
        <aw-textarea prop="data.description"></aw-textarea>
        <aw-listbox prop="data.types"></aw-listbox>
        <aw-textbox prop="data.team"></aw-textbox>
        <aw-numeric prop="data.priority"></aw-numeric>
        <aw-widget prop="data.assignee"></aw-widget>
        <aw-widget prop="data.isExternal"
action="markCustomerNameAsRequired"></aw-widget>
        <aw-textbox prop="data.customerName" visible-
when="conditions.isAnExternalIssue"></aw-textbox>
    </aw-panel-body>
    <aw-panel-footer>
        <aw-button action="add" enable-
when="formProp.attributes.valid">{{i18n.add}}</aw-button>
        <aw-button action="clear" enable-
when="formProp.attributes.dirty">{{i18n.clear}}</aw-button>
        <aw-button action="reset" enable-
when="formProp.attributes.dirty">{{i18n.reset}}</aw-button>
    </aw-panel-footer>
</aw-panel>
```

Let us define this action in JSON and also mark the property as optional in the VMP definition:

```
{
    "data":{
        ...
        "customerName": {
            "displayName": "Customer Name",
            "type": "STRING",
            "dbValue": "",
            "isRequired": "false"
        }
        ...
    },
    "actions:
    {
        "markCustomerNameAsRequired": {
            "actionType": "JSFunction",
            "method": "markCustomerNameAsRequired",
            "inputData": {
                "customerName": "{{fields.customerName}}",
                "isExternal": "{{fields.isExternal}}"
```

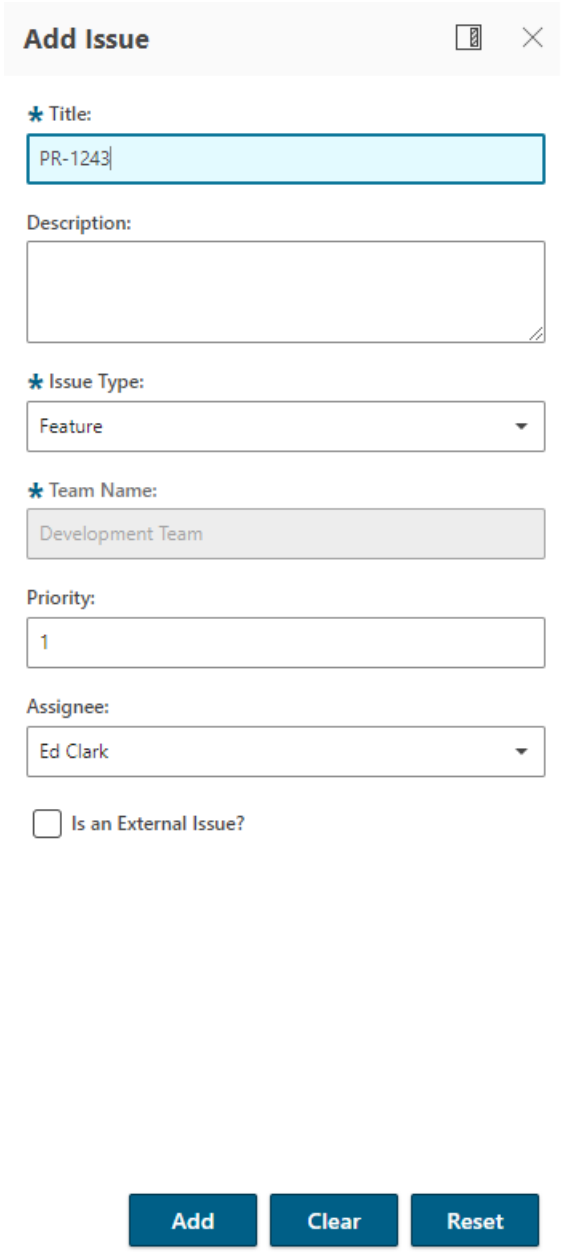
```
            },
            "deps": "js/issueHelperService"
        }
    }
    ...
}
```

Here is the corresponding JS:

```
export const markCustomerNameAsRequired = ( customerName, isExternal ) => {
    customerName.update( '', { isRequired: isExternal.checked },
{ markModified: false, runValidation: true } );
};
```

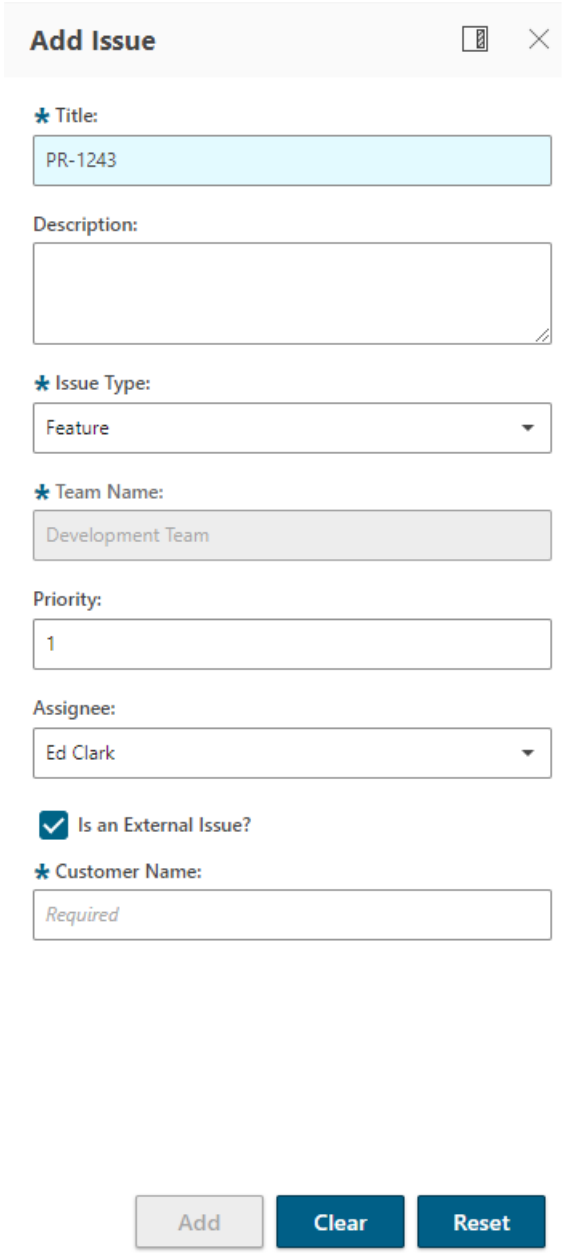
After building and running the code, the dialog should look like:


Unpublished work. © 2025 Siemens



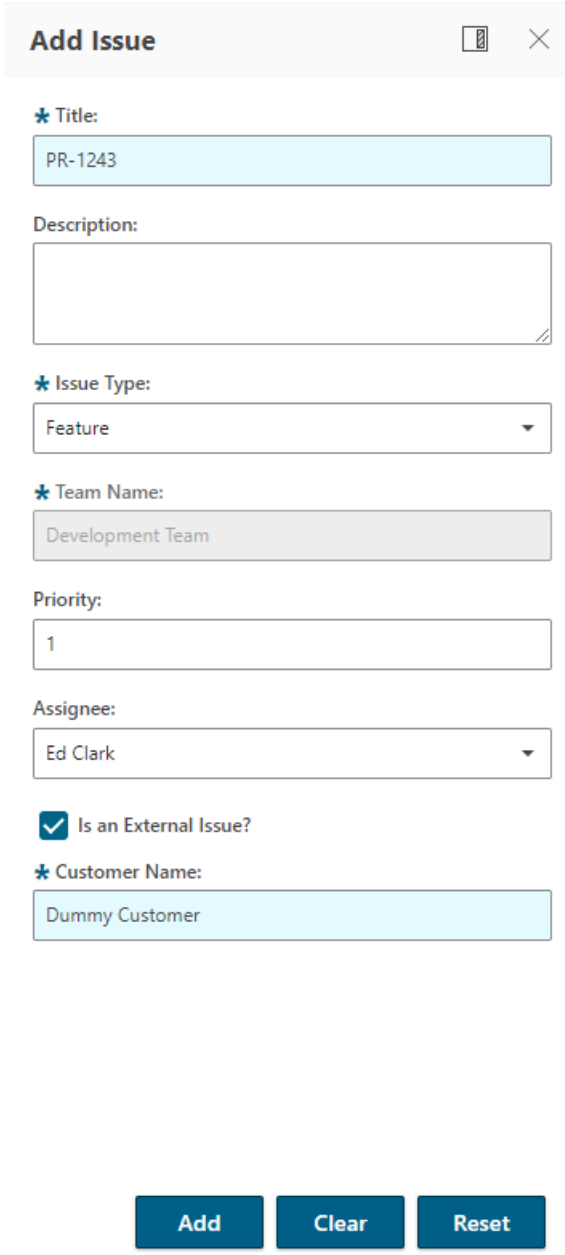

See, the Add button is enabled and we get what we need.

When the checkbox `isExternal` is checked, the Add button is disabled because the required property `customerName` is not specified.




Once the required properties are specified, the Add button becomes enabled:


Siemens Web Framework
Unpublished work. © 2025 Siemens




This meets our exact requirements.

## The Conclusion

Let's understand this a bit:

1. We have specified an action on the checkbox. This ensures that when the checkbox is selected, the specified action is invoked.
2. In the action definition, we have programmatically updated the `isRequired` field on `customerName`. This is using our standard way to update a field using the update API available

on it. The second argument to this API is taking the properties to be updated. And the third parameter is governing whether or not to mark the form as dirty (markModified) and whether or not to run the validation again on the form.

```
export const markCustomerNameAsRequired = ( customerName, isExternal ) => {
    customerName.update( '', { isRequired: isExternal.checked },
{ markModified: false, runValidation: true } );
};
```

So, every time the `isExternal` checkBox is checked, we are instructing the system to update the `isRequired` to be toggle of the checked state of the check box (i.e. if the `isExternal` is selected, make it a required property; if the `isExternal` is de-selected, make it an optional property ). Along with this we are also instructing the system to not make the form as dirty when this dynamic update is being done. And validate the form again, so the dependent visibility like that of add button can be re-computed and we get the desired result. More reference is available **here**

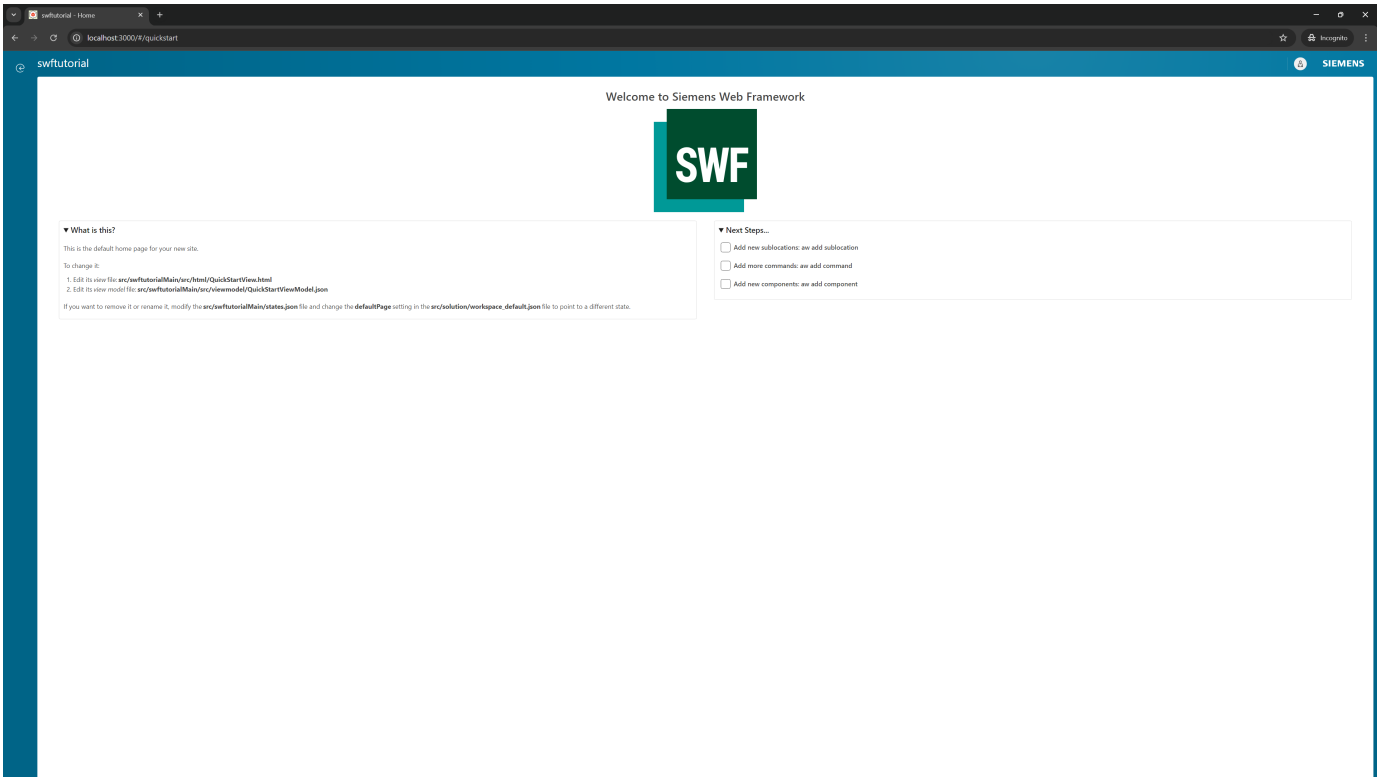

Siemens Web Framework
Unpublished work. © 2025 Siemens

# 5. Creating your solution

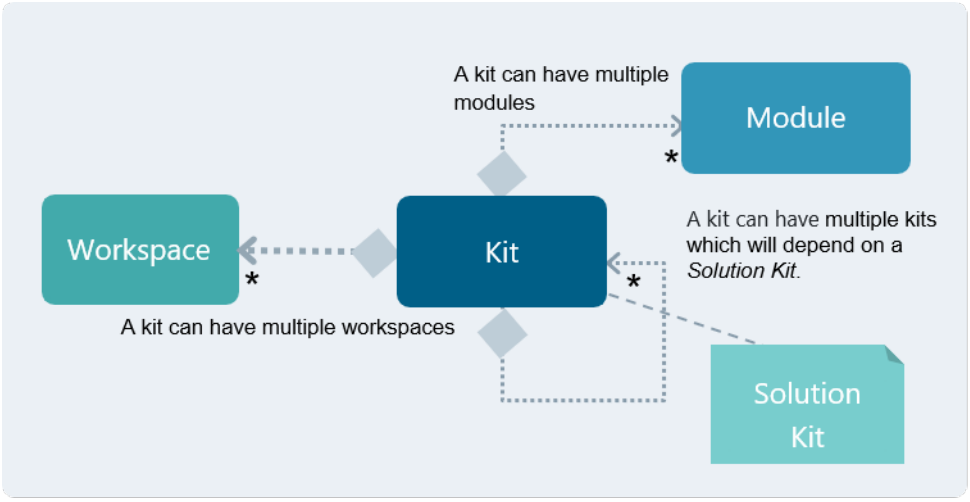
A declarative web application using Siemens Web Framework is based on configuration, allowing you to create a solution(read your web application) by following the stages as follows:

## Generate a basic web application solution

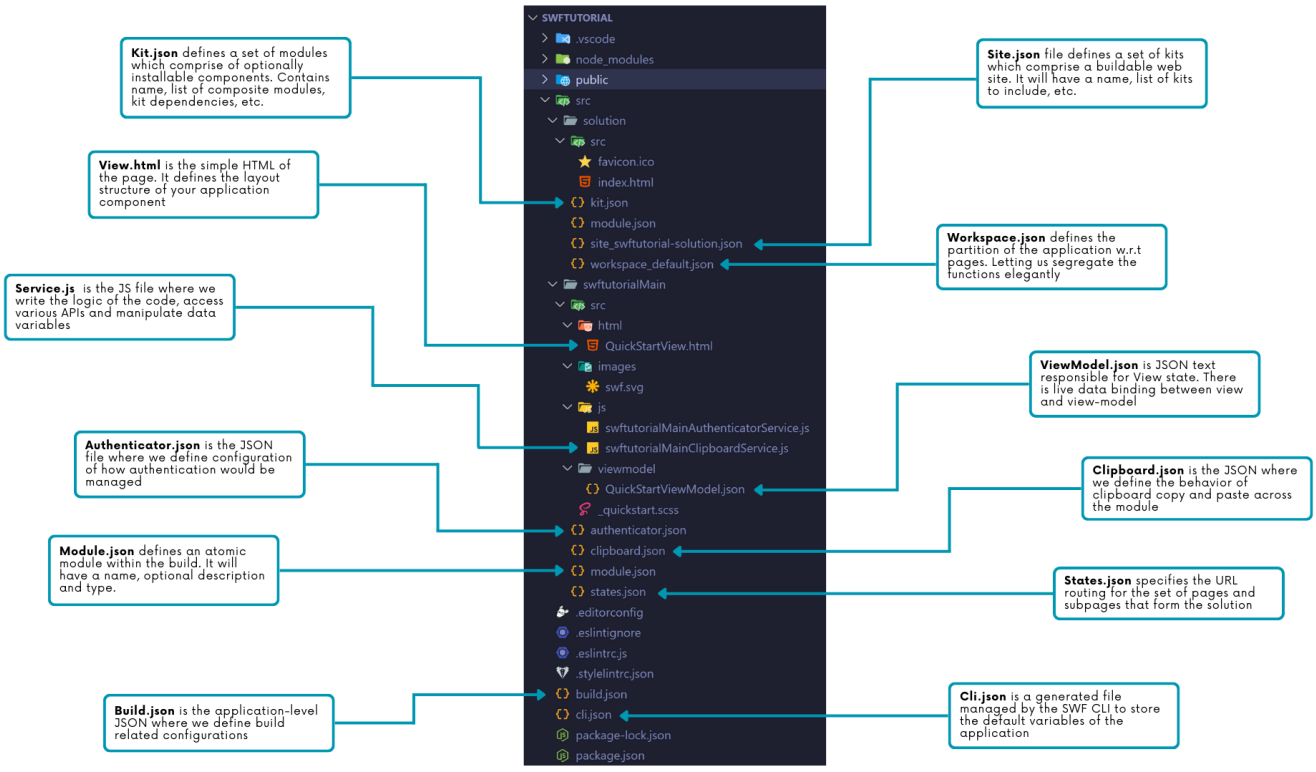
Siemens Web Framework provides a starter application, called **Solution Generator** to get you started with a **Solution** or a web application with a basic configuration. A **Solution**, over here, is a web application that has been built on Siemens Web Framework. Here is a tutorial to create a solution using **Solution Generator**. This would provide a solution, which when launched looks like:




The basic solution generated consists of a Site, Kit, Workspaces, Module, State, Basic View and View Model. The correlation between these concepts can be seen below:

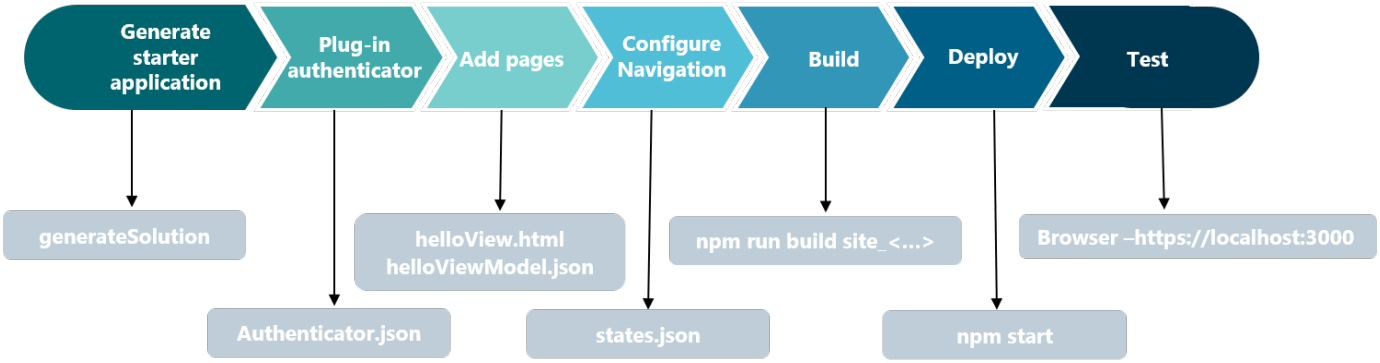



Here is the structure of the files and folders generated as part of solution generator.




The stages for creating the solution are mapped to specific files and commands as shown in below image.


Siemens Web Framework
Unpublished work. © 2025 Siemens

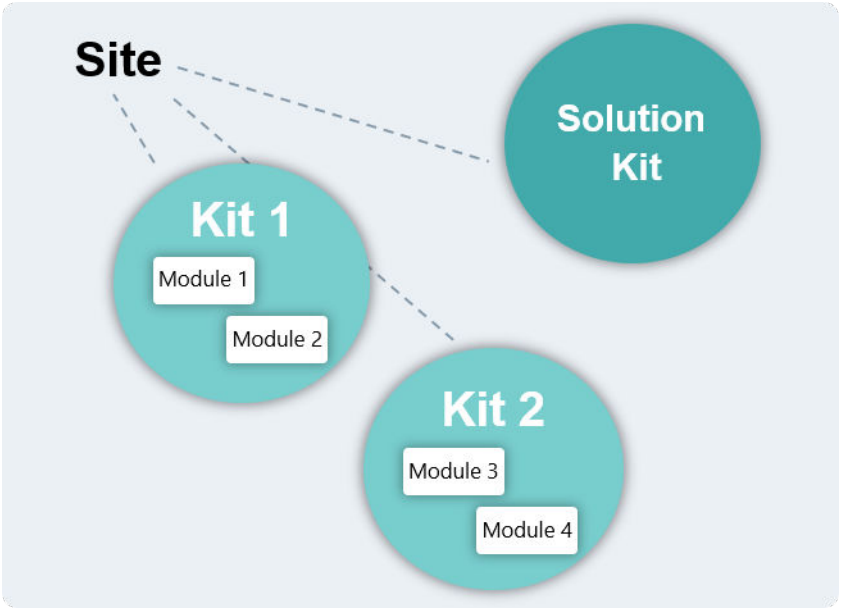



## Site

**Site** is the name given to the deployment package for the Solution that is deployed to an application server. In the above image *site_helloWorld.json* is the configuration file that got generated. Site comprises of one or more **Kits**.

## Kit

**Kit** is a combination of **Modules** that are to be included in a site build. The name of the kit is a unique identifier used by the build system. In a solution, the user can have only one *Solution Kit* inside the *src\solution* folder. It can have multiple kits which will depend on a *Solution Kit*.




In a simplified way.

## Workspaces

A Solution normally provides a set of **Workspaces** to configure some UI aspects of the solution. Within the same solution, we can partition capabilities in order to tailor for given categories/groups of the user. The solution provider determines which workspaces are provided and which one is the default for a fixed group of users.

- A workspace determines a set of available pages(views and view models, definition to follow) within a solution when the workspace is active.
- Solution provider determines workspaces to include and the default workspace through solution configuration through **kit.json**.

## Modules

**Modules** are highly self-contained, with distinct functionality, allowing them to be shuffled, removed, or added as necessary, within kits, without disrupting the entire system.

## States

**States** definitions specify the URL routing for the set of pages and sub pages that form the solution. This controls how a given URL fragment is associated to a given page or subpage's **view** and **view model** definition. In the Siemens Web Framework, each page and subpage can be defined as a unique state. These states get stored in a states.json file on a per module basis. Each states.json file lives as a peer to the module.json. When constructing a **Site** the Siemens Web Framework tooling pulls together the state definitions.

## View and View Model

**View** and **View Model** is declarative definition of UI and its behavior.


Unpublished work. © 2025 Siemens

## Build a site and Deploy

The system to concatenate the configurations from different modules (client-side configurations) and create the solution deployment package known as a **Site**.

```
aw build site_tutorial
```

This command creates the **Site** to deploy for the Solution defined by site_\<mysolution\>.json definition inside your solution moduleResulting **Site** is placed in "/out/site/mysolution".

```
aw start
```

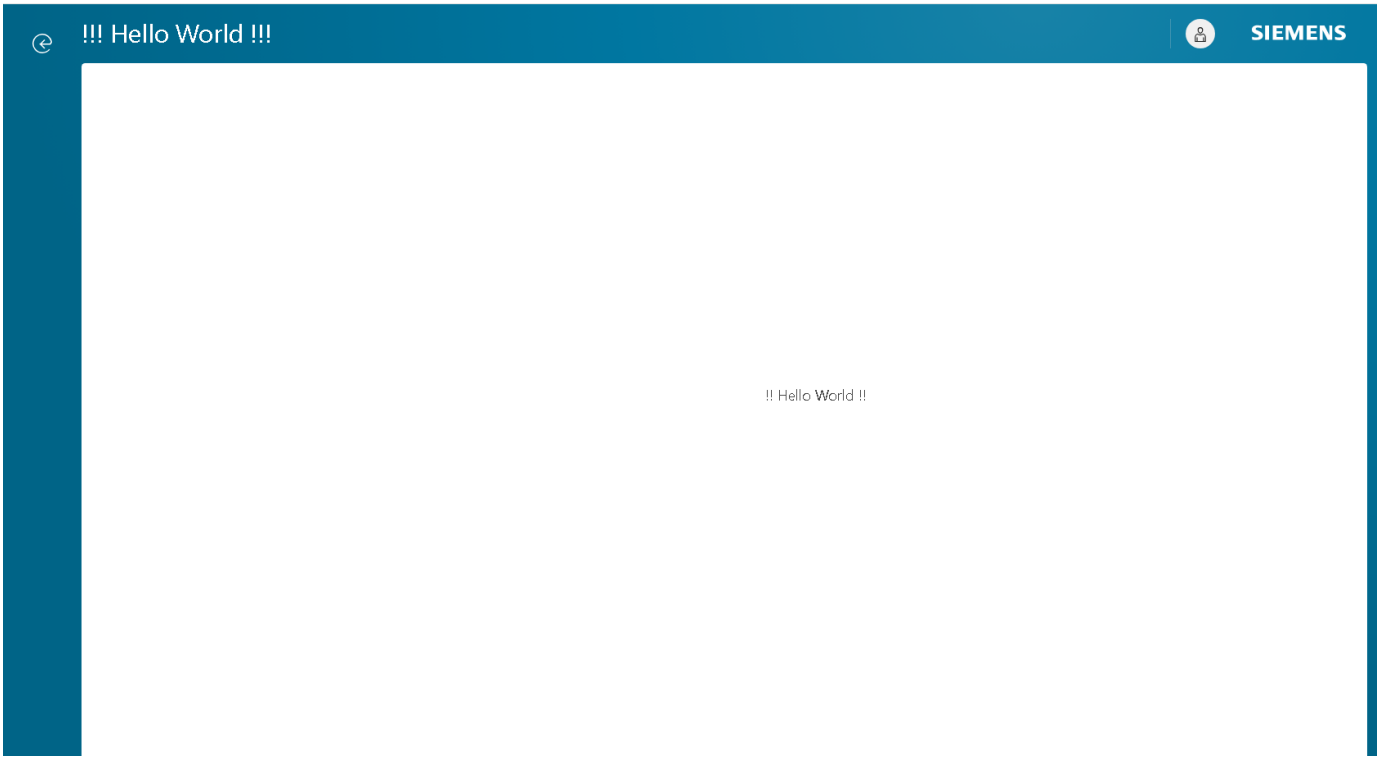
This common deploy the **Site** to a live-server server for testing. The live-server command line provides the URL (including port) to use.

To build and deploy the site, follow Step 3 of Your First Hello World Application **Your First Hello World Application** tutorial.

## Add framework provided common frame to your solution

The framework provides a common frame for the application. This is to bring in a uniform look and feel to all web application offerings from PL Cloud. To know how this can be done, please visit **Add Common Frame**. A tutorial is available with detailed steps to follow.


Unpublished work. © 2025 Siemens




## Authentication

Another basic configuration that is available in the starter application is the authenticator. **Authentication** defines the client authenticator. It is the process by which users securely access a server or remote computer by exchanging a Digital Certificate. *authenticator.json* file available at "src/helloWorld/helloModule" can be referred to as the default authenticator provided by the framework.

## Static Code Analysis

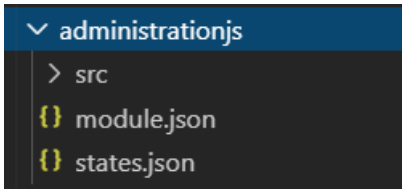
Finally, the compile time audit tool is available. A solution can be validated using the provided static analysis **Audit tooling**. This helps to detect typos, code problems, declarative constructs, etc. during development whereever possible before debugging at runtime.

# Modules

Modules are highly self-contained with distinct functionality, allowing them to be shuffled, removed, or added as necessary, without disrupting the system as a whole.

The `module.json` file defines an atomic module within the build. It will have a name, optional description & type. To add a new module, you simply need to create a module.json file & provide the required information. Please follow an example of an existing module.


Siemens Web Framework
Unpublished work. © 2025 Siemens




The simplest `module.json` file can be like:

```
{
    "name": "myModuleName",
    "description": "This module contains something very useful."
}
```

## List of Common Properties used in `module.json` files

| Module Type | Property | Type | | Description |
|---|---|---|---|---|
| * | `name` | string | required | unique module name |
| * | `description` | string | | module description |
| native<br>java<br>repo | `srcDir` | string | | source directory; defaults to `"src"` if not provided |
| * | `testDir` | string | | test directory; defaults to `"test"` |
| native | `pathOffset` | string | | path offset into war/site directory; defaults to `assets` |
| native | `skipAudit` | boolean | | skip audit; defaults to `false` |
| native | `copyAssets` | object | | Copies static assets to the output folder (see the example) |
| native<br>java | `sonarCoverageExclusions` | array | | array of paths for SonarQube Coverage exclusions |

## Example: Copying Static Assets

The following code example adapted from the Showcase application shows how to use the `copyAssets` property to copy static assets to the output folder:

```
{
  "name": "showcaseCoreModule",
  "description": "This module contains core framework of afx-showcase",
  "copyAssets": [
    {
      "baseDir": "src/pdf",
```

```
      "fileRegex": "**/*",
      "destPath": "assets/pdf",
      "copyDirectory": false
    },
    {
      "baseDir": "src/html",
      "fileRegex": "*.html",
      "destPath": "assets/html",
      "copyDirectory": false
    }
  ]
}
```

# Kits

Kit is a logical grouping of **modules** that are to be included in the site build. A module which has a `kit.json` is considered as a kit. The name of the kit, for e.g. "myKitName", has to be a unique identifier as required by the build system.

The `kit.json` file defines a set of modules which comprise optionally installable components. It will have a name, list of composite modules, kit zip filename, etc. To add a new kit, you simply need to create a new kit.json file & provide the required information. Please follow an example of an existing module.

```
{
    "name": "myKitName",
    "level": 5,
    "modules": [
        "myModuleName"
    ],
    "kitDeps": [
        "afx",
        "someOtherKitName"
    ]
}
```

## List of properties to be used in kit.json

| Field name | Value type | Mandatory | Description |
|---|---|---|---|
| `name` | string | yes | Name of the kit |
| `level` | number | yes | dependency level from 1-5; your kit can then depend on kits with level less than your kit |
| `modules` | array | yes | modules included in kit |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `kitDeps` | array | | list of other kits which are required by this kit |
| `solutionDef` | object | | solution definition |
| `imageRepositoryConfiguration` | object | | This is the declarative way of extension module where user can specify the image configuration on framework |
| `bundles` | object | | Array of the files which should be the part of the bundles. Instead of loading all files , bundles get loaded as and when require |
| `karmaKitDeps` | array | | array of keys required for Karma unit test execution |
| `srcModule` | string | | module name for base kit files |
| `configuration` | object | | Object to define how each configuration would be processed. `depsPath` specifies the array of path where the dependencies to be loaded are located for each of the configuration object |

A kit with a solution definition is a `solution` which is deployed. Its `kit.json` has a `solutionDef` configured, like:

```
{
    "name": "myKitName",
    "level": 5,
    "modules": [
        "myModuleName1",
        "myModuleName2"
    ],
    "kitDeps": [
        "afx",
        "someOtherKitName"
    ],
    "solutionDef": {
        "solutionId": "tutorial-solution",
        "solutionName": "tutorial-solution",
        "solutionVersion": "1.0",
        "browserTitle": "tutorial",
        "brandName": "Brand tutorial",
        "companyName": "Siemens PLM Software",
        "copyrightText": "Copyright © ..... Inc.",
        "workspaces": [
            "newWorkspace"
        ],
        "authenticator": "helloAuthenticator",
        "defaultWorkspace": "newWorkspace"
```

```
    },
    "configuration": {
        "commandsViewModel": {
            "output": "json",
            "definition": "module",
            "uniqueDepth": 2,
            "depsPath": ["actions.deps"]
        }
    }
}
```

Know more about **workspaces**.

## Sites

Site is the name given to the deployment package for the **Solution** that is deployed to an app server.

The `site_\<mysolution\>.json` file defines a set of kits which comprise a buildable web site. It will have a name, list of kits to include, etc. To add a new site definition, you simply need to create a new *site.json** file & provide the required information. Please follow an example of an existing module.

```
{
    "name": "siteName",
    "solutionName": "mySolutionKitName",
    "kits": [
        "mySolutionKitName",
        "myKitName"
    ]
}
```

For example :

```
{
    "name": "showcase",
    "solutionName": "showcase",
    "kits": [
        "showcase",
        "active-workspace-showcase"
    ]
}
```

### List of properties to be used in site.json

| Field name | Value type | Mandatory | Description |
|---|---|---|---|
| name | string | Yes | Name of the Site. For eg: "tutorial" |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `solutionName` | string | | Name of the Solution's kit . For eg: "tutorial-kit" |
| `kits` | array | | Array of the solution kit and all other kits which are required to build the solution |
| `locale` | array | | Array of the custom supported locales for the solution |

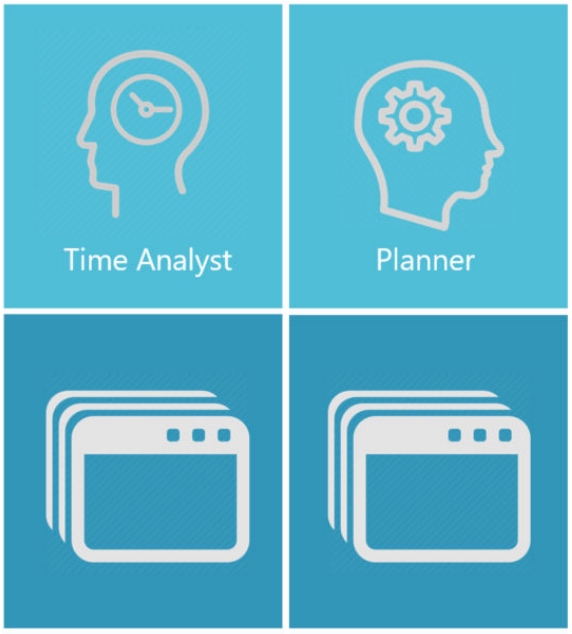
## Workspaces

A workspace partitions the user experience and control capabilities in a declarative way. It helps to define single Solution and yet provide way to configure logically separate functional behavior.

With workspaces,

- Capabilities can be partitioned within same Solution in order to tailor for given categories of user
- Some UI aspects of the application can be configured
- Solution provider configures which pages are made available and URL routing for given categories of users

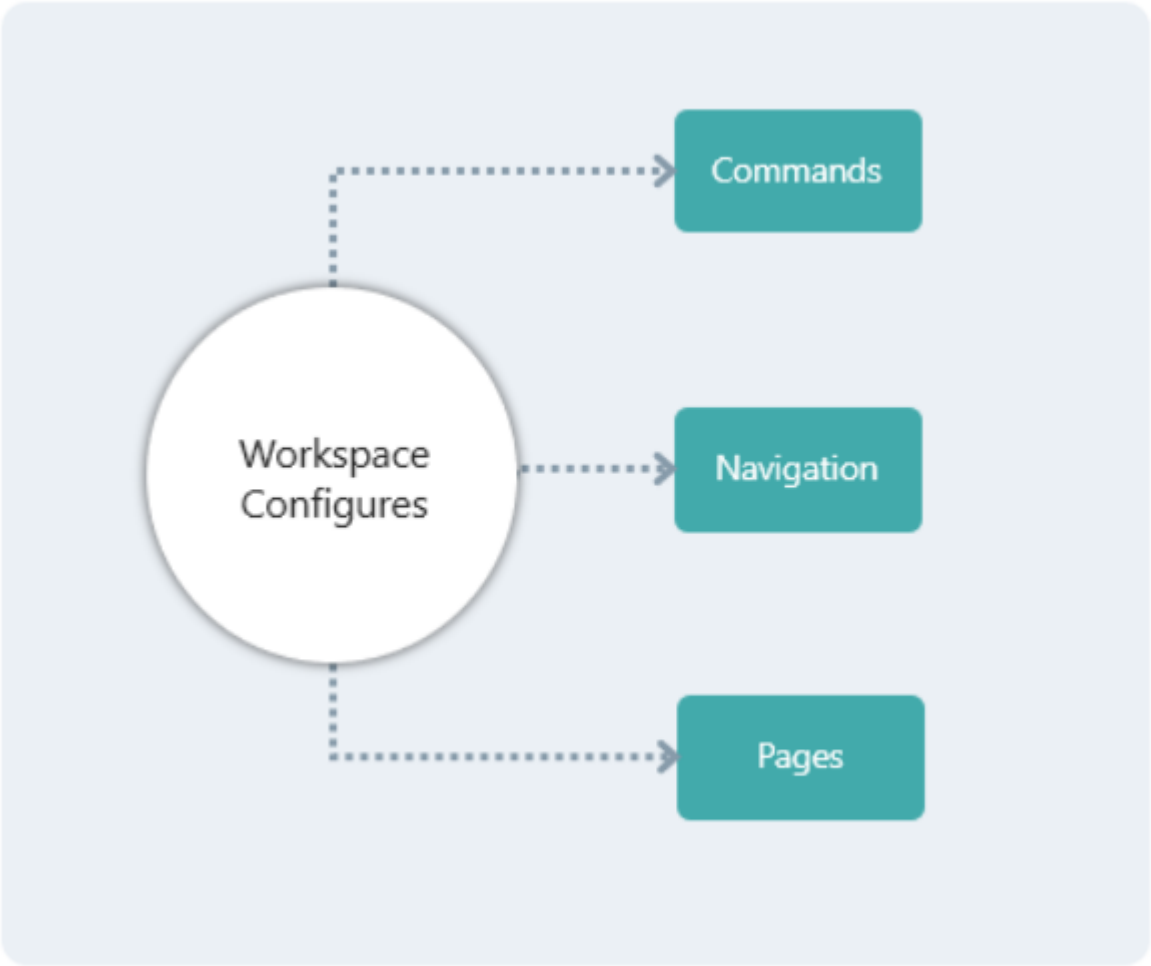
A solution can provide a set of OOTB workspaces. Solution provider maps Workspaces for given categories of user.





Unpublished work. © 2025 Siemens

## Supported Use-cases




- User interface and behavior configuration
  - Ability to control what Pages user can navigate to.
  - Ability to control what Commands user can use.
  - Ability to control what Navigation behavior (landing page, navigate actions) user can use.
- Ability to pick and choose individual declarative artifacts vs bulk inclusion (Inclusive vs Exclusive Workspaces)
  - Inclusive: Any new declarative artifact that is getting added to the system will be available to inclusive workspaces by default.
  - Exclusive: Any new declarative artifact that is getting added to the system will NOT be available until the workspace is explicitly modified and re-deployed.
- Build/Kit Infrastructure changes
  - Ability to define a new workspace on client side as a JSON file.
  - Ability to modify existing workspaces in the kitted environment and deploy it.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Workspace construct

```
{
    "schemaVersion" : "1.0.0",
    "workspaceId": "sampleWorkspace",
    "workspaceName":"sampleWorkspace",
    "workspaceType": "Exclusive",
    "defaultPage": "marsPage",
    "availablePages": [
        "mainPage",
        "goCategory",
        "kit::kit1"
    ],
    "availableNavigations":,
    "availableContextConfigurations": [
        "projectContext",
        "userContext",
        "groupRoleContext"
    ],
    "includedCommands": [
        "Awp0GoHome",
        "Awp0ShowHomeFolder",
        "cmdSignOut",
        "cmdViewProfile",
        "kit::workflow"
    ],
    "excludedCommands": [
       "kit::change",
       "Awp0Print"
    ]

}
```

- `schemaVersion` (Mandatory): Version of the declarative schema this workspace definition complies with. E.g. 1.0.0
- `workspaceId` (Mandatory): A unique identifier string for the workspace. Value needs to be unique across all the solutions.
- `workspaceName` (Mandatory): A JSON structure providing lookup details of localizable string for the workspace name.
  - source: An i18n file where key for the workspace name needs to be looked up
  - key: A string key which defines the workspace name
- `workspaceType` (Mandatory): A string value specifying the type of workspace. “workspaceType” could be one of the following
  - `Exclusive`: If the workspace is defined as exclusive workspace, user will be able to see only those UI elements (Pages, Commands, Tiles etc) which are 'exclusively' mapped in workspace definition. Any new declarative artifact (added by development teams / customizers) needs to be mapped in workspace definition to appear in the UI. For example, a new page (sub location) “consumerTasks” is being added in customer environment. To allow user to navigate to this page, ID of the new page

needs to be added in “availablePages” field in of exclusive workspaces mapped to that user. After workspace definition is modified, the war needs to be built and re-deployed.

  - `Inclusive`: If the workspace is defined as inclusive workspace, then all the UI elements (Pages, Commands, Tiles etc) created in the solution appear by default, i.e. as good as not using workspace concept itself. In theory no one would use inclusive workspace. It's introduced only to help migration from older release where workspace concept was not there.

- `defaultPage` (Mandatory): The page which will be shown by default when user logs in/changes workspace. Value is the state defined in states.json of the module.
- `availablePages` (Optional): An array of string values where each value is the page name (i.e. state name defined inside states.json) to which user can navigate to while working inside given workspace. The availablePages field will be ignored if the workspace is defined to be inclusive. User will be able to navigate to any page (state) while working inside inclusive workspace. However when workspace is defined to be exclusive, workspace author needs to explicitly mention each of the pages (states) to which user is expected to navigate to while working inside given workspace.

  **Note**

  When workspace author wants to make all of the pages available from a specific kit, then simply mention `kit::kit-name` inside the `availablePages` list

- `availableNavigations` (Optional): An array of JSON objects defining Navigate action behavior. For e.g. workspace author can configure default navigation to Search location instead of 'Show Object' location on opening Change Notice for Change administrator.
- `availableContextConfigurations` (Optional): An array of strings where each value is the `contextId` for each of link which would show-up in user context control. The `contextId` is defined inside `contextConfiguration.json`. For more details please **refer**.

  **Note**

  If availableContextConfigurations is not defined, then all of the contextIds/contextConfiguration available in the kit would show up.

- `includedCommands` (Optional): An array of commands where each value is the command Id (i.e. commands defined in the commandsViewModel.json) which needs to be seen inside the given workspace. The includedCommands field will be ignored if the workspace type is inclusive. In inclusive workspace type, the user will see every command. When workspace type is exclusive, the workspace author can explicitly mention the commands which is expected to be seen in the workspace using includedCommands. If the includedCommands is mentioning empty array , all command will be excluded from workspace.

  **Note**

  When workspace author wants to make all of the commands available from a specific kit, then simply mention `kit::kit-name` inside the `includedCommands` list. The command has atleast one valid handler defined. The visibleWhen and enableWhen conditions associated with command evaluates to TRUE


Siemens Web Framework
Unpublished work. © 2025 Siemens

- `excludedCommands` (Optional): An array of commands where each value is the command Id (i.e. commands defined in the commandsViewModel.json) which the user can exclude inside the given workspace. The excludedCommands field will be ignored if the workspace type is inclusive. In inclusive workspace type, the user will see every commands. When workspace type is exclusive, the workspace author can explicitly mention the commands which should not to be seen inside the given workspace. Excluded commands will override included commands.

  **Note**

  When workspace author wants to make all of the commands excluded from a specific kit, then simply mention `kit::kit-name` inside the `excludedCommands` list. The command has atleast one valid handler defined. The visibleWhen and enableWhen conditions associated with command evaluates to TRUE

## Default Workspace concept

| Use Case | Default Workspace Assigned? | Other Assigned Workspaces | Workspace(s) in Profile Menu | Applied Workspace | Notes |
|---|---|---|---|---|---|
| 1 | No | None | Not shown | Default | User is automatically placed in the default workspace – there is no display of the current workspace nor choice to change workspaces in the UI |
| 2 | No | One – Author | Not shown | Author | User is automatically placed in the “Author” workspace – there is no display of the current workspace nor choice to change workspaces in the UI |
| 3 | No | Two or more – Design Engineer, Mfg. Engineer, Manufacturing Line Operator, Etc. | Design Engineer (selected)Mfg. EngineerManufacturing Line Operator | Design Engineer (first time)Last used from there on out | User is placed in the first (or last used) workspace, e.g. Design Engineer – the current workspace is displayed in the profile menu and when clicked there is a choice to change workspaces to one of the other ones, e.g. Mfg. Engineer, Manufacturing Line Operator, etc. |

| | | | | | |
|---|---|---|---|---|---|
| 4 | Yes | None | Not shown | Default | Same as case 1 - User is automatically placed in the default workspace – there is no display of the current workspace nor choice to change workspaces in the UI |
| 5 | Yes | One – Author | Author (selected)Default Workspace | Author (first time)Last used from there on out | User is automatically placed in the “Author” workspace (or last used) workspace – but what is different from case 2 is that the current workspace (Author) is displayed in the profile menu and when clicked there is a choice to change workspace to the default workspace |
| 6 | Yes | Two or more – Design Engineer, Mfg. Engineer, Manufacturing Line Operator, Etc. | Design Engineer (selected) Mfg. Engineer Manufacturing Line Operator Default Workspace | Design Engineer (first time)Last used from there on out | User is placed in the first (or last used) workspace, e.g. Design Engineer – the current workspace (Author) is displayed in the profile menu and when clicked there is a choice to change workspace to one of the other ones, e.g. Mfg. Engineer, Manufacturing Line Operator, etc. and also the Default one. |

## Solution construct

The workspces needs to be defined in “solutionDef” attribute of solution's kit.json file.

```
"solutionDef": {
        "solutionId": "showcase",
        "brandName": "SIEMENS SHOWCASE SOLUTIONS",
        "solutionName": "Showcase",
        "solutionVersion": "1.0",
        "browserTitle": "Showcase",
        "companyName": "Siemens PLM Software",
        "copyrightText": "Copyright © 2023 Siemens",
        "workspaces": [
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "showcaseWorkspace"
        ],
        "defaultWorkspace": "showcaseWorkspace",
        "authenticator": "dummyAuthenticator",
        "clipboard": "dummyClipboard"
    },
```

Here,

- `workspaces:` An array containing workspace IDs of the workspaces to be shipped as part of the solution.
- `defaultWorkspace:` The workspace id for the workspace to be used as default workspace. A Solution typically contains multiple workspaces, however there needs to be one default workspace which every user would see on App launch. Note the default workspace can be configured for the group-role/group/role.

Both “workspaces” and “defaultWorkspace” are mandatory attributes.

## Adding new Workspace

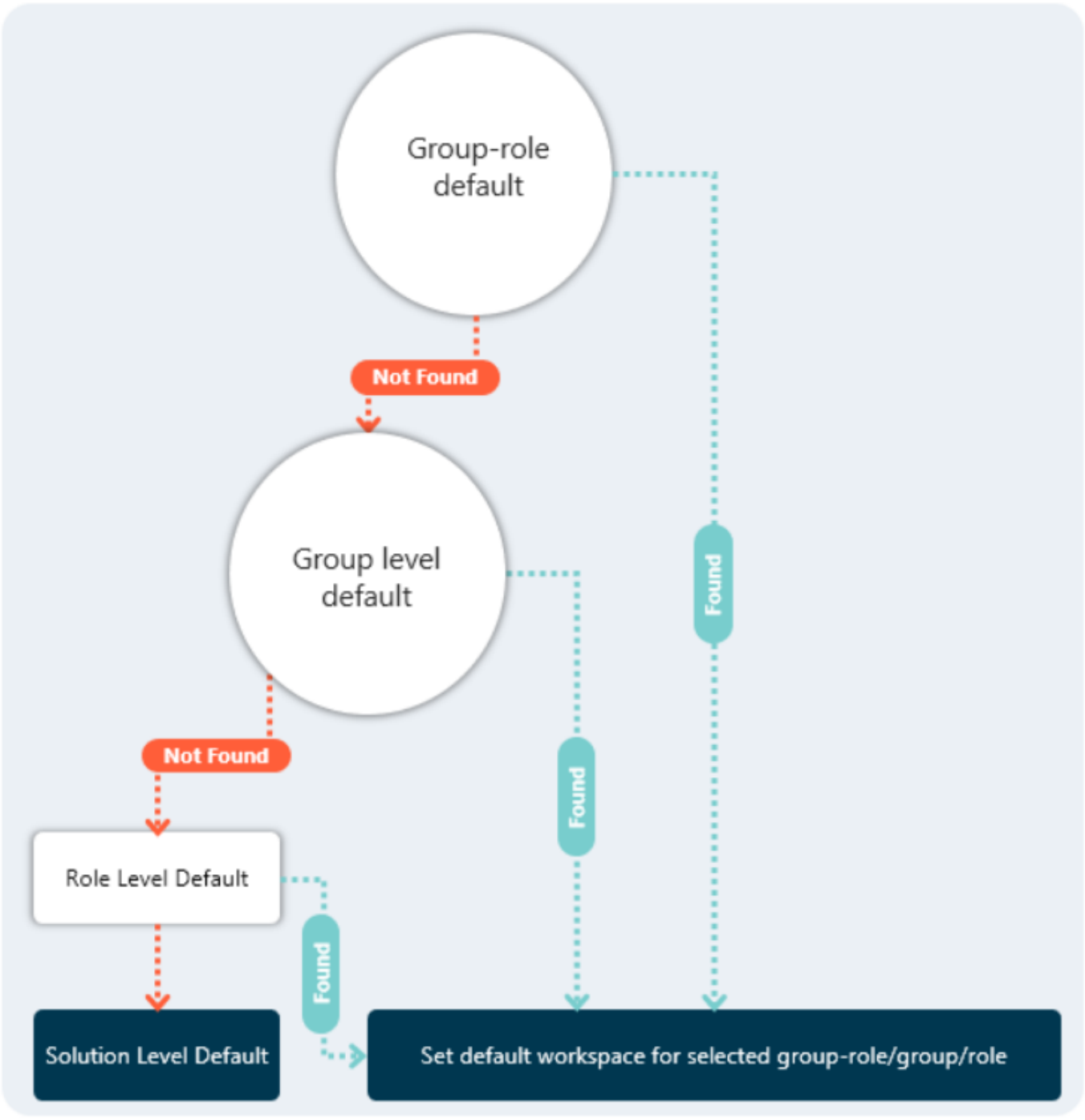
- Create workspace definitions i.e. workspace.json files. Please refer section Workspace Construct for guidance
- Design and Finalize the solution definition. Please refer section Solution Construct for guidance

## Mapping Workspaces to Organization

***Note***: Siemens Web Framework is a Frontend Framework, so it is up to consuming teams to define the Organization structure at their BackEnd. This section is meant to provide guidance on how to map Workspaces to Organization.

Admin can create group level, role level or group-role level default workspaces. It's assumed that consuming team has their own login service implemented. Once login is complete, they will inform Workspace service to initialize the default workspace (defaultWorkspace).

- `Setting defaultWorkspace` Once login REST call returns, they need to fire an event `sessionInfo.updated` with workspace ID of whatever workspace they want user to be logged into Siemens Web Framework has workspaceInitService which listens to this event and overwrites the defaultWorkspace defined in solutionDef section of kit.json




- `Solution level default` : For the selected group-role combination, if there is no default workspace(group-role/group/role level) defined in workspace mappings xml then solution level default is considered. This solution level default is the “defaultWorkspace” defined in solutionDef of solution's kit.json. Please refer section Solution construct for guidance.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Navigation Configuration

As an administrator you can configure behavior for Navigate action via Workspaces. For example, when a specific type of object is opened say `Change Notice`; workspace author can configure to override the default navigation from `Show Object location` to say `Search location`. This configuration can be defined at Workspace level. Here is the syntax for defining Navigation configuration.

```
availableNavigations
{
    "key" : "navigate",
    "page": "samplePage",
    "params":{"param1":"1",
              "param2":"2"
             },
    "activeWhen": "{{data.condition}}",
 }
```

Where:

- `key`: Action id - Semantically meaningful action name, e.g. when I navigate to a URL and perform search action - use key as `search`, or When I navigate to classification library, use key as `classify` as key. Default is `navigate`.
- `page`: The id of the page to navigate to, e.g. 'showObject' if you want to navigate to showObject location.
- `params`: The set of parameters that are added to URL when user is navigated to designated page
- `activeWhen`: It is the condition which decides which navigation configuration to pick

Here are few sample conditions:

```
condition : "myObject.type === 'ItemRevision' &&
ctx.selected.type==='Newstuff Folder'"
condition : "ctx.locationContext['SubLocation'] === 'myFavorites'"
condition : "state.currentState=== 'myFavorites'"
```

Consumer will consume the navigate action with Navigation context and myObject.

Example:

```
 "showHoveredObject": {
       "actionType": "Navigate",
       "navigateTo": "showObject",
       "inputData": {
           "navigationParams": {
               "uid": {
                   "$adapt": "{{commandContext.vmo}}",
                   "path": "uid"
```

```
                }
            },
            "navigationContext": {
                "myObject": {
                    "$adapt": "{{commandContext.vmo}}"
                },
                "ecn": "{{data.i18n.copyTitle}}"
            }
        }
    }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

# 6. Creating User Interfaces

Siemens Web Framework provides all the means to create complex user interfaces, from the layout to the smallest UI element.

- Configure **URL routing and navigation**.
- Create and configure **page-level components**.
- Use built-in **UI elements**.
- Configure **selection management**.
- Manage your application **responsiveness**.
- Leverage View and VewModel **reusability**.

## URL Routing

Siemens Web Framework helps you build Solutions which are Single Page Applications **SPA**. As a single page application, Siemens Web Framework based Solutions do not reload the single page at any point, nor does control transfer to another page. Through the use of a the **location hash** (#) and a third party URL router library, **ui-router**, the perception of navigability of logical separate pages is provided by dynamically modifying the **HTML DOM** for the application.

The basic page for Siemens Web Framework based Solution is very simple HTML5 which identifies a single element with the attribute "**ui-view**". The URL router library then dynamically swaps out content based on the Solution's configuration defined in the various module states.json file.

```
<!DOCTYPE html>
<html>

    <head>
        <script type="text/javascript" src="assets/lib/require-2.1.17/require.min.js" data-main="assets/js/bootstrap.js"></script>
    </head>

    <body>
        <div id="main-view" class="aw-layout-mainView" ui-view></div>
    </body>

</html>
```

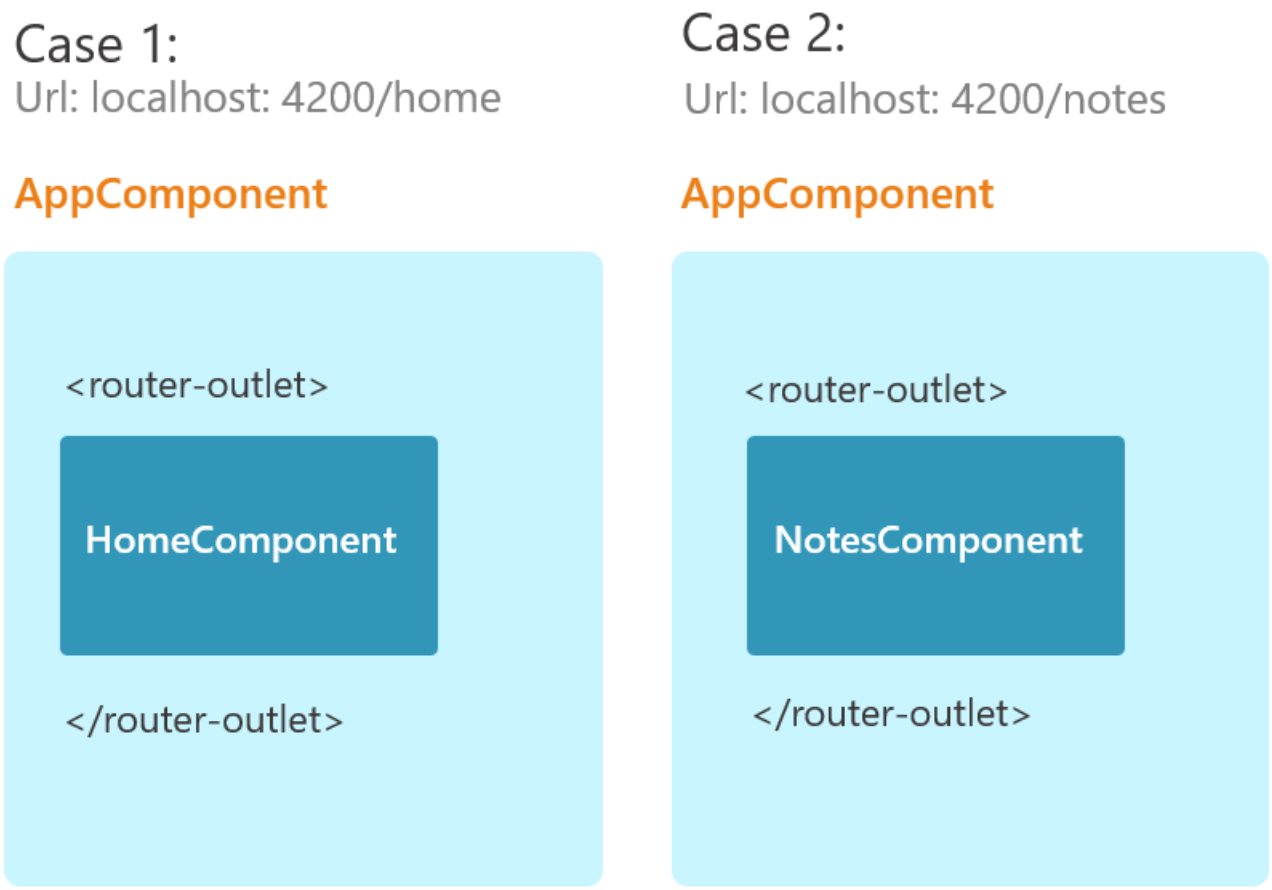
For more details, please refer **https://github.com/angular-ui/ui-router/wiki**

### A tree of states

Consider application as a tree of components. Some of those components, such as the root component, will remain in place over the course of the application. However, we also want the ability to display certain components dynamically, and one way to achieve this is through use of the router. Using the router module it’s possible to define parts of our application which will display different sets of components based


Unpublished work. © 2025 Siemens

on the current url. For example, in a simple note-taking application, you might want to display a home component for one url, and a list of notes for a different url.



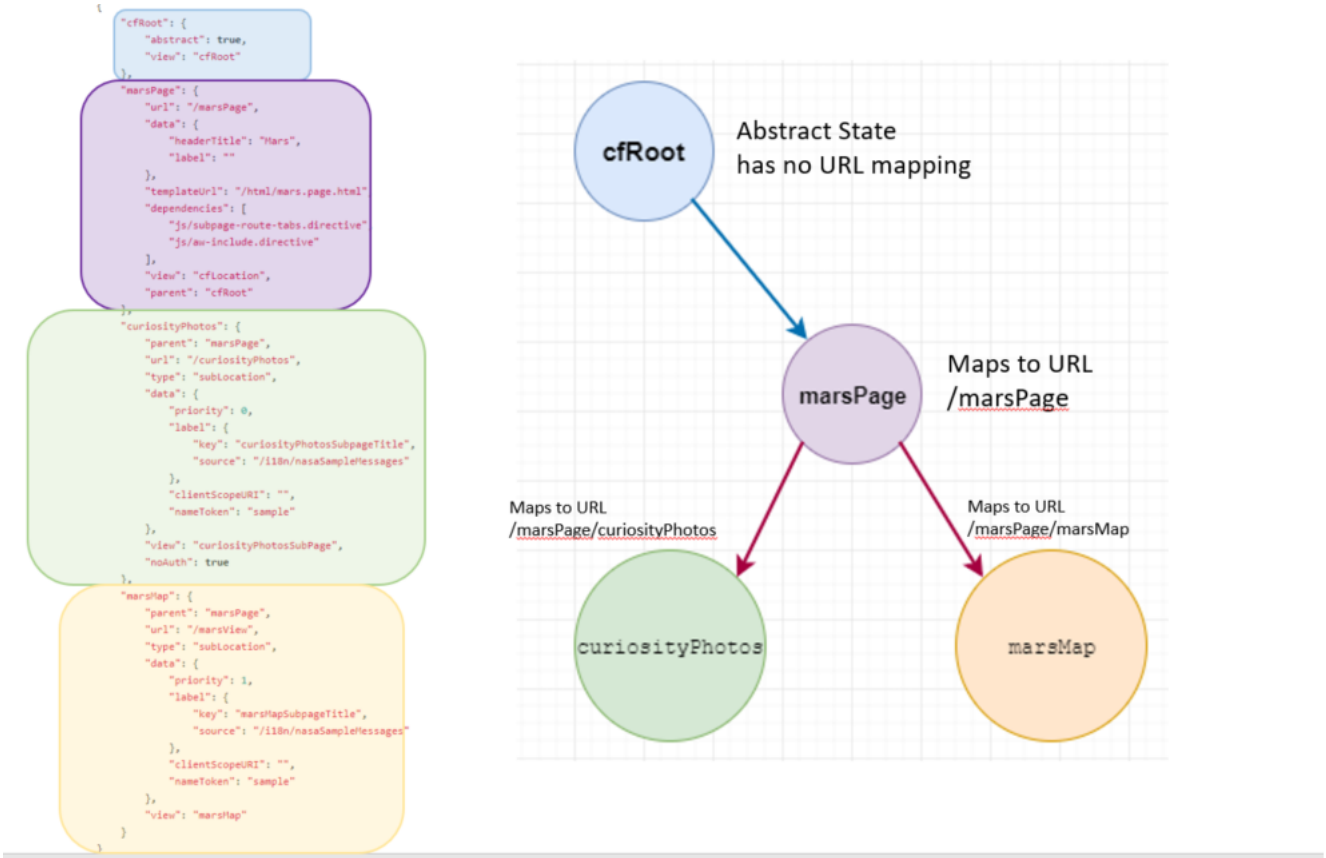

Depending on the url (home or notes), a different component will be rendered. Internally, these routable sets of components are known as router states. The router will model the routable components in an application as a tree of router states. Continuing with the example above, the home page would be a router state, while the components for displaying a list of notes would be another router state.

The core focus of the router is to enable navigation among routable components, which requires the router to render a set of components using an outlet on the page, and then reflect the rendered state in the url. In order to do this, the router needs some way to associate urls with the appropriate set of components to load. It accomplishes this by letting a developer define a router state configuration object, which describes which components to display for a given url.

Router states are defined within a Siemens Web Framework application inside `states.json` file

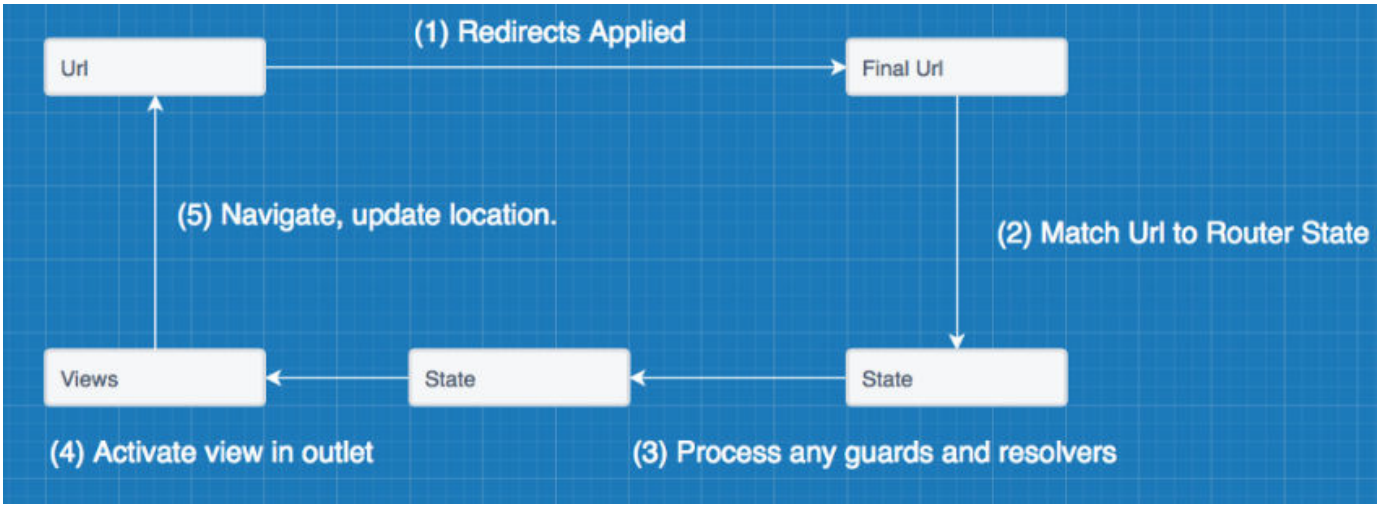

Siemens Web Framework
Unpublished work. © 2025 Siemens

## Sample states.json




## The lifecycle of a router

The router has a series of steps that it cycles through each time the router state changes.
The router navigation cycle runs whenever there is a change to router state or the url.





Unpublished work. © 2025 Siemens

## Defining states

State definitions specify the URL routing for the set of pages and subpages that form the **Solution**. This controls how a given URL fragment is associated to a given page or subpage's **view** and **view model** definition.

In Siemens Web Framework, each page and subpage can be defined a unique state. These states get stored in a `states.json` file on a per module basis. Each `states.json` file is located in the folder structure as a peer to the `module.json`. When constructing a **site** the Siemens Web Framework tooling pulls together the state definitions.

Let's understand States in more details by looking at `states.json` file from one of sample application built using Siemens Web Framework. There are currently two subpages defined in the `states.json` file: one for the "curiosityPhotos" and the other for the "marsMap". i The states are defined using JSON configuration. For each subpage its parent state, url fragment, localized text for the subpage display and the declarative page to load are defined. The "view" attribute specifies the declarative **View** and **View Model** to demand load (so for example, line 50, this will demand load marsMapView.html and marsMapViewModel.json).

A combination of the "parent" and "url" fields mean that the respective subpages will have the following URLs `<machine:port>/apollo-sample/#/marsPage/curiosityPhotos` and `<machine:port>/apollo-sample/#/marsPage/marsView` respectively. When the user navigates to those URLs the specified **View** and **View Model** (specified by the "view" attribute) for the page definition will be demand loaded.

> **Note**
>
> Although not enforced at framework level, it is recommended that the state `name` (e.g. `marsPage`) matches the corresponding state `url` (e.g. `/marsPage`). If the name and `url` do not match, currently the **Back** command included by default in the common frame may not work as expected in certain situations.

## Params in state

SWF supports both `query` parameters and `path` parameters. By default, a parameter is treated as a query parameter and is appended to the URL after the `?`.

If you want a parameter to be part of the URL path, you need to define it as a path parameter, as shown below:

```
    "doc": {
        "url": "/doc/:id/:anchor",
        "params": {
            "id": {
                "type": "path",
                "dynamic": true,
                "value": "home"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            },
            "anchor": {
                "type": "path",
                "dynamic": true,
                "value": ""
            },
            "package": "",
            "q": ""
        },
        "parent": "doc-viewer",
        "view": "AwStandardSublocation"
    },
```

In the example above, `id` and `anchor` are path parameters, while `package` and `q` are query parameters. The URLs would be formed as follows:

Example 1: `id` and `anchor` are part of URL path.

```
http://swf/documentation/#/doc/<id>/<anchor>
http://swf/documentation/#/doc/workspaces/toc_h2__Default_Workspace_concept
```

Example 2: `id` is part of URL path and `q` is appened at end after ?

```
http://swf/documentation/#/doc/<id>/?q=<value>
http://swf/documentation/#/doc/home/?q=states
```

## Declarative State Resolve

Declarative state resolve configuration provides the way to execute declarative actions after state transition started and before the state transition completed.

The main benefits are:

- To get the data from server before page load.
- In page refresh (F5) use case.

To define resolve actions, tou need to define the actions in "resolveActions" map inside the states.json.

Example:

```
"sampleState": {
   "resolveActions": {
       "getData": {
           "actionType": "JSFunctionAsync",
           "method": "getDataBeforeInitializeTheController",
```

```
            "inputData": {
                "timeout": 3000
            },
            "outputData": {
                "ctx.myData": "result.data"
            },
            "deps": "js/getDataService"
        }
    }
}
```

Here getData is declarative action it will execute the getDataBeforeInitializeTheController method from getDataService.js and this method will return the object contains data property. and using outputData we can set this data on ctx.

getData action will execute after the state transition started and before initialize the state controller and state transition completed. you can define the multiple actions inside the resolveActions.

The following action types are supported:

- TcSoaService
- RESTService
- JSFunction
- JSFunctionAsync
- Custom Actions

## How to reload a state

`AwStateService` provides a method to reload the current state:

```
AwStateService.instance.reload();
```

## Missing page

Siemens Web Framework will show the following message if an incorrect URL has been entered in the browser.

`Header Title` : "Missing Page" `Message` : "The requested page does not exist."

Note- If a user opens an swf-based application in a new tab as a result, The browser's back button may appear to be enabled on the initial page. In this case, clicking on the back button will display the same missing page.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Subpage Visibility

The ‘visibleWhen’ attribute of state (sub-page) definition can be used to control the visibility (or availability) of a sub-page. The conditions on visibleWhen can be defined as **simple strings or object**.

Expression Evaluation context:

- `ctx` – application context e.g.

```
"expression": "ctx.selected.type === 'aType'"
```

```
"visibleWhen": {
    "condition": {
        "expression": {
            "$source": "ctx.selected",
            "$query": {
                "$typeOf": "aType"
            }
        }
    }
}
```

- `parentState.data` – expression can include properties from parent page's data section. e.g.

```
"expression": "data.aProp === 'aValue'"
```

```
"visibleWhen": {
    "condition": {
        "expression": {
            "$source": "data.aProp",
            "$query": {
                "$eq": "aValue"
            }
        }
    }
}
```

- `parentState.params` – expression can include params from parent page's data section. e.g.

```
"expression": "params.aParam === 'aValue'"
```

```
"visibleWhen": {
    "condition": {
        "expression": {
            "$source": "params.aParam",
            "$query": {
                "$eq": "aValue"
            }
        }
    }
}
```

Example: A definition of a sub-page (state):

```
"movies": {
    "abstract": true,
    "data": {
        "headerTitle": {
        "source": "/i18n/locationAndSublocationMessages",
            "key": "moviesLocationTitle"
        },
        "viewBase": "favorites",
        "clientScopeURI": "Favorites"
    },
    "params": {
        "uid": "id01"
    },
    "view": "cfLocation",
    "parent": "cfRoot"
},
"myFavs": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "url": "/myFavs",
    "parent": "movies",
    "view": "AwStandardSublocation",
    "data": {
        "label": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "favoritesSubLocationTitle"
        },
        "nameToken": "favoritesSubLocation"
    },
    "visibleWhen": {
        "condition": {
            "expression": {
                "$and":[
                    {
                        "$source": "data.clientScopeURI",
                        "$query": {
                            "$eq": "Favorites"
                        }
                    },
                    {
                        "$source": "params.uid",
                        "$query": {
                            "$eq": "id01"
                        }
                    }]
            }
        }
    }
},
"myRecent": {
    "url": "/myRecent",
    "parent": "movies",
    "view": "AwStandardSublocation",
    "data": {
        "label": {
            "source": "/i18n/locationAndSublocationMessages",
            "key": "recentSubLocationTitle"
        },
        "nameToken": "recentSubLocation"
    },
    "visibleWhen": {
        "expression": "ctx.showRecent === true"
    }
}
```

## Page-Level Components

Siemens Web Framework provides Single Page Applications SPA. As a single page application solutions do not reload the single page, index.html, at any point, nor does control transfer to another page. Through the use of a the location hash (#) and a third party URL router library, ui-router, the perception

of navigability of logical separate pages is provided by dynamically modifying the HTML DOM for the application.

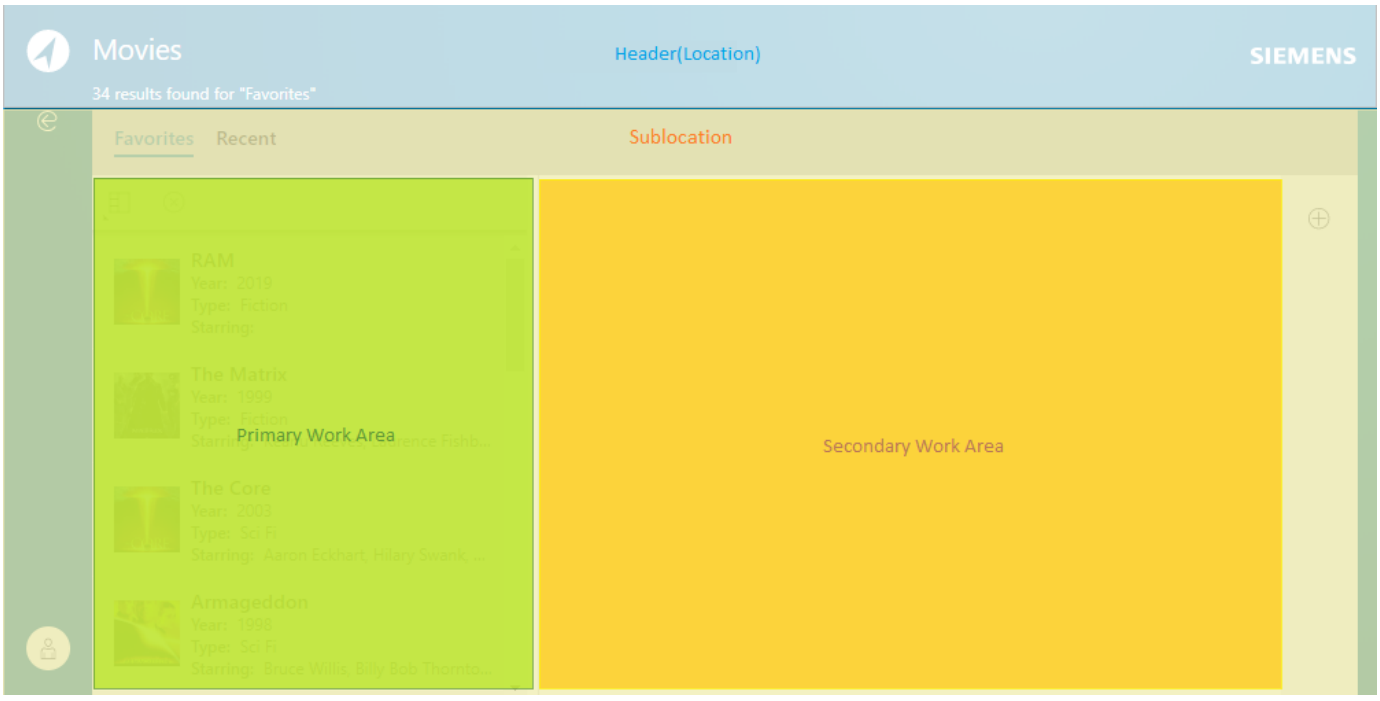
The basic page is very simple HTML5 which identifies a single element with the attribute "ui-view". The URL router library then dynamically swaps out content based on the configuration defined in the various module states.json file.

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="x-ua-compatible" content="IE=11">
        <link rel="stylesheet" type="text/css" href="assets/main.css">
    </head>

    <body>
        <div ui-view id="main-view" class="aw-layout-mainView"></div>
    </body>
</html>
```

## Basic Page Layout

The basic page layout using the framework would look like this:



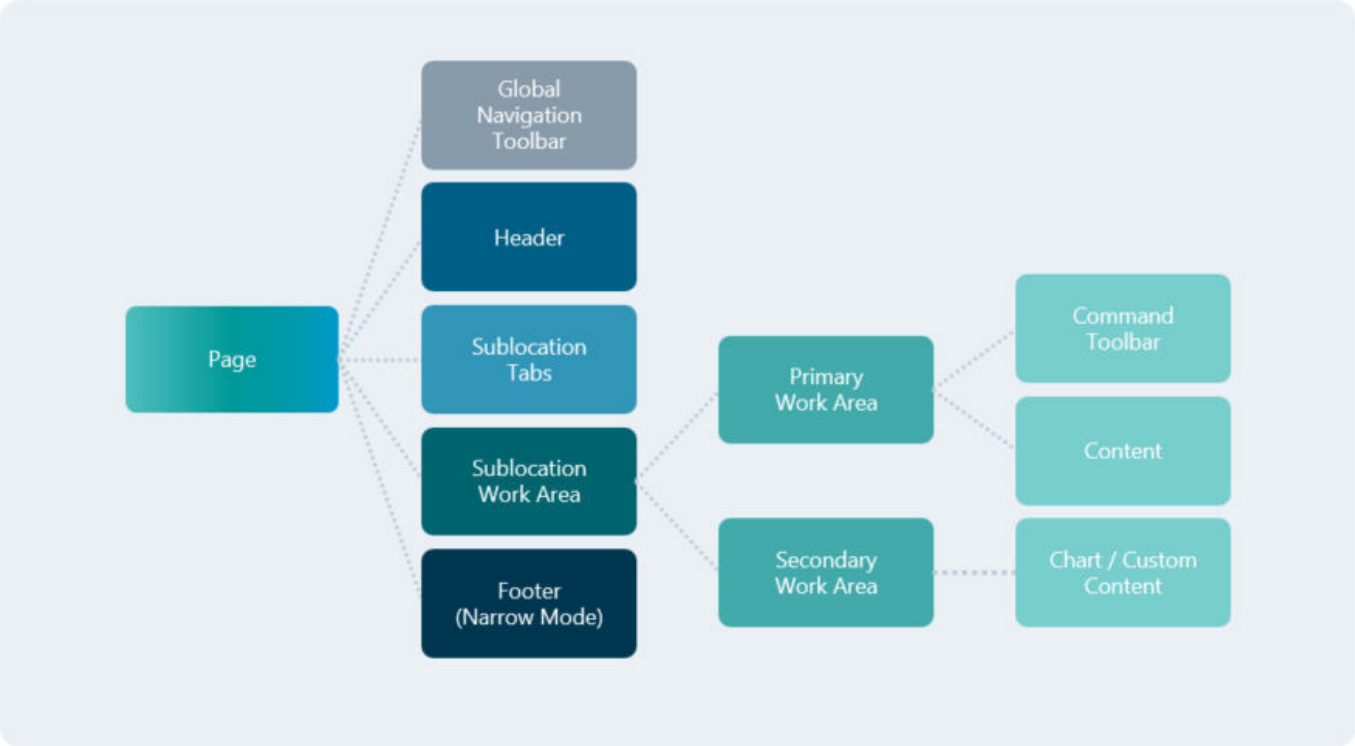

More specifically:

- A **Location** can be seen as a container grouping together one or more Sublocations, accessible via tabs.
- A **Sublocation** has a **Primary Work Area** and a **Secondary Work Area**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- A Sublocation provides three main command bars:
  - **Navigation** (left) — containing commands that open panels displayed on the left side of the Primary Work Area, used primarily for filtering data and generally navigating through displayed data.
  - **One Step** (middle) — containing commands that do not open panels. Note that the left side is reserved for commands that change the view mode or the way data is displayed on the screen.
  - **Tools and Information** (right) — containing commands that open panels displayed on the right side of the secondary work area, used primarily for commands that require user input or display additional information.

Or from a component decomposition view:

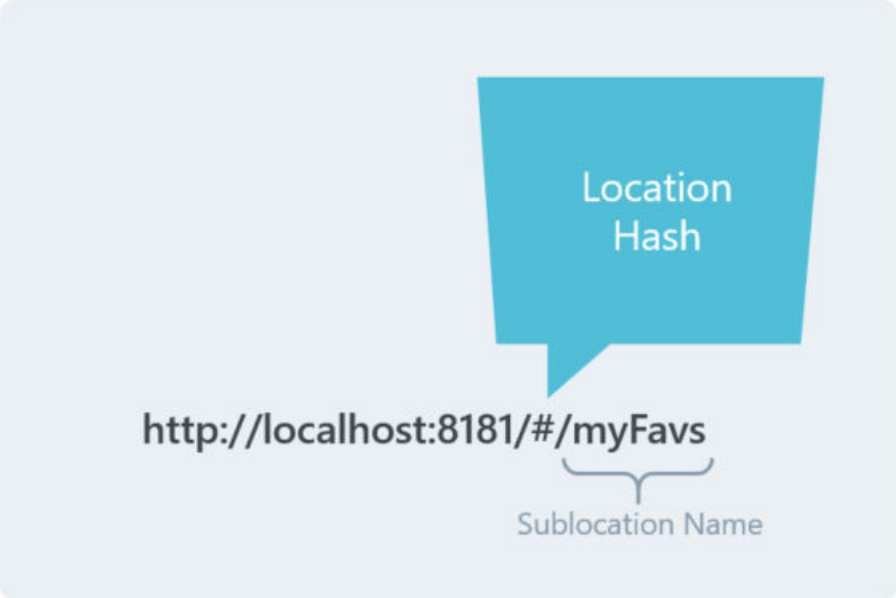



## Locations vs Sublocations

One of the layout patterns supported is the "location+sublocation" pattern. For this layout pattern, from a strictly container point of view a single location contains one or more sublocations. A location will have a title such as Movies, Inbox, Changes, Search, etc. However, in actuality a Location is just an abstract concept because a Location can not exist without at least one Sublocation and you cannot navigate directly to a location. So in real terms a Location is really nothing more than a Title plus one or more Sublocations.

While a sublocation is what the URL actually points to. So you actually navigate to a specific sublocation and not a location. In the example below the URL indicates you are navigating to the "myFavs" subLocation.


Unpublished work. © 2025 Siemens




## Configuring view from a URL

For mapping URL to content we internally use a third party library, UI-Router for helping render the UI based on anchor fragments and query parameters from the URL. UI router allow us to configure this by defining "states". A state corresponds to a "place" in the application in terms of the overall UI and navigation. Among other things with a state you can define its name, URL fragment identifier, HTML template URL to render, Angular Controller, and any ad-hoc properties you may want to associate with a state. States can be nested, by providing a parent property for a given state. That parent state will be triggered active before the child state is activated.

The user can implement Location/Sublocation paradigm by defining a state for each unique location and each unique sublocation. These states get stored in a states.json file on a per module basis. Each states.json file lives as a peer to the module.json.

The location state is provided as the parent to the sublocation state. We define a standard default HTML page template that we associate with all parent location states. This default html template can be found at /html/cfLocation.html which contains global navigation toolbar, header, sub-location tabs and sublocation content.

The row of sub-location title tabs which appears immediately below the header, is actually composed of all the sublocation states that share the common parent location state.

```
{
  "hello": {
    "abstract": true,
    "view": "cfLocation",
    "parent": "cfRoot",
    "dependencies": []
  },
  "cfRoot": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "abstract": true,
    "view": "cfRoot"
  },
  "myFavs": {
    "url": "/myFavs",
    "data": {
      "label": "Favorites",
      "nameToken": "com.siemens.splm.client.favorites:favoritesSubLocation",
      "viewBase": "favorites",
      "clientScopeURI": ""
    },
    "parent": "hello",
    "view": "AwStandardSublocation"
  }
}
```

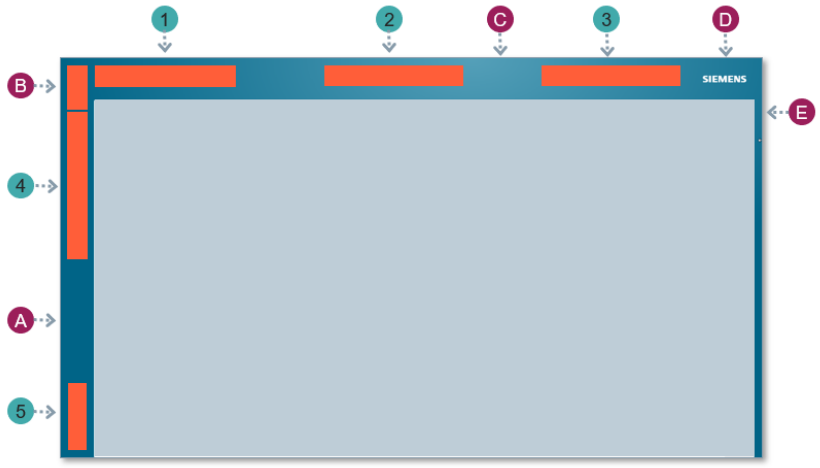
## Common Frame

Siemens Web Framework provides a Common Application Frame that is universal, consistent and configurable to enable its usage across all PL web offerings.

This frame will provide a SINGLE web header and navigation controls following the User Experience standards for web application of Siemens Cloud strategy.

Common Frame plays a vital role in maintaining the consistency of Siemens Web Framework. It provides the idea of similar structure, basic color schema and design approach. It helps in the creation of the basic structure and layout for the whole framework.

| | |
|---|---|
| A | Left Navigation area, fixed width, constant blue background*** |
| B | Global Navigation controls and app logo, common to all uses |
| C | Header area, gradient blue background, variable height from minimum value |
| D | Siemens Logo area, fixed, common to all |
| E | Right hand bar, fixed width, smaller than left (WIP) |
| 1 | Header left custom area* |
| 2 | Header centre custom area* |
| 3 | Header right custom area* |
| 4 | Main Navigation custom area** |
| 5 | Main Navigation profile and settings controls |




There are two ways in which you can add an outer frame to your application:

| Frame Type | Description | Note |
|---|---|---|
| **Common Frame with Slot Layout** | Framework also provides approved version of outer frame, also called common frame at framework level. This ensures any application using common frame is guaranteed to look consistent. Additionally there are slots provided to configure contents inside the common frame based on application needs. | From **Siemens Web Framework v3.2.0** onwards, Any future changes to common frame shall get reflected to your application automatically. It's mandatory for applications built internally to use common frame. |
| **Common Frame using the Generator** | Command line interface (CLI) utility is provided to generate files necessary to show outer frame with necessary styling. As part of this, the Root and Location view/view model files gets added to your application module. This provides you an option to start with consistent outer frame definition, but still lets you customize it as per your needs. | Any future styling changes to outer frame in framework will not be reflected. Not recommended for consumers inside Siemens network. Starting **Siemens Web Framework v5.0** this mechanism is no longer supported. |

For better understanding, try **Add Common Frame** tutioral.

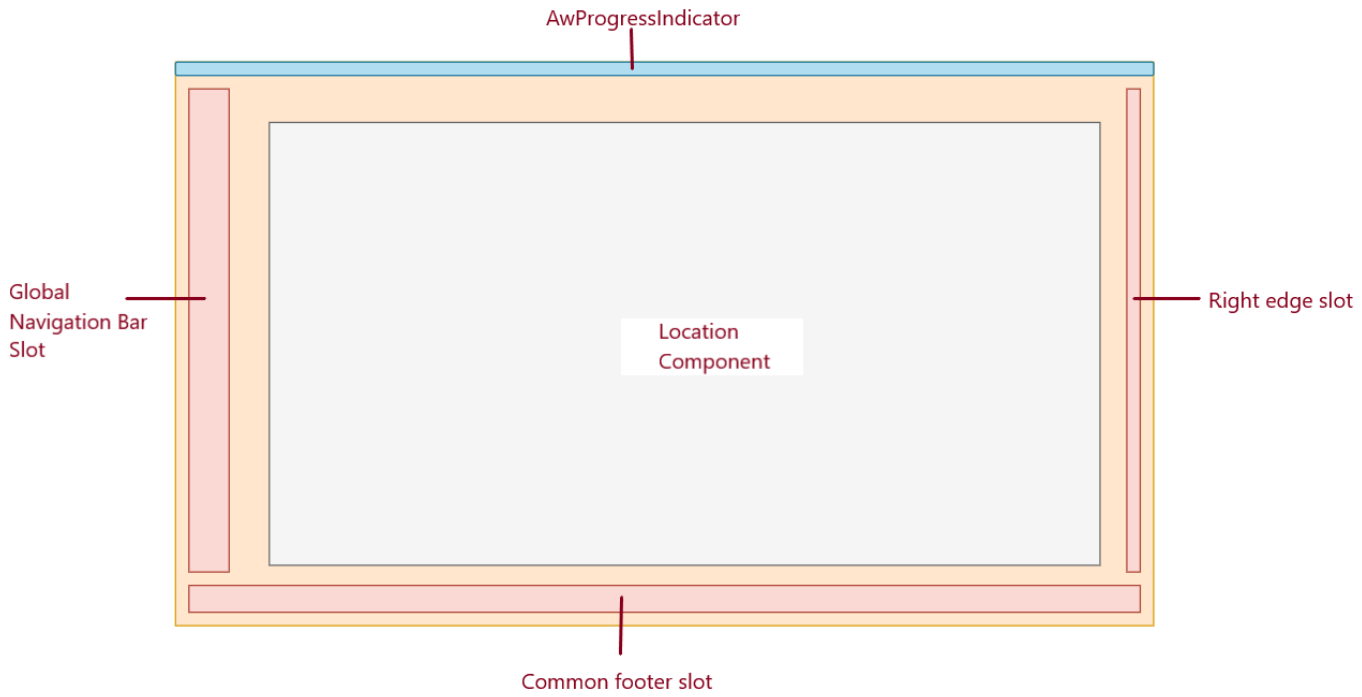
## Common Root

CommonRoot is a page level component which acts as the outermost frame of an application.
It acts as a container for the following components:

1. Location component.
2. Progress indiator.
3. Configurable Global Navigation bar.
4. Configurable RightEdge.
5. Configurable Page Footer.

Common Root has the following component hierarchy:


Siemens Web Framework
Unpublished work. © 2025 Siemens




Siemens Web Framework provides an OOTB root state definition. It can be used directly in your states.json by defining the parent for the location state as `root`. Example -

```
{
  "customLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/customConstants",
        "key": "customLocationTitle"
      }
    },
    "view": "commonLocation",
    "parent": "root"
  }
}
```

This configuration means that `commonLocation` will be rendered as a child of `commonRoot` along with progress indicator, global navigation bar, right edge and footer.

1. `Global Navigation bar`: This is the left hand navigation bar. It has a slot with name `aw_globalNavigationbar` which is configured to show `CommonGlobalNavigation` component by default. This can be **overridden** by using the **layoutSlots** configuration.
2. `RightEdge`: This is a layout slot with name `aw_rightEdge` which is configured to show `CommonRightEdge` component by default. This can be **overridden** by using the **layoutSlots** configuration.


Unpublished work. © 2025 Siemens

3. `CommonFooter`: This is the footer slot for the page.It has a layout slot with name `aw_footer` which is configured to show `CommonUserSettingsBar` component by default. This can be **overridden** by using the **layoutSlots** configuration.

## Location

A Location is a page level component which acts as a container grouping together one or more Sublocations, accessible via tabs. It is abstract which means you cannot navigate directly to it and it can not exist without at least one **Sublocation** in it.

Typically it is responsible for:

1. Providing the basic layout of the page.
2. Providing and managing the header for the page.
3. Creating and managing the sublocation tabs corresponding to each of the child states.

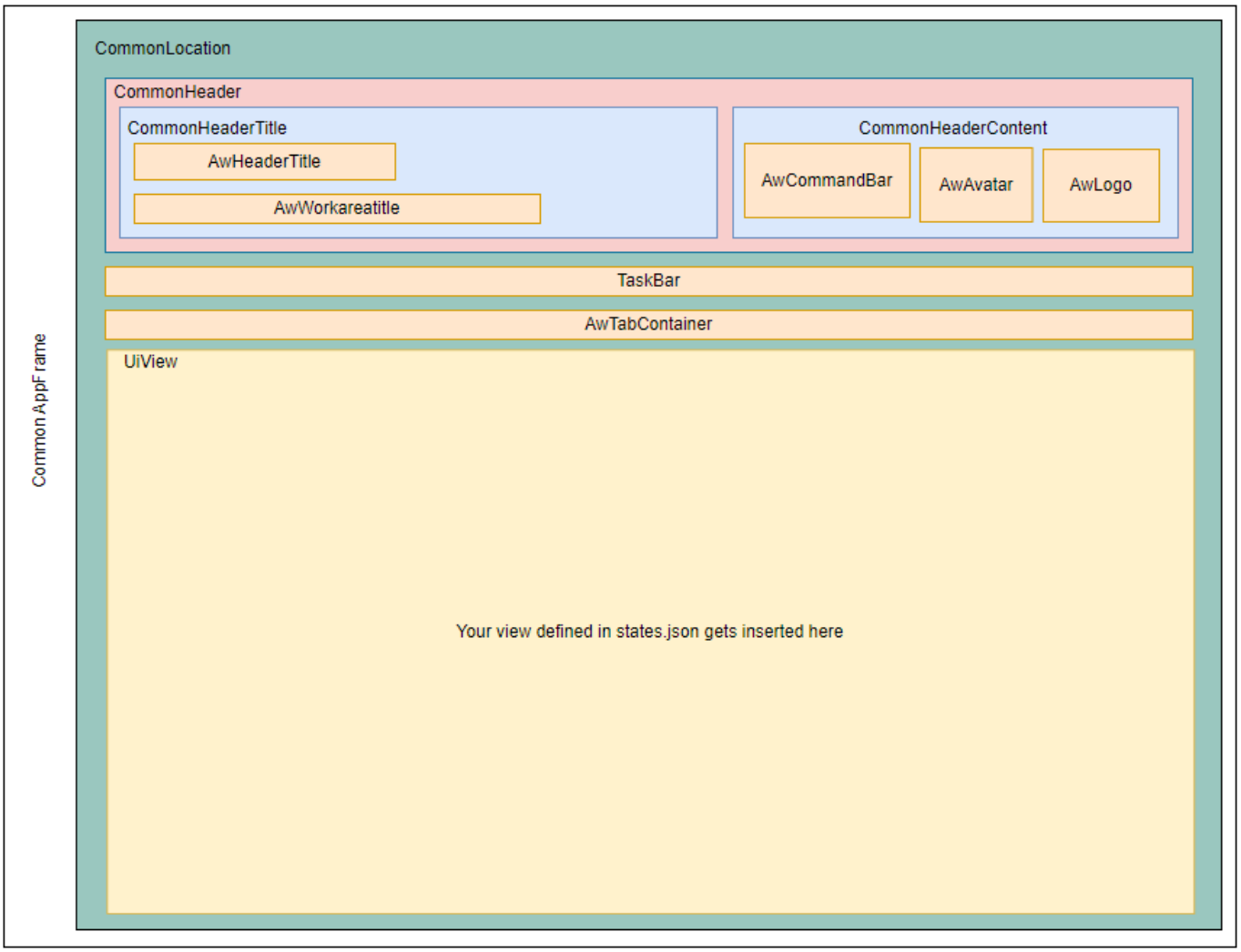
### Types of Locations

There are following location templates available in SWF or users could create their own custom flavor.

### 1. Common Location

Common Location has the following component hierarchy:


Siemens Web Framework
Unpublished work. © 2025 Siemens




It can be used directly in your states.json by defining this component in the view property of the location defined in the states.json. Example -

```
{
 "customLocation": {
   "abstract": true,
   "data": {
     "headerTitle": {
       "source": "/i18n/customConstants",
       "key": "customLocationTitle"
     }
   },
   "view": "commonLocation",
   "parent": "root"
 },
 "customSublocation": {
   "url": "/customLink",
   "data": {},
   "parent": "customLocation",
```

```
    "view": "CustomSublocationComponent"
  }
}
```

This configuration means that when `customLink` is visited; the view for `commonLocation` should be rendered - this will bring in header, taskbar, tabs and the UiView.

1. **Header**- This is the header component which brings Siemens logo by default. It also has a slot with name `aw_headerTitle` which is configured to show `CommonHeaderTitle` component by default. This can be **overridden** by using the **layoutSlots** configuration.

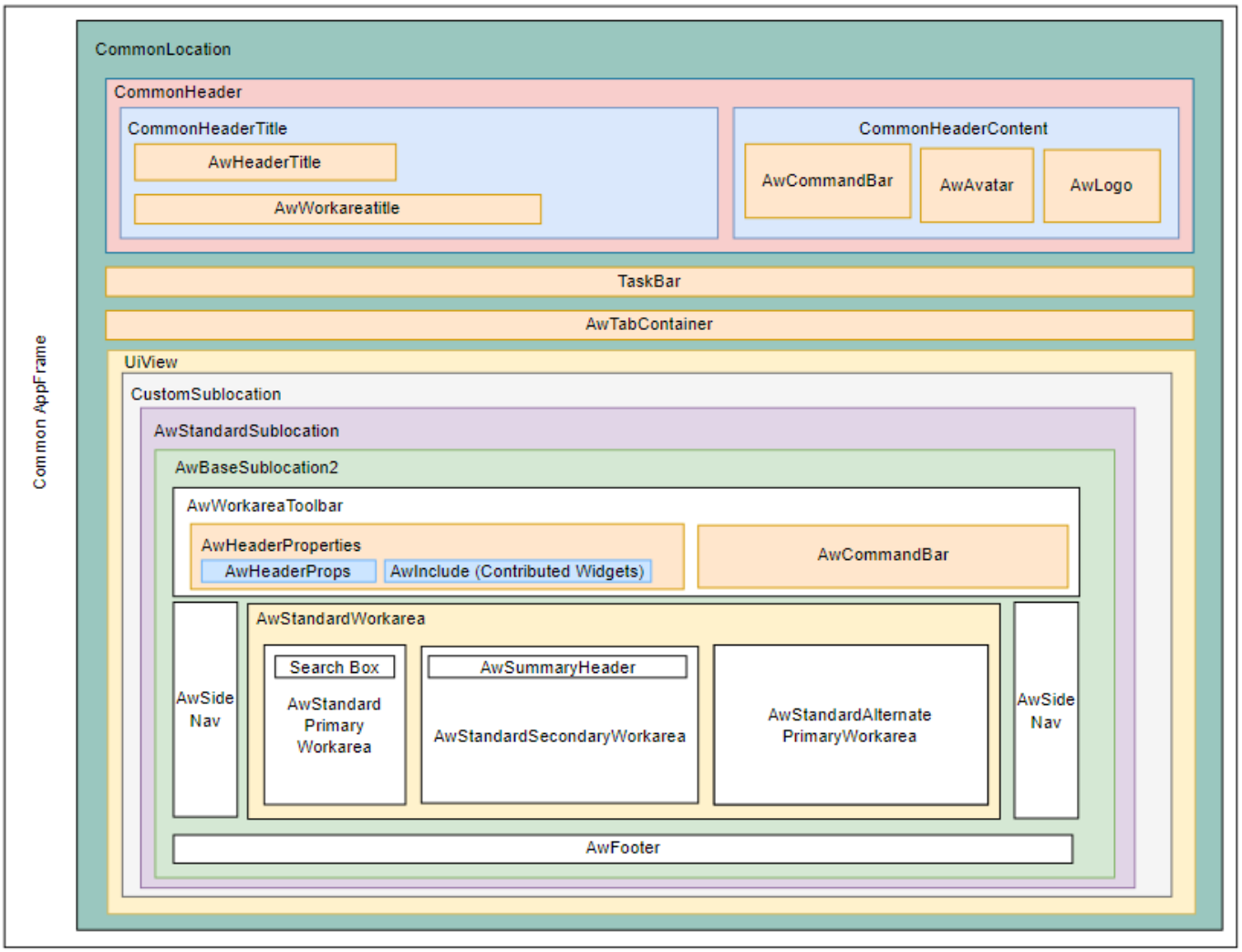
   *HeaderTitle* - This component renders the Header title, Header VMO and the Header properties. For the above example, the localized `customLocationTitle` will show up in the header section as title.

   *AwWorkareaTitle* - This component renders the breadcrumb configured via `breadcrumbConfig` property specified in the `states.json`

2. **Taskbar**- This is a layout slot with name `aw_taskBar` which is configured to show `commonTaskbar` component by default. This can be **overridden** by using the **layoutSlots** configuration.
3. **Tabs**- Tabs for each sublocation will get created. And clicking on each sublocation tab would route user to the `view` specified in the sublocation state. If there is only one tab, tab container is hidden.
4. **UiView**- the view configured in sublocation state will render here. `CustomSublocationComponent` for the example above.

For the above example, if the `CustomSublocationComponent` was using **AwStandardSublocation** component; this is how the component hierarchy of the complete page would look like.


Siemens Web Framework
Unpublished work. © 2025 Siemens

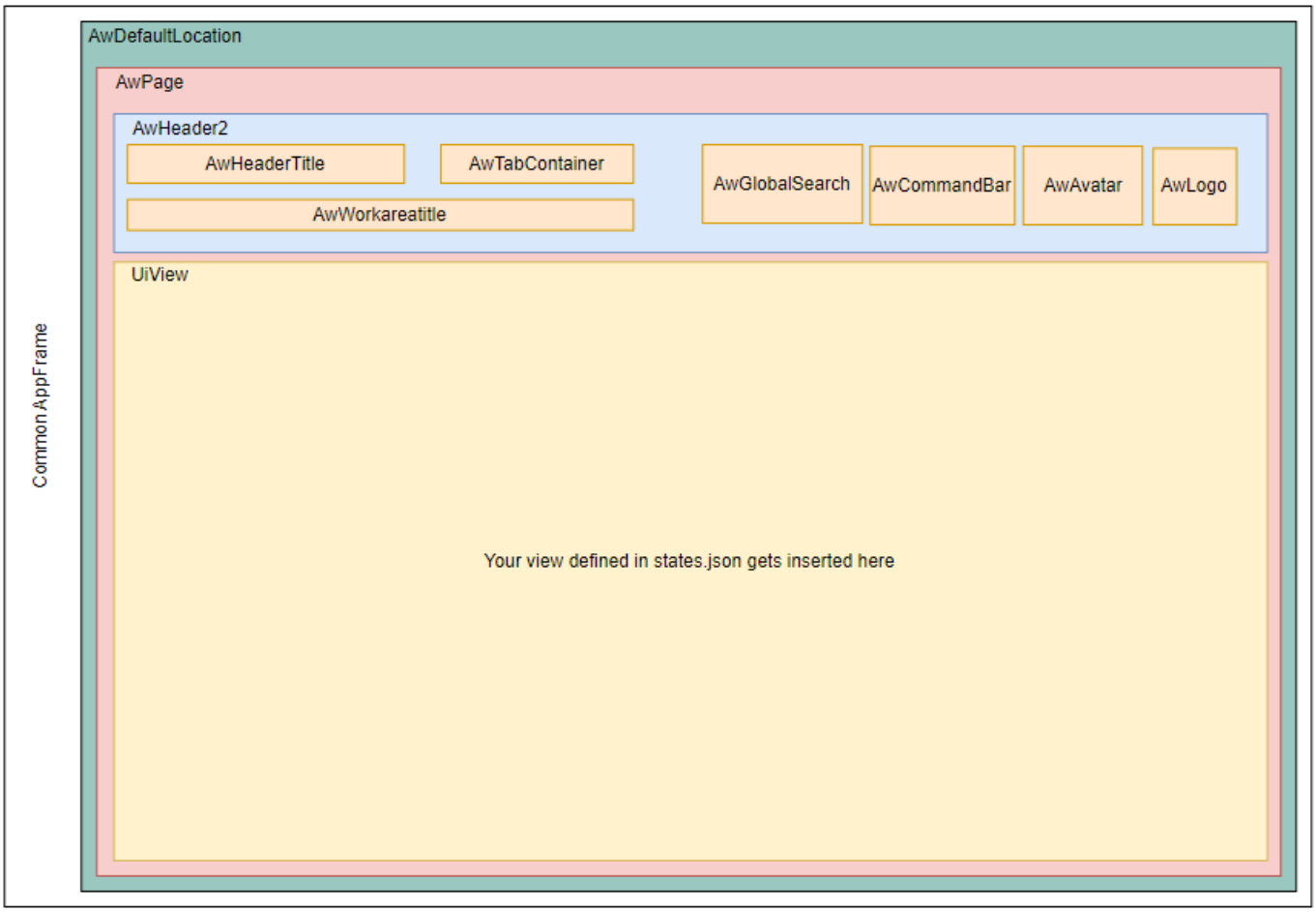



## Props sent down in the component hierarchy

| Output Prop | Description |
|---|---|
| subPanelContext.pageContext.primaryActiveTabId | The active sublocation tab |
| subPanelContext.pageContext.primarySublocTabState | The active sublocation tab state which can be updated to indicate CommonLocation to switch tab programatically |

## 2. Default Location

AwDefaultLocation has the following component hierarchy:




It can be used directly in your `states.json` by defining this component in the view property of the location defined in the `states.json`. Example -

```
{
  "customLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/customConstants",
        "key": "customLocationTitle"
      }
    },
    "view": "AwDefaultLocation",
    "parent": "root"
  },
  "customSublocation": {
    "url": "/customLink",
    "data": {},
    "parent": "customLocation",
    "view": "CustomSublocationComponent"
  }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

**OR**

It can also be configured by simply specifying `type` as `location` in the states.json. Example -

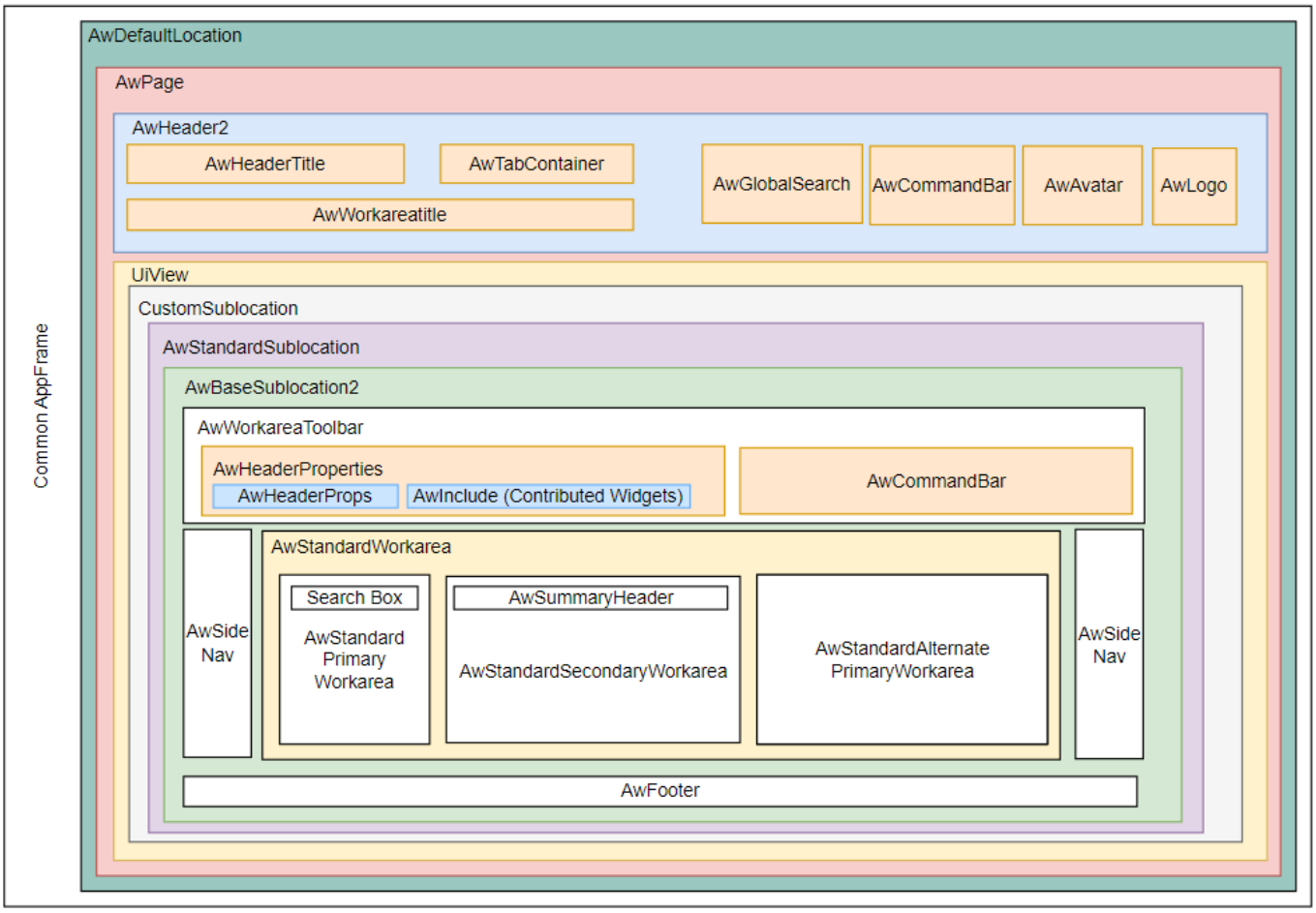
```
{
  "customLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/customConstants",
        "key": "customLocationTitle"
      }
    },
    "type": "location",
    "parent": "root"
  },
  "customSublocation": {
    "url": "/customLink",
    "data": {},
    "parent": "customLocation",
    "view": "CustomSublocationComponent"
  }
}
```

This configuration means that when `customLink` is visited; the view for `AwDefaultLocation` should be rendered - this will bring in header, taskbar, tabs and the UiView.

1. **Header**- This is the header component which brings Siemens logo by default. *HeaderTitle* - This component renders the Header title, Header VMO and the Header properties. For the above example, the localized `customLocationTitle` will show up in the header section as title. *AwTabContainer* - This component renders the tabs for each sublocation. And clicking on each sublocation tab would route user to the `view` specified in the sublocation state. *AwWorkareaTitle* - This component renders the breadcrumb configured via `breadcrumbConfig` property specified in the states.json
2. **UiView**- the view configured in sublocation state will render here. `CustomSublocationComponent` for the example above.

For the above example, if the `CustomSublocationComponent` was using **AwStandardSublocation** component; this is how the component hierarchy of the complete page would look like.


Unpublished work. © 2025 Siemens




## Props sent down in the component hierarchy

| Output Prop | Description |
|---|---|
| subPanelContext.pageContext.primaryActiveTabId | The active sublocation tab |
| subPanelContext.pageContext.primarySublocTabState | The active sublocation tab state which can be updated to indicate AwDefaultLocation to switch tab programatically |

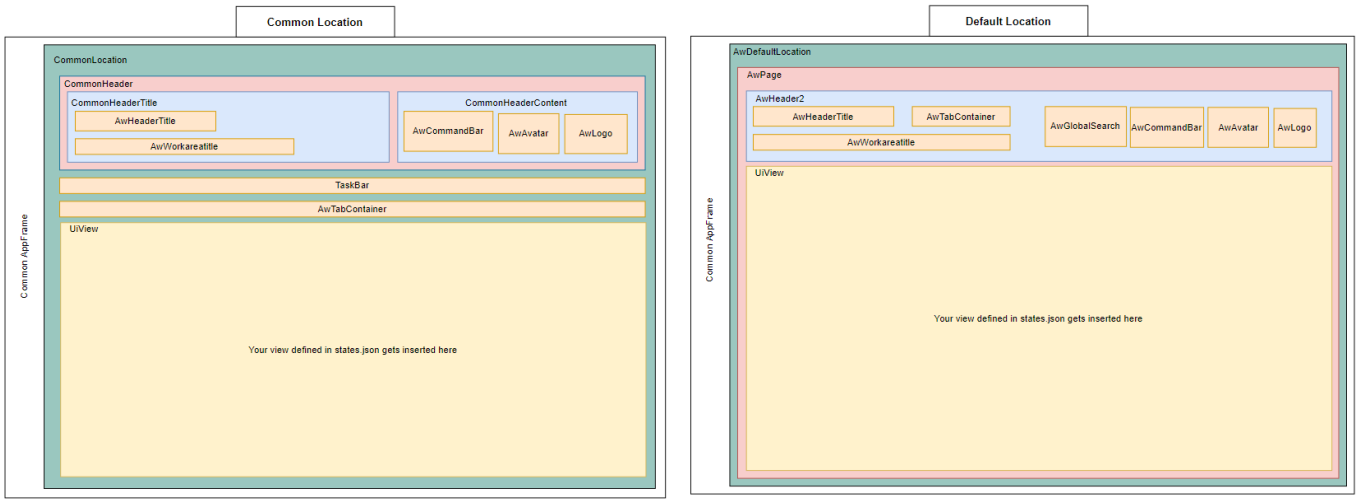
## 3. Custom Location

Any custom view can be used as a location which can be constructed as per the business requirements.

## Side by Side view of CommonLocation and AwDefaultLocation

The main difference between these templates is in the components being used and the way they are laid out. Here is a side by side difference between CommonLocation and AwDefaultLocation.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Sublocation

A Sublocation is a page-level component which brings in the actual content of the page which should render when the URL specified in the sublocation definition in `states.json` is hit. Every sublocation is rendered as a tab inside its parent **location** component and switching these tabs changes the sublocation view that is rendered.

Typically a sublocation is responsible for:

- providing the basic layout of the content of the page
- managing the interaction between a workarea component and sidenavs and the `aw_workarea_toolbar` command bar.

### Configuring a Sublocation

By default, if you set the `view` of your state configuration to `AwStandardSublocation`, the **AwStandardSublocation** is used as sublocation component for that specific state.

In this case, the sublocation component can be configured by specifying properties in the `data` section of your state configuration, like this:

```
{
  "customLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/customConstants",
        "key": "customLocationTitle"
      }
    },
    "view": "commonlocation",
```

```
    "parent": "root"
  },
  "customSublocation": {
    "url": "/customLink",
    "data": {
      "nameToken": "SelectionExample",
      "viewBase": "SelectionExample",
      "defaultDisplayMode": "SummaryView",
      "viewModes": {
        "ListView": {
          "primaryWorkArea": "list",
          "secondaryWorkArea": false
        },
        "SummaryView": {
          "primaryWorkArea": "list",
          "secondaryWorkArea": true
        }
      }
    },
    "parent": "customLocation",
    "view": "AwStandardSublocation"
  }
}
```

While this method can be used for standard scenarios, as of version 5 of Siemens Web Framework it is possible to define your own custom components to act as sublocation component, or wrap existing sublocation component. This approach is preferred in case you need a higher degree of customization.

In this case, specify the component you want to use a sublocation using the `view` property of your state configuration, like this:

```
{
  "customLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/customConstants",
        "key": "customLocationTitle"
      }
    },
    "view": "commonLocation",
    "parent": "root"
  },
  "customSublocation": {
    "url": "/customLink",
    "data": {},
    "parent": "customLocation",
    "view": "CustomSublocationComponent"
  }
}
```



Unpublished work. © 2025 Siemens

The `CustomSublocationComponent` can then be defined in terms of a View, which can wrap one of the existing sublocation components, like this:

```
<aw-standard-sublocation provider="data._sublocation" sub-panel-
context="data._sublocation"></aw-default-sublocation>
```

and data can be passed down to the contained components via subPanelContext and provider, whose value is specified in the component ViewModel, like this:

```
{
    "schemaVersion" : "1.0.0",
    "imports": [],
    "data": {
        "_sublocation": {
            "nameToken": "SelectionExample",
            "viewBase": "SelectionExample",
            "defaultDisplayMode": "SummaryView",
            "viewModes": {
                "ListView": {
                    "primaryWorkArea": "list",
                    "secondaryWorkArea": false
                },
                "SummaryView": {
                    "primaryWorkArea": "list",
                    "secondaryWorkArea": true
                }
            }
        }
    }
}
```

While this approach actually requires you to create a new component to act as a sublocation, it allows further customizations, for example you can execute actions via **lifecycle hooks** and so on.

## High-level Sublocation Components

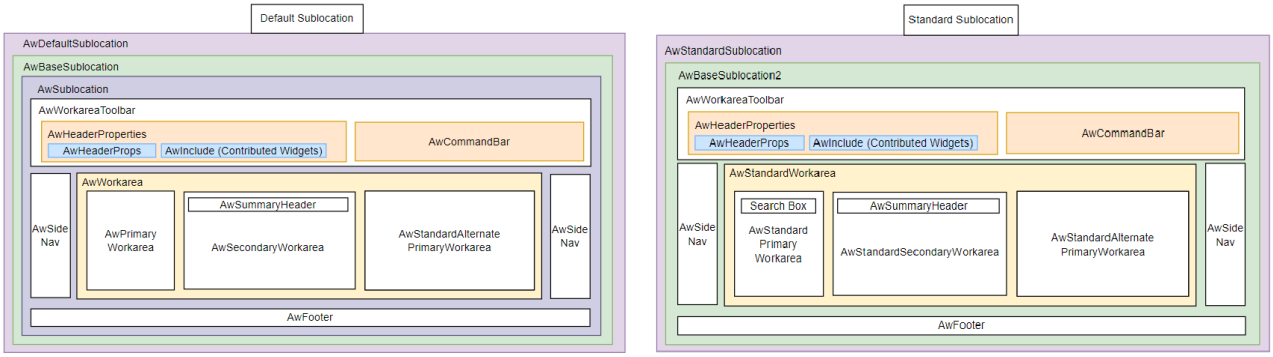
The following types of sublocations are provided by Siemens Web Framework:

- **AwDefaultSublocation** [Deprecated]
- **AwStandardSublocation**

The main difference between these templates is in the interaction model being used for communication between the sub-components. The AwStandardSublocation works off the parentState while that is not the case for AwDefaultSublocation [Deprecated]. The AwStandardSublocation also brings with itself a Search box in **AwStandardPrimaryWorkarea** which can be hidden by setting `data._provider.context.search.hideInContentSearchBox` to `true` in the state configuration..


Unpublished work. © 2025 Siemens

Here is a side by side view of AwDefaultSublocation [Deprecated] and AwStandardSublocation.




### Low-level Sublocation Components

Additionally, more low-level components are also provided to be used as sublocations:

- **AwBaseSublocation**
- **AwBaseSublocation2**
- **AwSublocation**

Note that these components are also used internally within the high-level sublocation components.

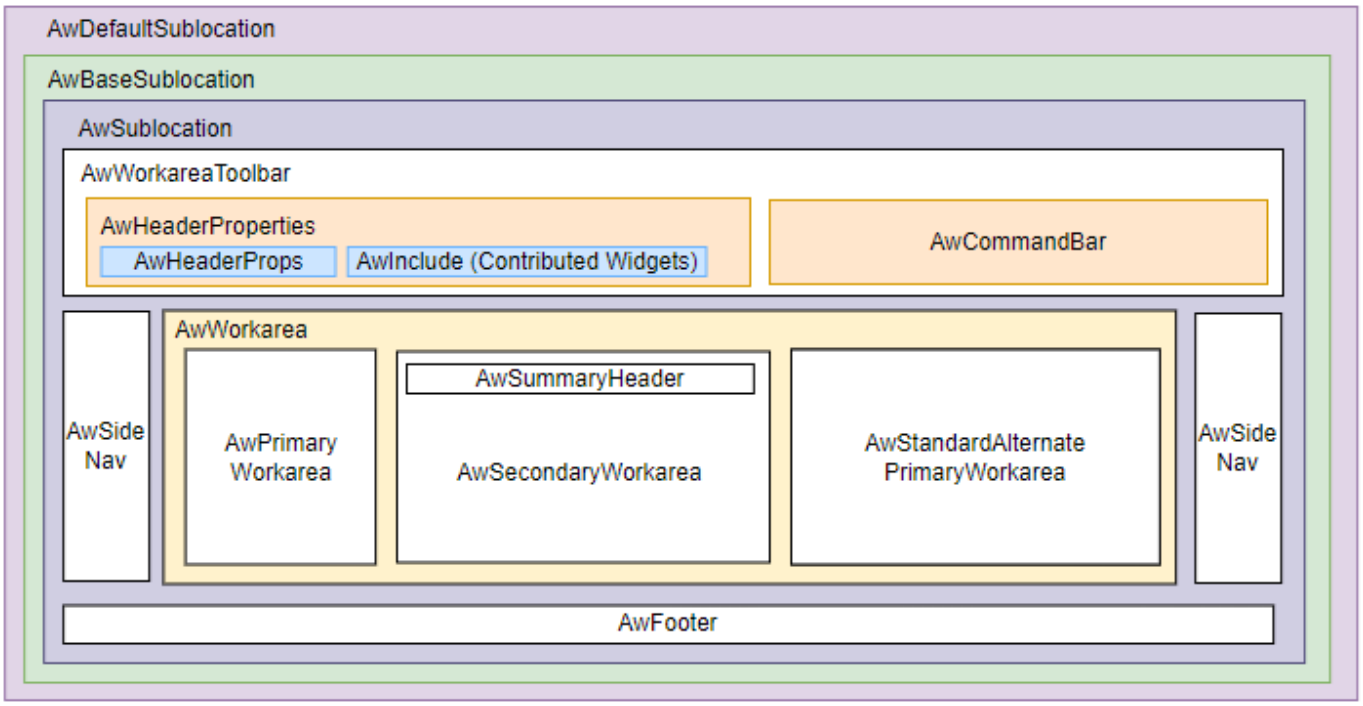
### ViewModes and ViewBase

As part of the data that can be used to customize the behavior of a sublocation, you can specify **ViewModes and ViewBase**. These properties are essential to realize common master/detail layouts with a primary and secondary work area.

## AwDefaultSublocation [Deprecated]

`AwDefaultSublocation` [Deprecated] has the following component hierarchy:


Unpublished work. © 2025 Siemens




In particular note that, the following components are used:

- **AwBaseSublocation**, which in turn consists of **AwSublocation**, which brings in both the left and right AwSideNavs, the `aw_workarea_toolbar` command bar as well as the `AwFooter` component.
- **AwWorkarea**, which in turn consists of **AwPrimaryWorkarea**, a splitter and **AwSecondaryWorkarea**.

By default, if selectionData is not overridden( not passed by parent via prop ); the sublocation would automatically synchronize the selection in URL with the PWA selection and vice versa. However, if the page does not support storing selections on URL, this would be a no-op. And if selectionData has been overridden, which basically means, page wants to manage selections on its own, then this URL and PWA selection synchronization also would need to be explicitly configured.

### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| selectionData | The selection data (additional object required for managing the selections in PWA and SWA) which is passed down to child components. This is required only when Applications wants to override default selection management. | Optional |
| selectionModel | The selection model passed in from the parent | Optional |
| baseSelection | The default or implicit selection if the sublocation supports base selection | Optional |

| | | |
|---|---|---|
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |
| subPanelContext.nameToken | The unique name to identify the sublocation | Mandatory |
| subPanelContext.viewBase | The name to be used in conjunction with viewModes to identify the views which will be shown in the Primary Workarea shown in the sublocation | Mandatory |
| subPanelContext.viewModes | The supported viewModes to be used in conjunction with viewBase to identify the views which will be shown in the Primary Workarea shown in the sublocation. If not specified, default viewModes defined in SWA (List, Table, Image) are used | Optional |
| subPanelContext.defaultDisplayMode | The default display mode from the list of supported viewModes to be launched by default. If not specified, 'default viewModes defined in SWA (List, Table, Image) are 'List with Summary' is used by default | Optional |
| subPanelContext.selectionQueryParamKey | This is URL paramater key which is used to populate selection on URL. If it is not passed then 's_uid' is default key populated by Sublocation. | Optional |

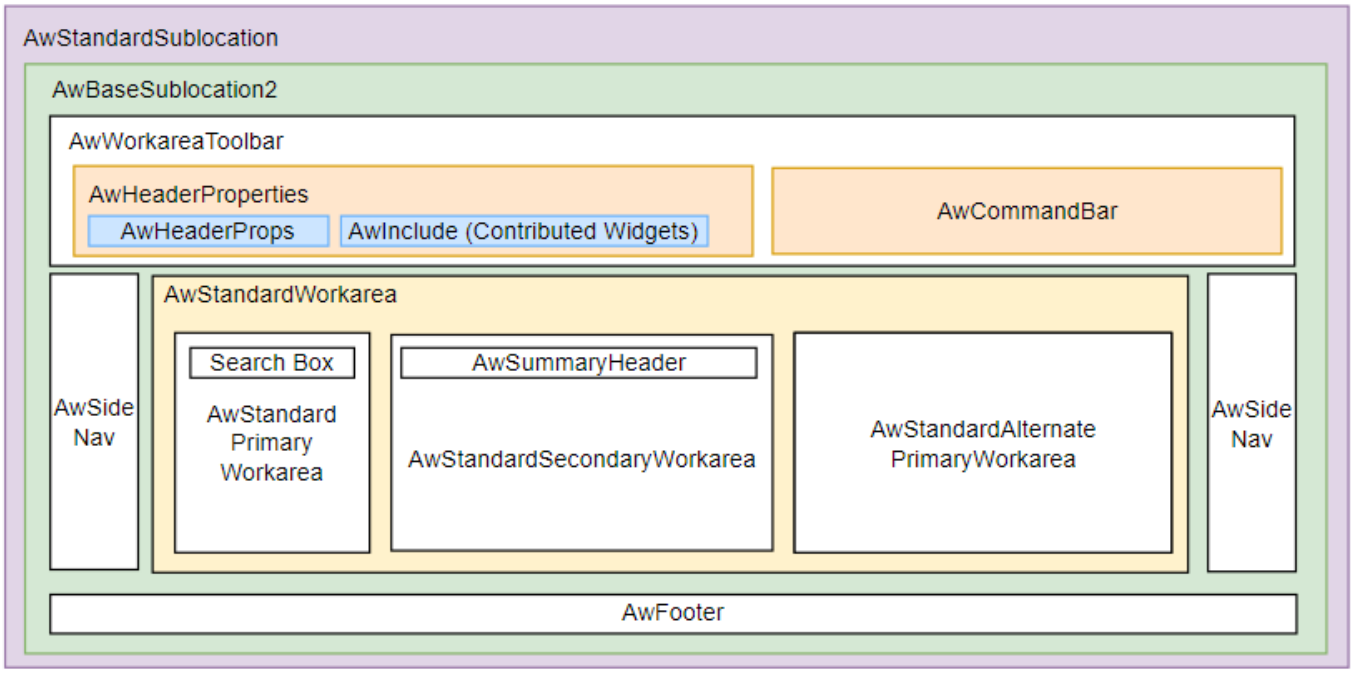
**Props sent down in the component hierarchy**

| Output Prop | Description |
|---|---|
| selectionData | The selection data (additional object required for managing the selections in Sublocation) which is passed down to child components |
| selectionModel | Parent selection model if received |
| subPanelContext | Parent subPanelContext if received and any additional data which needs to be passed down to child components |
| subPanelContext.pageContext.secondaryActiveTabId | The active tab in **SecondaryWorkarea** |
| subPanelContext.pageContext.sublocationState | The active SecondaryWorkarea tab state which can be updated to indicate SecondaryWorkarea to switch tab programatically |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## AwStandardSublocation

`AwStandardSublocation` has the following component hierarchy:




In particular note that, the following components are used:

- **AwBaseSublocation2**
- The right and left `AwSideNav` components
- the `aw_workarea_toolbar` command bar
- **AwStandardWorkarea**, which in turn consists of **AwStandardPrimaryWorkarea**, a splitter and **AwStandardSecondaryWorkarea**.

By default, if selectionData is not overridden( not passed by parent via prop ); the sublocation would automatically synchronize the selection in URL with the PWA selection and vice versa. However, if the page does not support storing selections on URL, this would be a no-op. And if selectionData has been overridden, which basically means, page wants to manage selections on its own, then this URL and PWA selection synchronization also would need to be explicitly configured.

### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| searchState | Parent State | Mandatory |
| selectionData | The selection data (additional object required for managing the selections in PWA and SWA) which is | Optional |

| | | |
|---|---|---|
| | passed down to child components. This is required only when Applications wants to override default selection management. If selectionData is sent from the parent, the parent component also needs to manage the lifecycle of searchState.pwaSelection where pwaSelection points to an array of objects selected in PWA. If selectionData is not sent by parent then searchState.pwaSelection is managed by default. | |
| selectionModel | The selection model passed in from the parent | Optional |
| baseSelection | The default or implicit selection if the sublocation supports base selection | Optional |
| provider | viewBase config and the sublocation preference config to control the layout of the page | Mandatory |
| provider.nameToken | The unique name to identify the sublocation | Mandatory |
| provider.viewBase | The name to be used in conjunction with viewModes to identify the views which will be shown in the Primary Workarea shown in the sublocation | Mandatory |
| provider.viewModes | The supported viewModes to be used in conjunction with viewBase to identify the views which will be shown in the Primary Workarea shown in the sublocation. If not specified, default viewModes defined in SWA (List, Table, Image) are used | Optional |
| provider.defaultDisplayMode | The default display mode from the list of supported viewModes to be launched by default. If not specified, 'default viewModes defined in SWA (List, Table, Image) are 'List with Summary' is used by default | Optional |
| provider.selectionQueryParamKey | This is URL paramater key which is used to populate selection on URL. If it is not passed then 's_uid' is default key populated by Sublocation. | Optional |
| provider.enableWorkareaToolbarLite | Configuring this value as 'true' will hide the AwWorkareaToolbar and enable a slot where consumers can display their custom workarea toolbar. The slot has the name 'aw_workareaToolbarLite' where consumers can configure their desired view. | Optional |
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Props sent down in the component hierarchy**

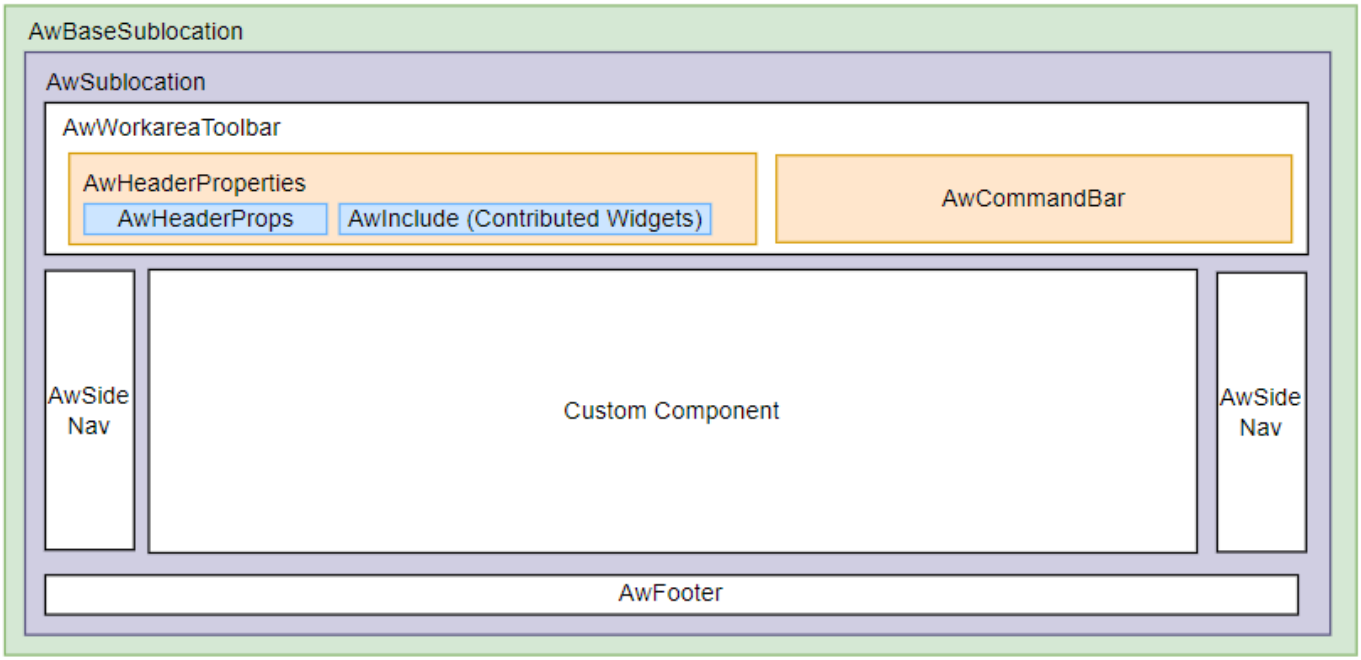
| Output Prop | Description |
|---|---|
| searchState | Parent state |
| selectionData | The selection data (additional object required for managing the selections in Sublocation) which is passed down to child components |
| selectionModel | Parent selection model if received |
| subPanelContext | Parent subPanelContext if received and any additional data which needs to be passed down to child components |
| subPanelContext.pageContext.secondaryActiveTabId | The active tab in **SecondaryWorkarea** |
| subPanelContext.pageContext.sublocationState | The active SecondaryWorkarea tab state which can be updated to indicate SecondaryWorkarea to switch tab programatically |

## AwBaseSublocation

This is a page level component which consists of AwSublocation which brings in both the AwSideNavs, Right Wall command bar as well as AwFooter. This component provides a tranclusion slot to render the contents. This component is also responsible to set the global selection if `baseSelection` provided.

AwBaseSublocation has the following component hierarchy:


Unpublished work. © 2025 Siemens




### Usage

```
<aw-base-sublocation provider="<provider>" sub-panel-
context="<additionalData>">
    <custom-component></custom-component>
</aw-base-sublocation>
```

### Props received from the parent component hierarchy

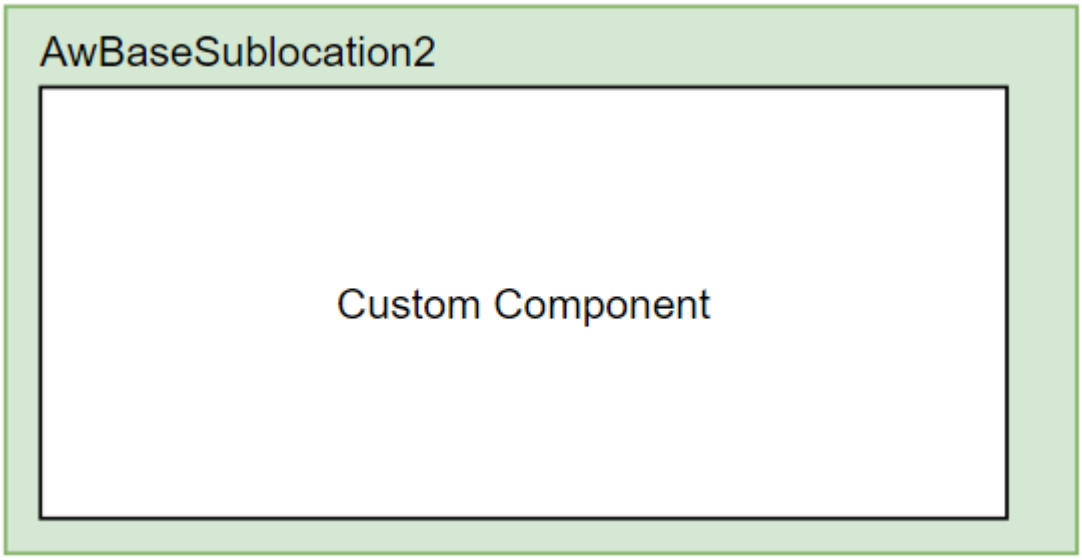
| Input Prop | Description | Mandatory |
|---|---|---|
| provider | An object consisting sublocation preference config to control the layout of the page | Optional |
| provider.hideNavigationArea | Configuring this value as `true` will hide the left sidenav | Optional |
| provider.hideRightWall | Configuring this value as `true` will hide the right sidenav | Optional |
| provider.hideWorkareaToolbar | Configuring this value as `true` will hide the work area toolbar | Optional |
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |
| baseSelection | The default or implicit selection if the sublocation supports base selection | Optional |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## AwBaseSublocation2

This is a page level component providing complete flexibility to the consumers to structure the component layout. It is also responsible to set the global selection if `baseSelection` provided.

AwBaseSublocation2 has the following component hierarchy:



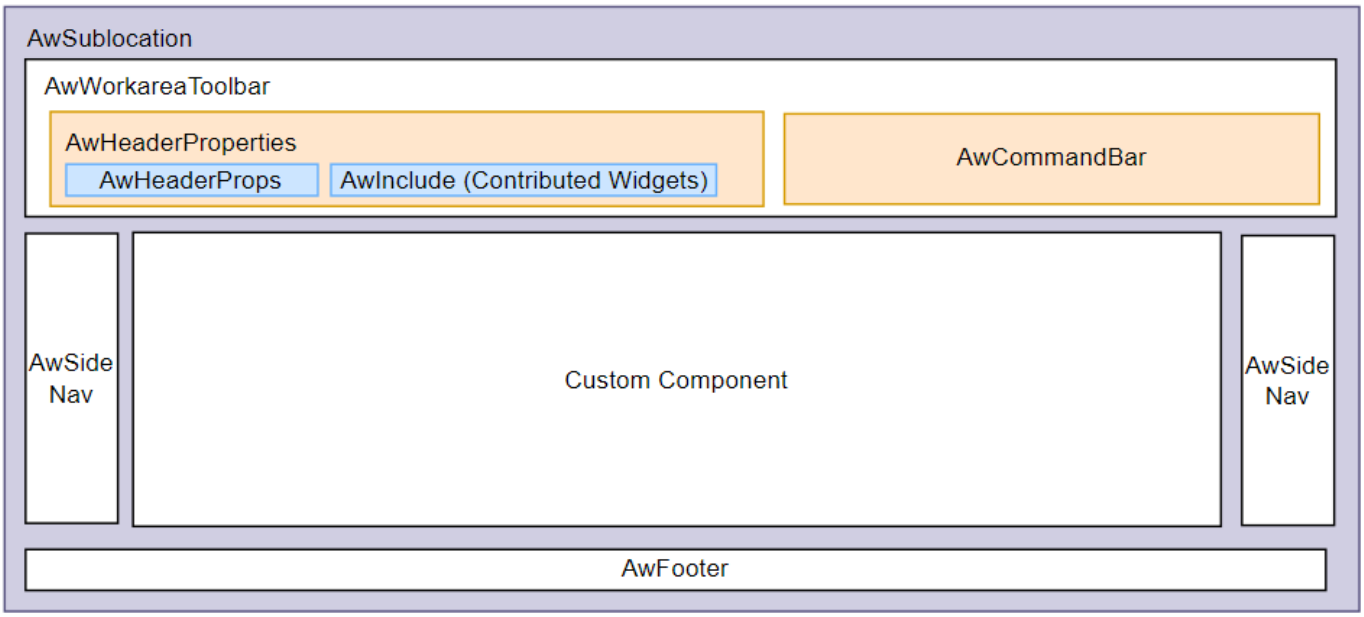

| Input Prop | Description | Mandatory |
|---|---|---|
| baseSelection | The default or implicit selection if the sublocation supports base selection | Optional |

## AwSublocation

This is a page level component which brings in both the AwSideNavs, Right Wall command bar, AwFooter and a tranclusion slot to render the contents.

AwSublocation has the following component hierarchy:


Unpublished work. © 2025 Siemens




### Usage

```
<aw-sublocation sub-location-pref="<prefs>" sub-panel-
context="<additionalData>">
    <custom-component></custom-component>
</aw-sublocation>
```

### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| subLocationPreference | A config to control the layout of the component | Optional |
| subLocationPreference.hideNavigationArea | Configuring this value as `true` will hide the left sidenav | Optional |
| subLocationPreference.hideRightWall | Configuring this value as `true` will hide the right sidenav | Optional |
| subLocationPreference.hideWorkareaToolbar | Configuring this value as `true` will hide the work area toolbar | Optional |
| subPanelContext | Any additional data which is passed to both the sidenavs and the command bar | Optional |
| subPanelContext.provider.enableWorkareaToolbarLite | Configuring this value as 'true' will hide the AwWorkareaToolbar and enable a slot where consumers can display their custom workarea | Optional |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | |
|---|---|---|
| | toolbar. The slot has the name 'aw_workareaToolbarLite' where consumers can configure their desired view. | |

## ViewModes

Sublocations can have different views and layouts to display the data. The three main view types provided in Siemens Web Framework are:

- **Primary Workarea View** (PWA)
- **Secondary Workarea View** (SWA)
- **Alternate Primary Workarea View** (AltPWA)

The **ViewTuple** is an object that contains information about current view in PWA/SWA/AltPWA. This is available on subPanelContext on PWA/SWA/AltPWA and can be used to write command condition expressions based on presence of View.

PWA View Tuple

```
{
    "pwaViewMode": "tree", // Current View in PWA
    "viewId": 0,
    "name": "Tree"
}
```

Alt PWA View Tuple

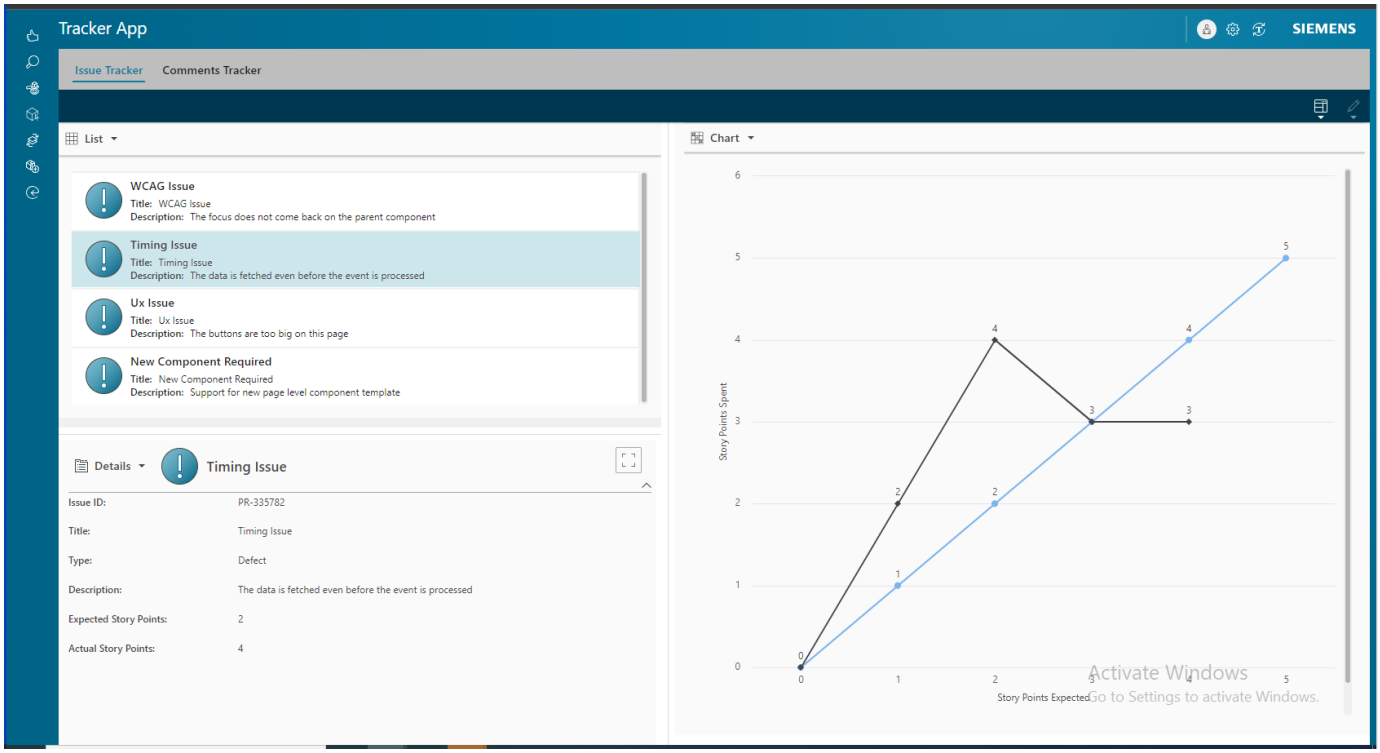
```
{
    "name": "Charts",
    "viewId": 1,
    "alternatePWA": "ChartComponent" // Current View in Alternate PWA
}
```

SWA View Tuple

```
{
    "secondaryWorkarea": true, // is SWA enabled
    "viewId": 0,
    "name": "Details"
}
```

### Layouts

Above View Types can be rendered in different layouts in sublocation. By default, SWF provides 6 types of layouts. By using those layouts data can be shown in One/Two/Three panes. Below image show Two Left One Right layout where PWA and 2 Alt PWA's are configured.




### What are the supported layouts available in SWF?

| Layout | Layout Id | Available by default |
|---|---|---|
| Single | Awp0SinglePaneLayout | Yes |
| Left Right | Awp0LeftRightLayout | Yes |
| Top Bottom | Awp0TopBottomLayout | Yes |
| Details Only | Awp0DetailsOnlyLayout | No, can be enabled by supportedLayouts |
| Two Left One Right | Awp0TwoLeftOneRightLayout | No, can be enabled by supportedLayouts |
| One Left Two Right | Awp0OneLeftTwoRightLayout | No, can be enabled by supportedLayouts |
| One Top Two Bottom | Awp0OneTopTwoBottomLayout | No, can be enabled by supportedLayouts |
| Three Vertical | Awp0ThreeVerticalLayout | No, can be enabled by supportedLayouts |

Note : These layouts are shown as Commands but those should not be overriden. These is internal and not published for override.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### How to override default layouts?

Define supportedLayouts on sublocation provider object in below format to override the default layouts. supportedLayouts array is not additive/incremental which means if you define only one layout here you will not get default plus this one but only this one.

```
"supportedLayouts": [ "Awp0TopBottomLayout", "Awp0LeftRightLayout" ]
```

### How to enable 3 view layouts(hidden by default)?

Define supportedLayouts on sublocation provider object in below format to override the default layouts.

```
"supportedLayouts": [ "Awp0SinglePaneLayout", "Awp0TopBottomLayout",
"Awp0LeftRightLayout", "Awp0TwoLeftOneRightLayout",
"Awp0OneLeftTwoRightLayout", "Awp0OneTopTwoBottomLayout",
"Awp0ThreeVerticalLayout" ]
```

### How to define default layout for a sublocation?

Define defaultLayout on sublocation provider object using **LayoutContext** type. Here, layout can be defined using row/columns keys and corresponding pwaViewMode and secondaryWorkarea value can be provided. layoutId should match predefined ID's listed above. Here you need to provide details of PWA/SWA view also.

### Default layout for Awp0SinglePaneLayout

```
"defaultLayout": {
    "layout": {
        "column": [
            {
                "row": [ { "pwaViewMode": "list" } ]
            }
        ]
    },
    "layoutId": "Awp0SinglePaneLayout"
}
```

### Default layout for Awp0LeftRightLayout

```
"defaultLayout": {
    "layout": {
        "column": [
            {
                "row": [ { "pwaViewMode": "list" } ]
            },
```

```
            {
                "row": [ { "secondaryWorkarea": true } ]
            }
        ]
    },
    "layoutId": "Awp0LeftRightLayout"
}
```

**Default layout for Awp0TopBottomLayout**

```
"defaultLayout": {
    "layout": {
        "row": [
            {
                "column": [ { "pwaViewMode": "list" } ]
            },
            {
                "column": [ { "secondaryWorkarea": true } ]
            }
        ]
    },
    "layoutId": "Awp0TopBottomLayout"
}
```

**Default layout for Awp0DetailsOnlyLayout**

```
"defaultLayout": {
    "layout": {
        "column": [
            {
                "row": [ { "secondaryWorkarea": true } ]
            }
        ]
    },
    "layoutId": "Awp0DetailsOnlyLayout"
}
```

**Default layout for Awp0TwoLeftOneRightLayout**

```
"defaultLayout":{
    "layout": {
        "column": [
            {
                "row": [
                    { "pwaViewMode": "list" },
                    { "secondaryWorkarea": true }
                ]
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            },
            {
                "row": [ { "alternatePWA": "ChartComponent" } ]
            }
        ]
    },
    "layoutId": "Awp0TwoLeftOneRightLayout"
}
```

**Default layout for Awp0OneLeftTwoRightLayout**

```
"defaultLayout": {
    "layout": {
        "column": [
            {
                "row": [ { "pwaViewMode": "list" } ]
            },
            {
                "row": [
                    { "secondaryWorkarea": true },
                    { "alternatePWA": "ChartComponent" }
                ]
            }
        ]
    },
    "layoutId": "Awp0OneLeftTwoRightLayout"
}
```

**Default layout for Awp0OneTopTwoBottomLayout**

```
"defaultLayout": {
    "layout": {
        "row": [
            {
                "column": [ { "pwaViewMode": "list" } ]
            },
            {
                "column": [
                    { "secondaryWorkarea": true },
                    { "alternatePWA": "ChartComponent" }
                ]
            }
        ]
    },
    "layoutId": "Awp0OneTopTwoBottomLayout"
}
```

**Default layout for Awp0ThreeVerticalLayout**

```
"defaultLayout": {
    "layout": {
        "column": [
            {
                "row": [ { "pwaViewMode": "list" } ]
            },
            {
                "row": [ { "secondaryWorkarea": true } ]
            },
            {
                "row": [ { "alternatePWA": "ChartComponent" } ]
            }
        ]
    },
    "layoutId": "Awp0ThreeVerticalLayout"
}
```

**How curent viewMode and Layout is retained on page refresh or back button use cases for Location/ Sublocation?**

- The name of the nameToken specified in your provider is used to identify the views/layouts unique to that page.
- If there are two or more pages sharing the same nameToken, the views/layouts will be shared between them. To avoid this, specify a unique nameToken.
- If nameToken is Not defined, a generic name is used - which again will be shared between all the pages which do not specify nameToken
- Below table shows how nameToken is used to generate unique keys for location/sublocation.

| nameToken | Key |
|---|---|
| location:subLocation | AW_SubLocation_subLocation |
| location | AW_SubLocation_location |
| undefined | AW_SubLocation_Generic |

**How to access current viewModeContext?**

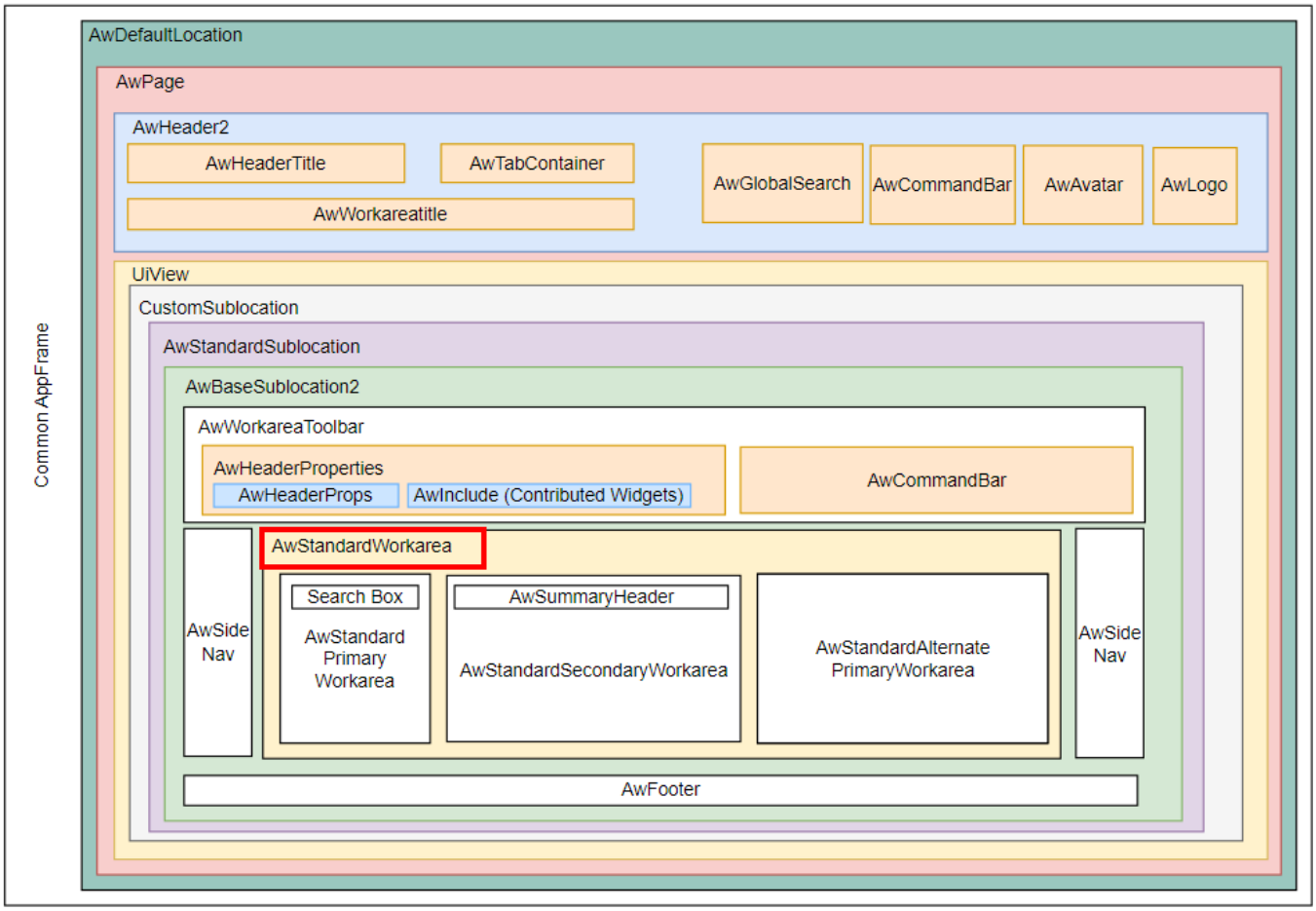
- `ViewModeContext` on `ctx` is now deprecated. It will be populated to support backword compatability based on primary and secondary workarea.
- Complete **`ViewModeContext`** will be available on subPanelContext in `pageContext.sublocationState`.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Workarea

This is a higher level component available in Fx which takes care of performing following responsibilities:

1. Manage the display modes between its child components - **PrimaryWorkarea** (PWA) **SecondaryWorkarea** (SWA) and **Alternate PrimaryWorkarea** (Alt PWA)
2. Manage the layout for its child components - PWA, SWA and Alt PWA
3. Manage the selection orchestration between its child components PWA, SWA and Alt PWA




Essentially, it is only a wrapper over PWA, SWA and Alt PWA which acts as a bigger level component ready to be re-used along with the interaction management between both these child components.

### Types of Workarea (WA)

#### 1. AwWorkarea

It is the Workarea component which consists of AwPrimaryWorkarea, Splitter and AwSecondaryWorkarea. The communication between PWA, SWA and Alt PWA in Workarea component is via Sync Strategy.

### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| subPanelContext.viewBase | The base view name which need to be rendered as per the display mode in PWA | Mandatory |
| selectionModel | The selection model passed in from the parent | Optional |
| baseSelection | The default or implicit selection if the sublocation supports base selection | Optional |
| subPanelContext | Any additional data which needs to be passed down to child components | Mandatory |
| subPanelContext.isHorizontalOrientation | The boolean to specify whether horizontal (default) or vertical layout is required between PWA and SWA | Optional |
| subPanelContext.nameToken | The unique name to identify the sublocation | Mandatory |

### Props sent down in the component hierarchy

| Output Prop | Description |
|---|---|
| selectionModel | Parent selection model if received else newly created selection model (with name `nativeSelectionModel`) |
| subPanelContext | Parent subPanelContext if received and any additional data which needs to be passed down to child components |
| selectionData | The selection data (additional object required for managing the selections in PWA and SWA) which is passed down to child components |

```
AwWorkarea should be wrapped with a flex row wrapper to get the optimal
layout.
```

### 2. AwStandardWorkarea

It is the Workarea component which consists of AwStandardPrimaryWorkarea, Splitter and AwStandardSecondaryWorkarea.

The communication between StandardPWA and StandardSWA in StandardWorkarea component is via sharing the data with the common parent i.e. the StandardWorkarea itself.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| subPanelContext.viewBase | The base view name which need to be rendered as per the display mode in PWA | Mandatory |
| searchState | Parent state | Mandatory |
| selectionModel | The selection model passed in from the parent | Optional |
| baseSelection | The default or implicit selection if the sublocation supports base selection | Optional |
| subPanelContext | Any additional data which needs to be passed down to child components | Mandatory |
| subPanelContext.nameToken | The unique name to identify the sublocation | Mandatory |
| isHorizontalOrientation | The boolean to specify whether horizontal (default) or vertical layout is required between PWA and SWA | Optional |

### Props sent down in the component hierarchy

| Output Prop | Description |
|---|---|
| searchState | Parent state |
| selectionModel | Parent selection model if received else newly created selection model (with name `nativeSelectionModel`) |
| subPanelContext | Parent subPanelContext if received and any additional data which needs to be passed down to child components |
| selectionData | The selection data (additional object required for managing the selections in PWA and SWA) which is passed down to child components. |

```
AwStandardWorkarea should be wrapped with a flex row wrapper to get the
optimal layout.
```

### Selection Management in Workarea

Now, as self contained components, each of the child components need to be capable of managing the selection changes within their component hierarchy i.e. for their child components. Additionally, they also need to update their parent component i.e. Workarea for any selection change happening within them along with additional information about the source of the selection change i.e. primary or secondary.

The Workarea component orchestrates/ does the selection management between PWA and SWA. PWA and SWA depict the Master Details pattern which means:


Unpublished work. © 2025 Siemens

- Every selection in PWA needs to be fed as an input to SWA and updated to the sublocation
- Every de-selection in PWA needs to be considered as a fall back to implicit/baseSelection i.e. the selection coming from the sublocation
- Every selection in SWA needs to be updated to the sublocation
- Every de-selection in SWA needs to be considered as a fall back to PWA selection i.e. the selection coming from PWA becomes active selection OR if nothing is selected in PWA; it further falls back to implicit/baseSelection if the sublocation supports it else it stays as no selection

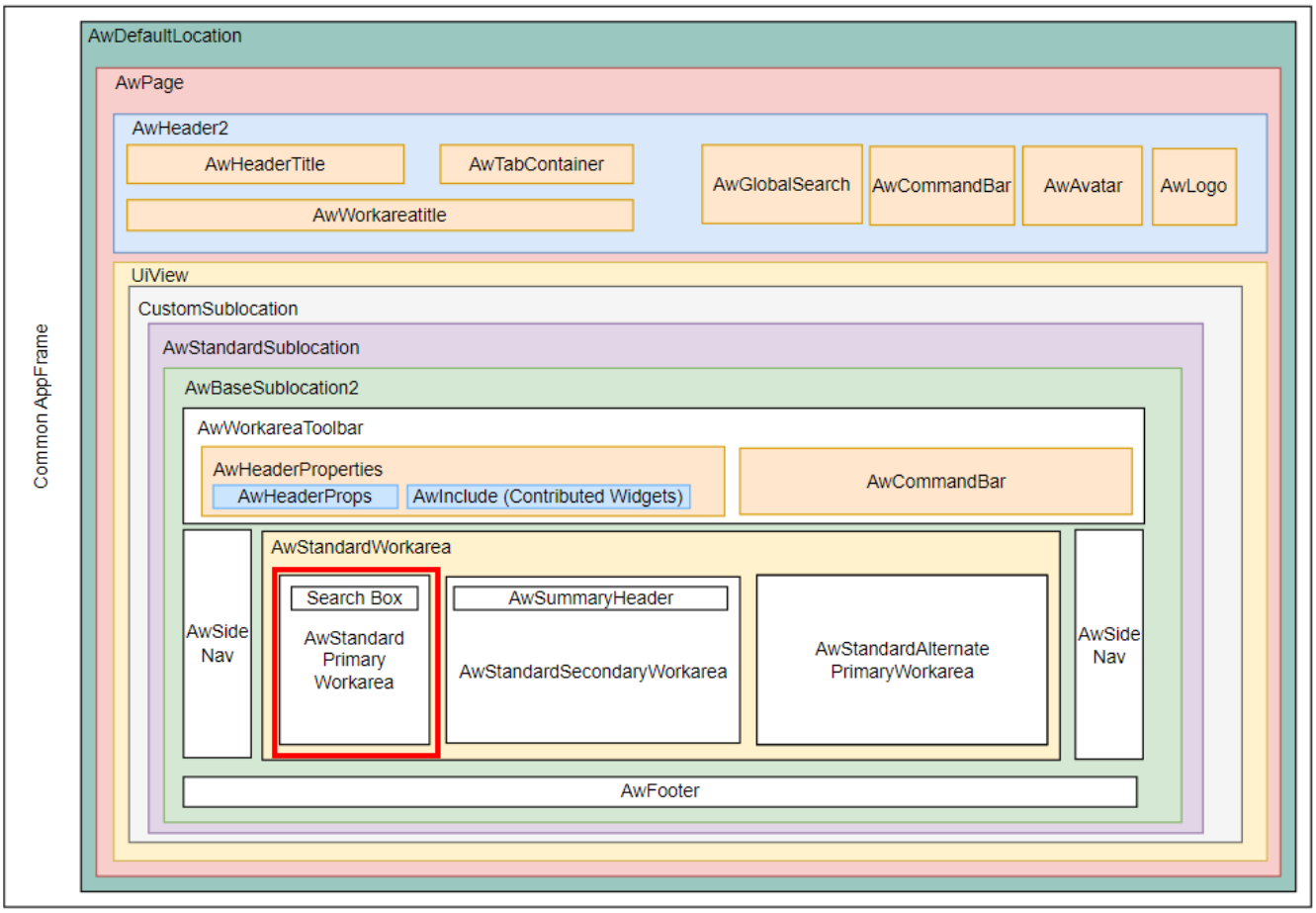
The selection management is achieved via **selectionData**

### Primary Workarea (PWA)

This is a higher-level component available in Fx which takes care of performing following responsibilities:

1. Render the Primary view configured via the supported display mode
2. Consolidate the selections coming from the child components and update them to the parent component
3. Managing the layout slot for the PWA command bar and creating the right commandContext for the commands in this command bar to work on.
4. Facilitate the in-content search to filter out the contents in PWA (AwStandardPWA only) based on the specified search criteria.


Siemens Web Framework
Unpublished work. © 2025 Siemens



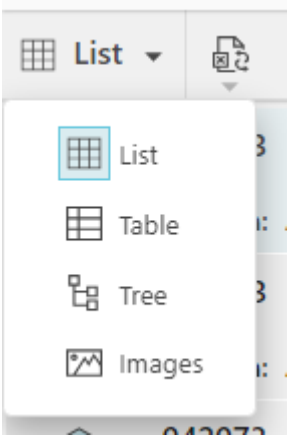

### How to Configure Primary Workarea View

### How to contribute PWA View modes?

PWA view commands can be configured by defining it on sublocation provider like below.

```
{
"pwaViewModes": [ "list", "table", "tree", "image" ]
}
```

Once you define these pwaViewModes in sublocation provider, you will start seeing PWA commands in PWA command Bar like below.




Note : If pwaViewModes are not defined on sublocation provider then by default SWF will show List, Table, and Image views.

### How to contribute PWA Views?

SWF uses viewBase defined on sublocation provider in conjunction with current view mode to determine the component to be rendered in PWA. The viewBase plays a key role here using which sublocations can define their unique PWA views.

As an example, if you specify viewBase as 'selectionExample' - the framework would use this key to determine which View and ViewModel to pick up when the display Modes are switched. So, in the List modes, the component shown up in the PWA would be `selectionExampleListViewModel`. Similarly in Table modes, the component shown up in the PWA would be `selectionExampleTableViewModel`.

```
    "viewBase": "selectionExample"
```

### How to contribute custom Primary Work Area view mode

First you need to define a custom view mode in pwaViewModes on sublocation provider object like below.

```
"pwaViewModes": [ "list", "custom" ]
```

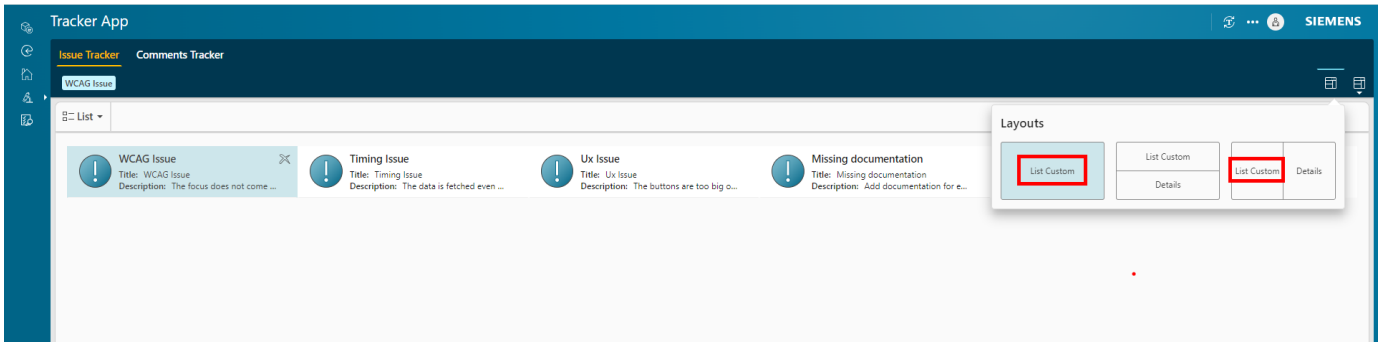
If you want to associate localized value for custom **view mode** then `pwaViewModes` can be defined in below format.

```
"pwaViewModes": [
    {
        "viewMode": "list",
        "label": {
            "source": "/i18n/testMessages",
            "key": "listTitle"
        }
    },{
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "viewMode": "custom",
        "label": {
            "source": "/i18n/testMessages",
            "key": "customTitle"
        }
    }
]
```

This value will be updated in **ViewTuple**.




Then you need to contribute view mode command in commandsViewModel.json in below format so that it appears on PWA command Bar.

```
"Awp0CustomViewDisplay": {
    "id": "Awp0CustomView",
    "priority": 80,
    "parentGroupId": "Awp0ModelObjListDisplayToggles"
}
```

Then you need to write visibleWhen expression based on below condition so that command becomes visible only when subLocation supports it.

```
"supportsAwp0CustomView": {
    "expression":
"commandContext.pageContext.sublocationState.viewModeContext.supportedViewMo
des.pwaViewModes.indexOf( 'custom' ) > -1"
}
```

Then you need to write selectWhen expression based on below condition so that command is selected when PWA has custom view mode set.

```
"isAwp0CustomView": {
    "expression": "commandContext.viewTuple.pwaViewMode === 'custom'"
}
```

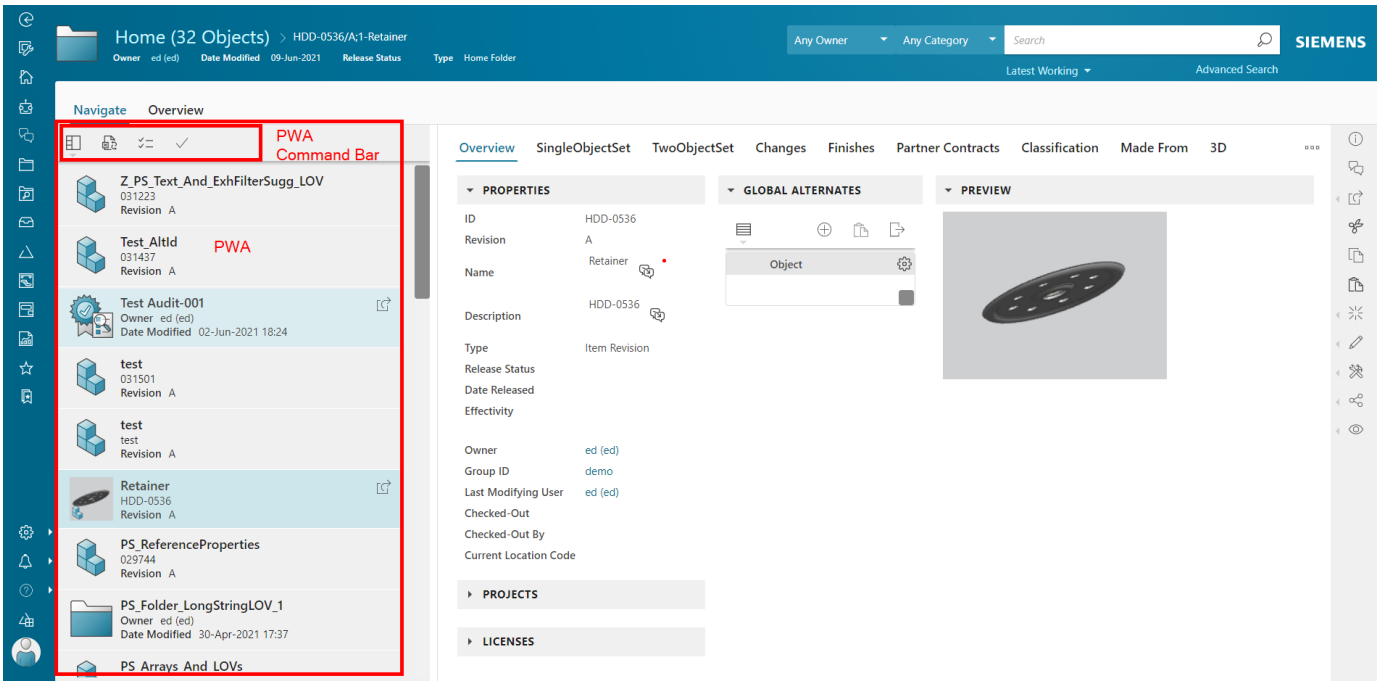
Then you need to write command action in below format so that when user selects custom view mode then respective view is rendered in PWA.

```
"setCustomView": {
    "actionType": "JSFunction",
    "method": "changeViewMode",
    "deps": "js/pageLayoutService",
    "inputData": {
        "viewTupleState": "{{commandContext.viewTuple}}",
        "viewTuple": { "pwaViewMode": "custom" }
    }
}
```

### Types of PWA

### 1. AwPrimaryWorkarea

This flavor of PWA works off Sync Strategy. The communication between PWA and SWA happens via Sync Strategy configured by default in the Fx.
`pwaSelection` - This is an output port for consumers where the information about current selections in PWA is maintained and available for them to configure a Sync Strategy on.





Siemens Web Framework
Unpublished work. © 2025 Siemens

**Props received from the parent component hierarchy**

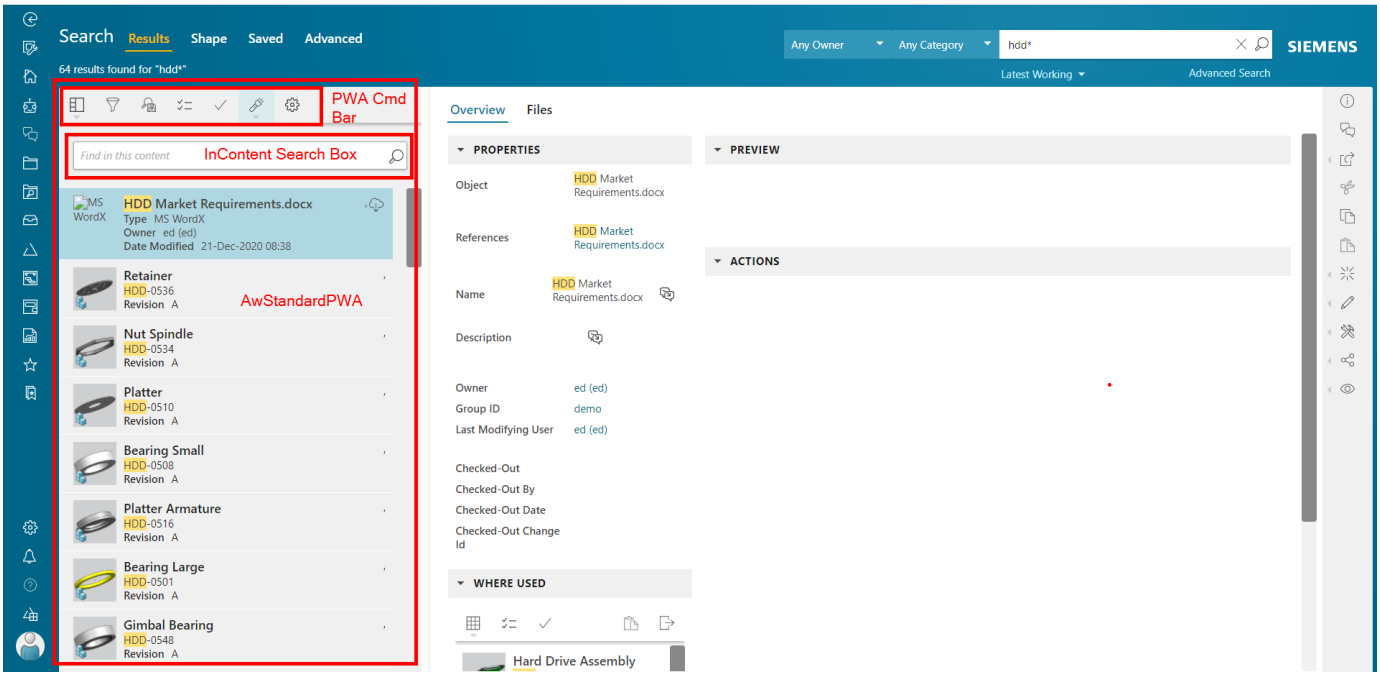
| Input Prop | Description | Mandatory |
|---|---|---|
| viewBase | The base view name which need to be rendered as per the display mode in PWA | Mandatory |
| view | The display mode name to be rendered as per the display mode in PWA | Mandatory |
| selectionModel | The selection model passed in from the parent | Mandatory |
| selectionData | The selection data (additional object required for managing the selections in PWA) which is passed down to child components | Mandatory |
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |

**Props sent down in the component hierarchy**

| Output Prop | Description |
|---|---|
| selectionModel | Parent selection model |
| subPanelContext | Parent subPanelContext if received and any additional data which needs to be passed down to child components |
| selectionData | The selection data (additional object required for managing the selections in PWA) which is passed down to child components |

**2. AwStandardPrimaryWorkarea**

This flavor of PWA works off the parent state. The communication between PWA and SWA happens via the common parent or search state as we call it. It also has an additional InContent search box which facilitates the search within the objects rendered in the PWA.




### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| searchState | Parent State | Mandatory |
| viewBase | The base view name which need to be rendered as per the display mode in PWA | Mandatory |
| view | The display mode name to be rendered as per the display mode in PWA | Mandatory |
| selectionModel | The selection model passed in from the parent | Mandatory |
| selectionData | The selection data (additional object required for managing the selections in PWA) which is passed down to child components | Mandatory |
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |

### Props sent down in the component hierarchy

| Output Prop | Description | |
|---|---|---|
| searchState | Parent State | |
| selectionModel | The selection model passed in from the parent | Mandatory |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | |
|---|---|
| subPanelContext | Parent subPanelContext if received and any additional data which needs to be passed down to child components |
| selectionData | The selection data (additional object required for managing the selections in PWA) which is passed down to child components |

### Sharing of selection model

The PWA component supports various display modes as configured by the parent component by defining the supported **viewModes** . To be able to retain the selections when the display modes are switched, PWA shares the selectionModel between its various custom components pointing to the configured display modes.

This selectionModel is created by the parent component i.e. **Workarea** if it is not received on its parent component hierarchy. This passed in selectionModel needs to be provided to the dataProvider so that a new selection model is not created rather the input selectionModel is re-used by all the various display modes configured for the PWA.

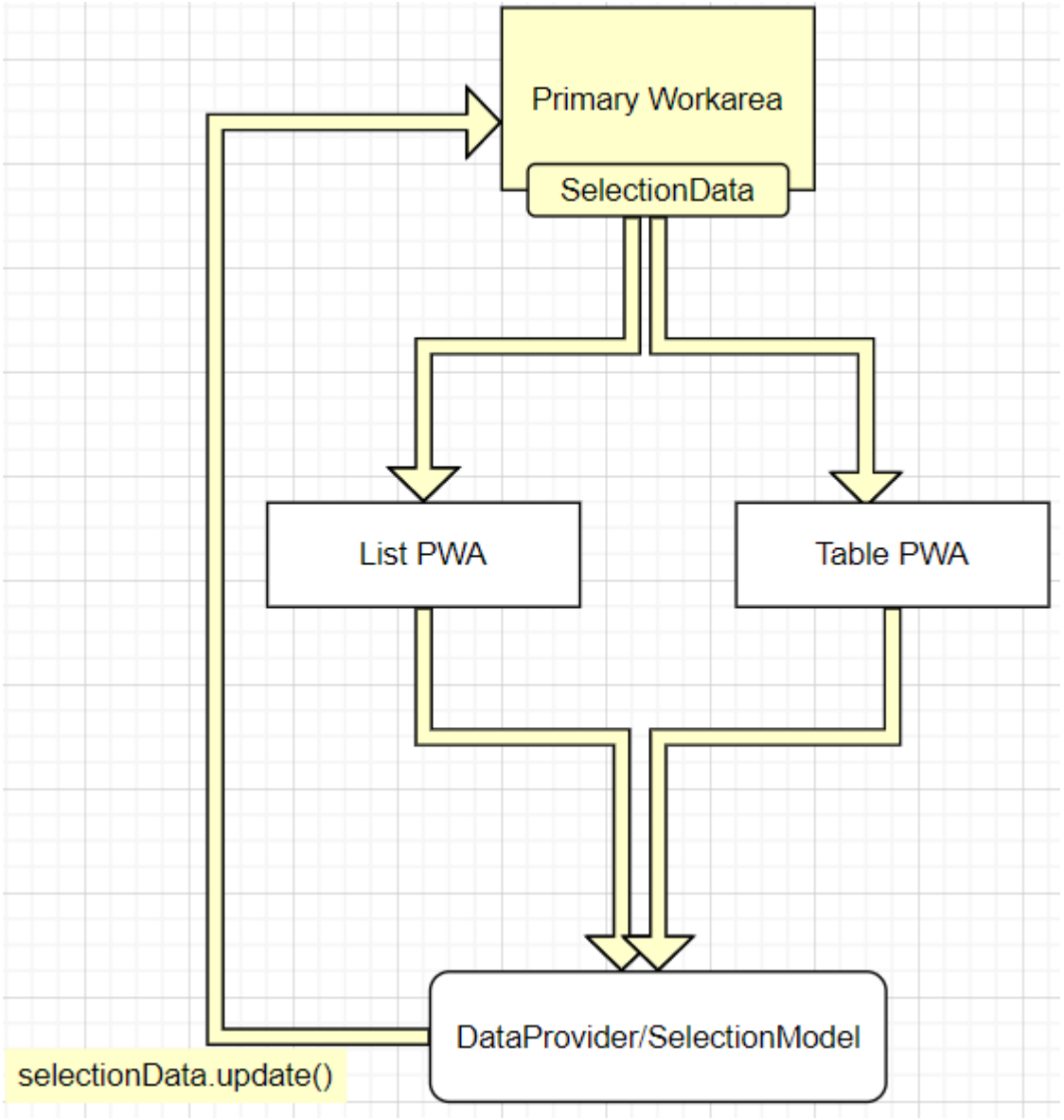
Example below demonstrates how the selectionModel received on subPanelContext needs to be passed on to the dataProvider.

```
"dataProviders": {
        "performSearch": {
            "action": "getMovies",
            "response": "{{data.searchResults}}",
            "totalFound": "{{data.totalFound}}",
            "selectionModelMode": "single",
            "inputData": {
                "selectionModel": "{{subPanelContext.selectionModel}}"
            }
        }
    }
```

### Selection Management in PWA

PWA is responsible for managing the selection within its child component hierarchy and updating the same information to the parent component hierarchy.

The selection management is achieved via **selectionData**




To manage the selection within its various child components, PWA creates its own selectionData and sends it to all its child components. It receives the update from its children and takes the responsibility of sending this information up in its hierarchy by invoking the update method available on the selectionData of its parent i.e., Workarea component. Before sending this information up, PWA appends the source as primary on the received selectionData. This information of source would be required by the parent component to orchestrate the selections properly.

### Secondary Workarea (SWA)

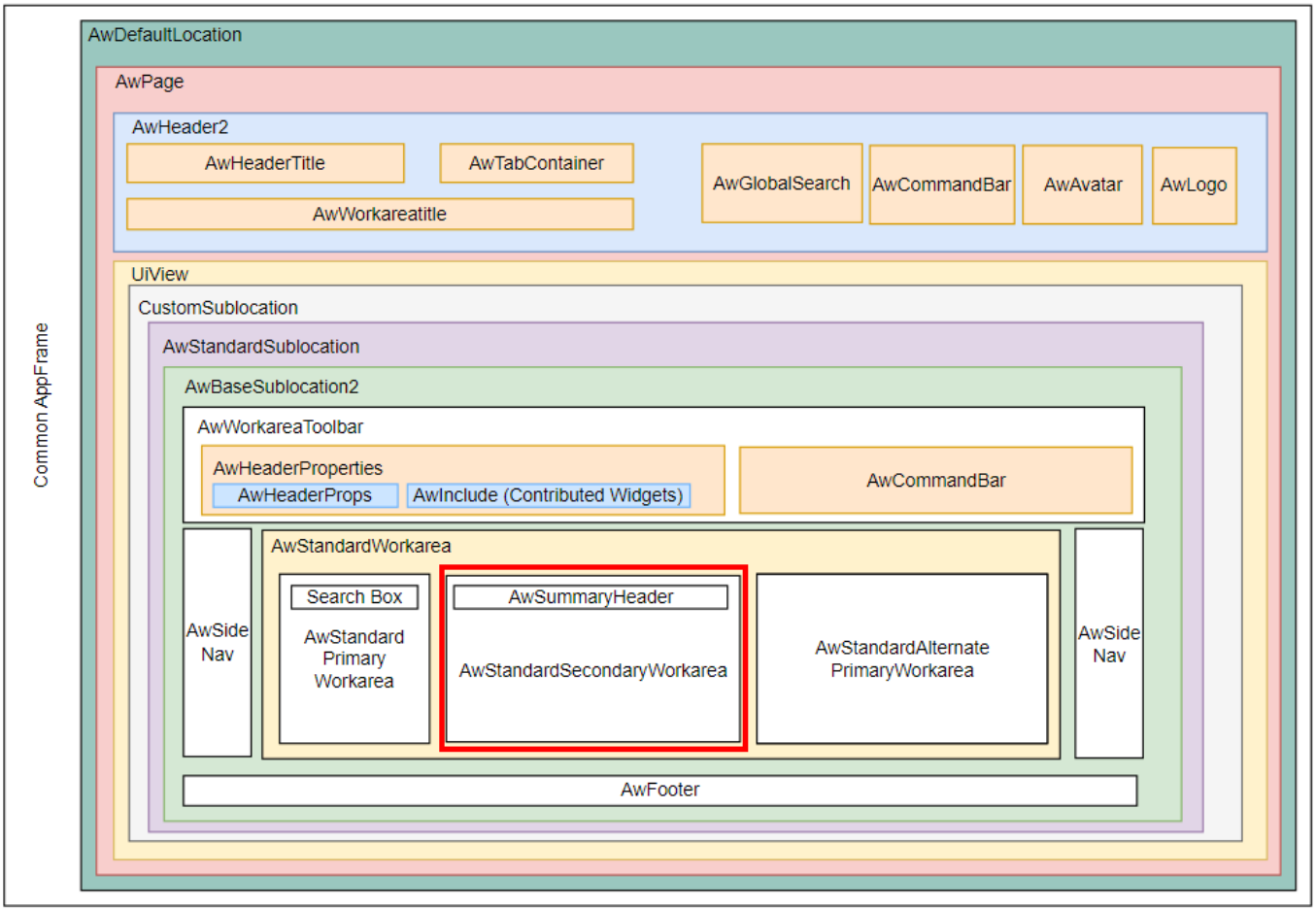
This is a higher-level component available in Fx which takes care of performing following responsibilities:

1. Given selections as input, show the summary/details as per the configuration specified via secondaryWorkareaTabs.json


Siemens Web Framework
Unpublished work. © 2025 Siemens

2. Consolidate the selections coming from the child components and update them to the parent component

It is a thin wrapper over another component called AwSelectionSummary which is responsible for reading and managing the configuration (secondaryWorkareaTabs.json) mentioned above. SWA as a component has very less to do except for passing the received selections to AwSelectionSummary to figure out which pages (tabs) should be rendered and under what conditions. Also, it displays Header information for selected object using AwSummaryHeader component.




### How to enable/disable Secondary Work Area?

Define 'secondaryWorkarea' on sublocation provider like below. By default, it is true. So, if not specified then Secondary Work area will be enabled by default.

```
"secondaryWorkarea": false
```

### How to configure View in Secondary Workarea?

You need to contribute tabs in secondaryWorkareaTabs.json like below. secondaryWorkareaTabs.json is module level configuration mechanism.

```
{
    "tabs": {
        "customView": {
            "name": "View Title",
            "priority": 100,
            "view": "viewComponentName",
            "visibleWhen": "selection.length === 1"
        },
        "customView2": {
            "name": {
                "source": "/i18n/UIMessages",
                "key": "viewTitleKey"
            },
            "priority": 200,
            "view": "viewComponentName",
            "visibleWhen": {
                "condition": "conditions.shouldCustomView2BeVisible"
            }
        }
    },
     "conditions": {
        "shouldCustomView2BeVisible": {
            "expression": {
                "$and": [ {
                        "$source": "selection.length",
                        "$query": {
                            "$gt": 0
                        }
                    }
                ]
            }
        }
    }
}
```

Details about attributes can be found in below table.

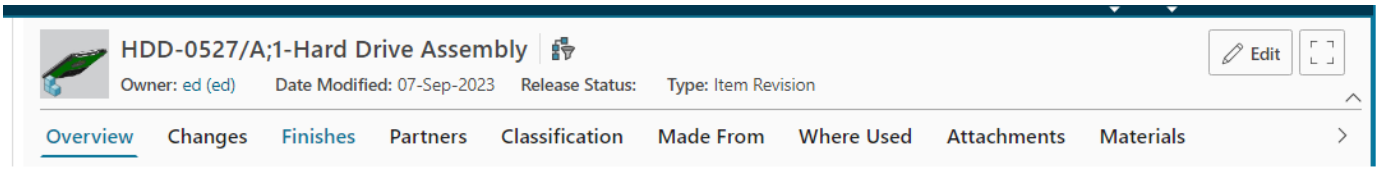
| Attributes | Description | Mandatory |
|---|---|---|
| name | This is Tab title. This can be either string or i18n configuration | true |
| view | This is a string; it represents component name which will be rendered in SWA | true |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | |
|---|---|---|
| visibleWhen | This is expression which will determine the visibility of Tab. This will be evaluated based on context value passed to SWA. This can be written in form of declarative condition object also. | false |
| priority | This is integer used to define tab order | false |

### Summary Header in SWA

SWA shows **Summary Header** which has basic information about current selection in PWA. It shows its icon, title, key properties, indicators and commands which can work in SWA content.

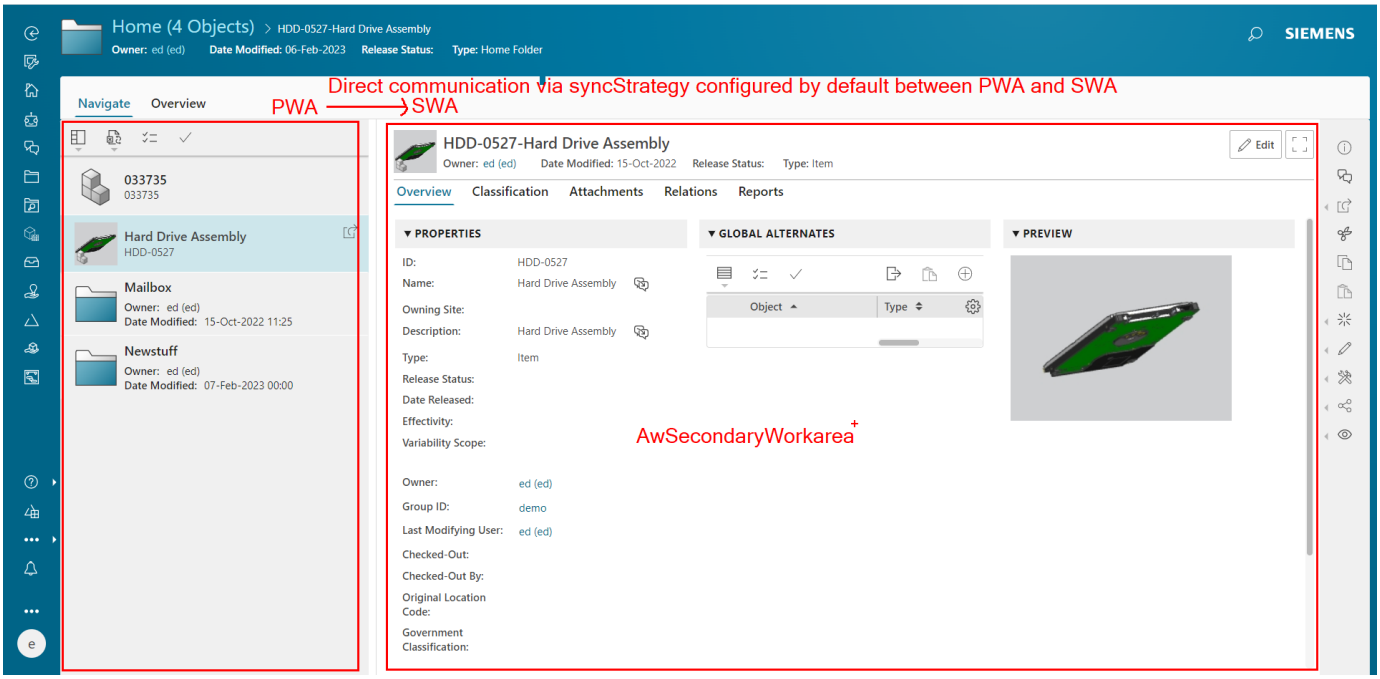



- This will be enabled if PWA selection has object_string **ViewModelProperty** populated on it.
- You can add **Commands** to command bars in Summary Header with anchors 'aw_summaryHeader' and 'aw_summaryHeaderRight' configured for SWA.
- Additionalaly you can override 'aw_summaryHeaderProperties' **layout slot** to show the header properties.

### Types of SWA

#### 1. AwSecondaryWorkarea

It works off the inputs received either from PWA (configured via syncStrategy) OR baseSelection (received from sublocation level component). It owns up the responsibility of managing a port called swaSelectionInput. It is bind to the PWA via Sync Strategy so whenever the selection in PWA is changed, the updated selections are received in this input port and SWA accordingly re-renders itself.




### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| baseSelection | The base or implicit selection; the details of which should be shown in SWA provided there are no explicit selections in PWA | Optional |
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |

### Props sent down in the component hierarchy

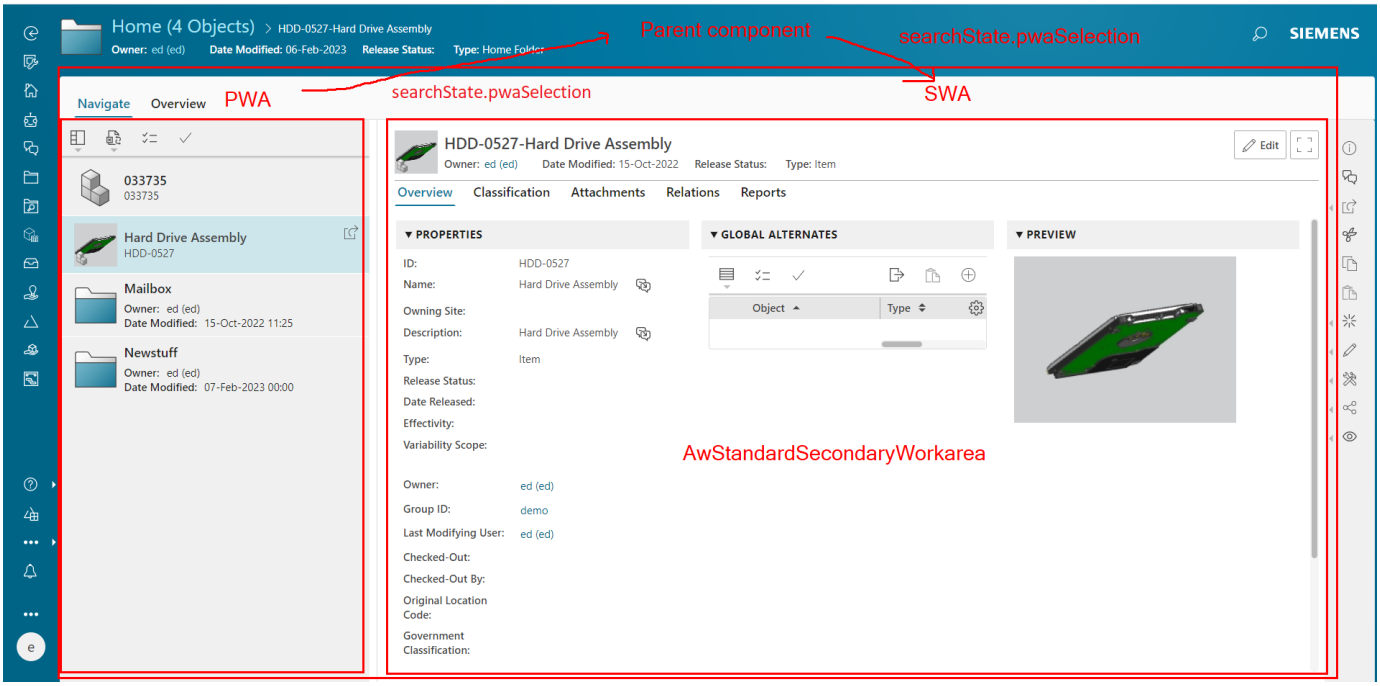
| Output Prop | Description |
|---|---|
| selection | The selections (irrespective of the source from which they came i.e., PWA or baseSelection) which are passed down to child components |
| context | The subPanelContext data which is passed down to child components |
| selectionData | The selection data (additional object required for managing the selections in SWA) which is passed down to child components |

### 2. AwStandardSecondaryWorkarea

It works off the inputs received either from PWA (received via parent component on searchState) OR baseSelection (received from sublocation level component). It owns up the responsibility of honoring the pwaSelection information received on searchState. The parent component is responsible for ensuring the selections from PWA are updated on pwaSelection and passed on to this component on searchState so


Siemens Web Framework
Unpublished work. © 2025 Siemens

whenever the selection in PWA is changed, the updated selections are received here and SWA accordingly re-renders itself.




### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| searchState | The parent state | Mandatory |
| searchState.pwaSelection | The parent state which contains the information about the explicit selections made in PWA | Mandatory |
| baseSelection | The base or implicit selection; the details of which should be shown in SWA provided there are no explicit selections in PWA | Optional |
| subPanelContext | Any additional data which needs to be passed down to child components | Optional |

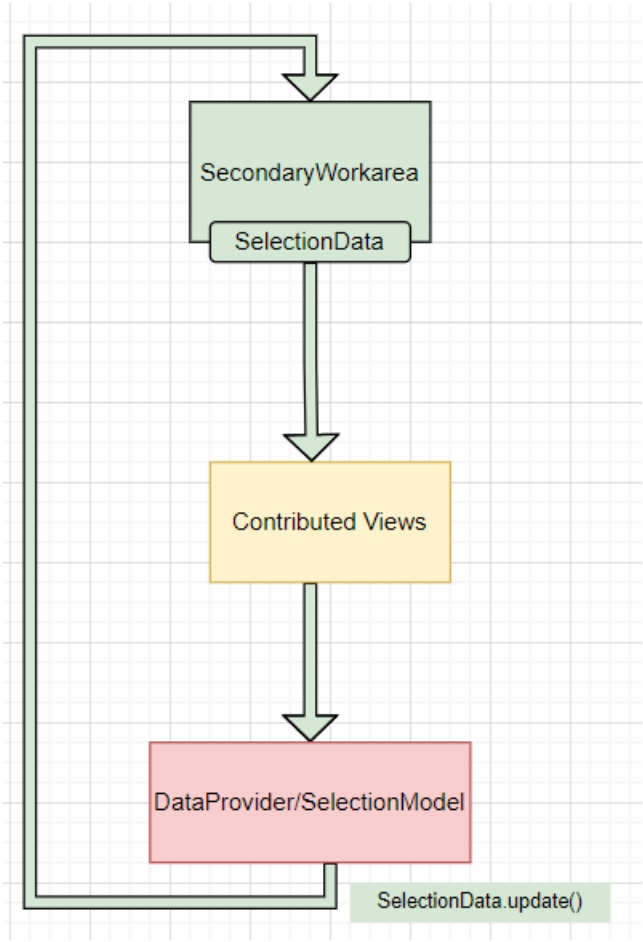
### Props sent down in the component hierarchy

| Output Prop | Description |
|---|---|
| searchState | The parent state |
| searchState.pwaSelection | The parent state which contains the information about the explicit selections made in PWA |
| selection | The selections (irrespective of the source from which they came i.e., PWA or baseSelection) which are passed down to child components |

| | |
|---|---|
| context | The subPanelContext data as well as the searchState which is passed down to child components |
| selectionData | The selection data (additional object required for managing the selections in SWA) which is passed down to child components |

### Selection Management in SWA

SWA is responsible for managing the selection within its child component hierarchy and updating the same information to the parent component hierarchy. The selection management is achieved via **selectionData**



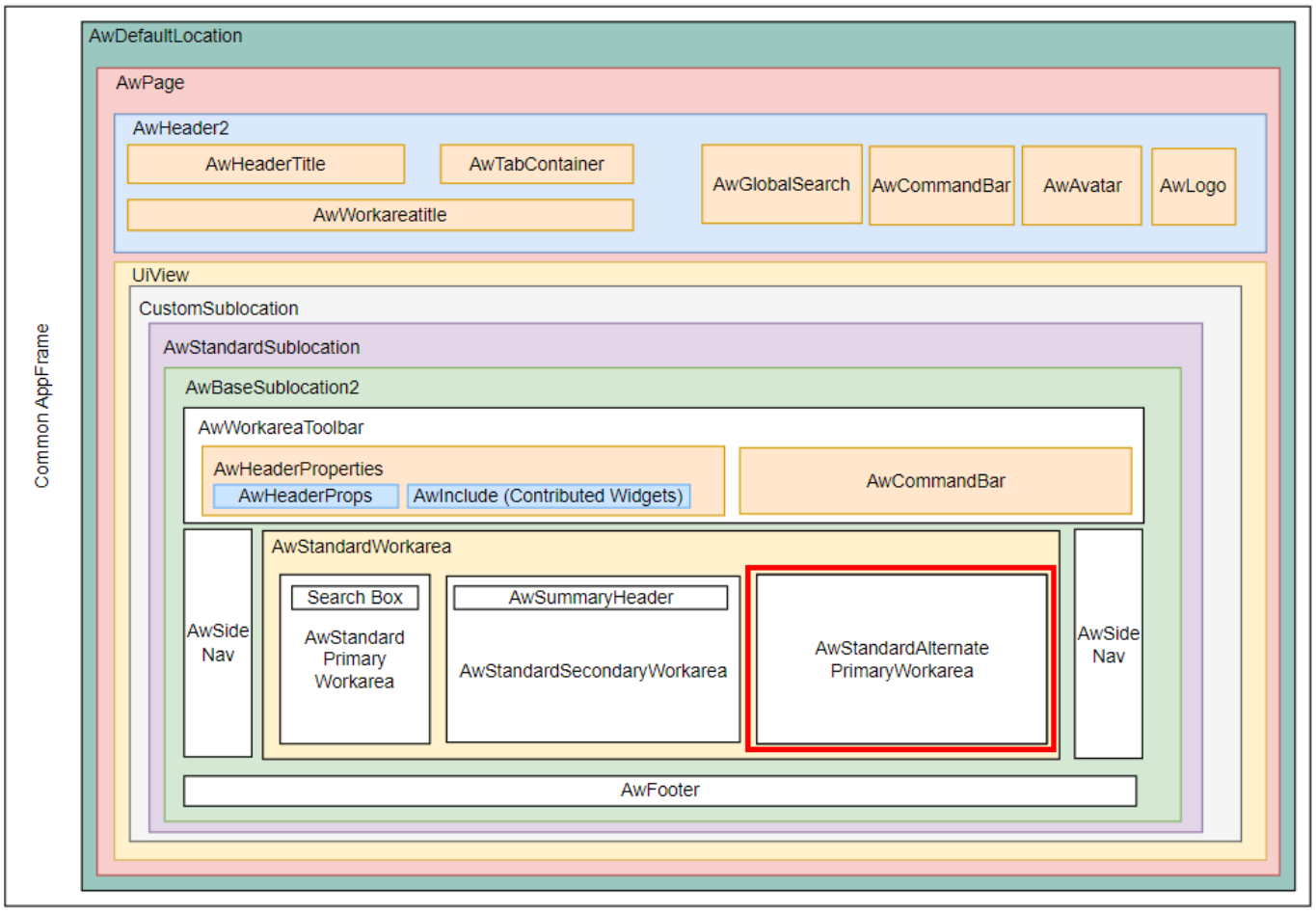

To manage the selection within its various child components, SWA creates its own selectionData and sends it to all its child components. It receives the update from its children and takes the responsibility of sending this information up in its hierarchy by invoking the update method available on the selectionData of its parent i.e., Workarea component. Before sending this information up, SWA appends the source as secondary on the received selectionData. This information of source would be required by the parent component to orchestrate the selections properly.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Alternate Primary Workarea (Alt PWA)

This is a higher-level component available in Fx which takes care of performing following responsibilities:

1. Render the Alternate Primary Workarea view configured via the command's mechanism.
2. Consolidate the selections coming from the child components and update them to the parent component.
3. It represents the same inforamtion as there in Primary workarea but just alternate representation of it. Say you are looking at a tree representing car assembly in PWA - you could have the same assembly visualized and rendered in AlternatePWA view. So you are essentially looking at same inforamtion in two different views - PWA and AltPWA.

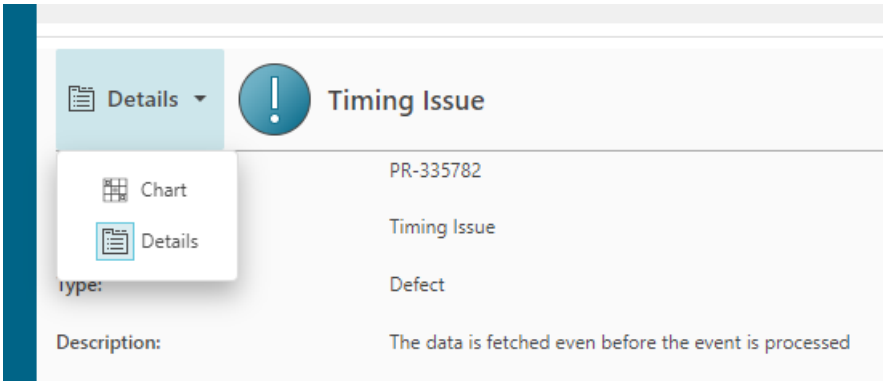


### How to contribute Alternate Primary Work Area view mode?

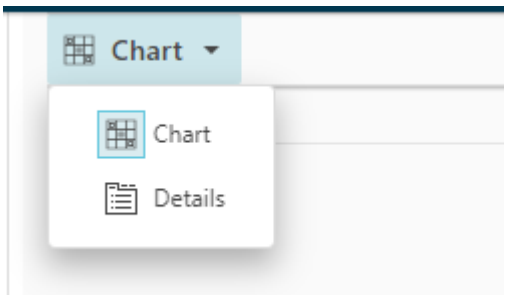
First you need to contribute a view mode command in commandsViewModel.json and write its placement in below format so that it appears on Summary Header in SWA and custom command bar in Alt PWA.This will show Alt PWA command in Summary Header View Toggle.

```
"customAlternatePwaSummaryCmdPlacement": {
  "id": "customAltwPwaCmdId",
  "parentGroupId": "Awp0DisplayTogglesSummary",
  "priority": 100
},
"customAlternatePwaCmdPlacement": {
  "id": "customAltwPwaCmdId",
  "parentGroupId": "Awp0DisplayToggles",
  "priority": 100
}
```

Alternate PWA commands in Summary Header View Toggle.




Alternate PWA commands in Alternate PWA View Toggle.




Then you need add command handler and write its action in below format. This will set alternate PWA view to given component in viewTuple->alternatePWA.

```
{
  "setCustomAlternatePWA": {
    "actionType": "JSFunction",
    "method": "changeViewMode",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "deps": "js/pageLayoutService",
    "inputData": {
      "viewTupleState": "{{commandContext.viewTuple}}",
      "viewTuple": {
        "alternatePWA": "customAltPWAView"
      },
      "sublocationState": "{{commandContext.pageContext.sublocationState}}",
      "title": "{{i18n.customAltPWATitle}}"
    }
  }
}
```

Then you need to write selectWhen expression based on below condition so that command is selected when Alt PWA is selected. Here, alternatePWA is Component name refers to the action->inputData->viewTuple->alternatePWA.

```
"isAwp0CustomView": {
    "expression": "commandContext.viewTuple.alternatePWA ===
'customAltPWAView'"
}
```

**How to contribute Alternate Primary Workarea View?**

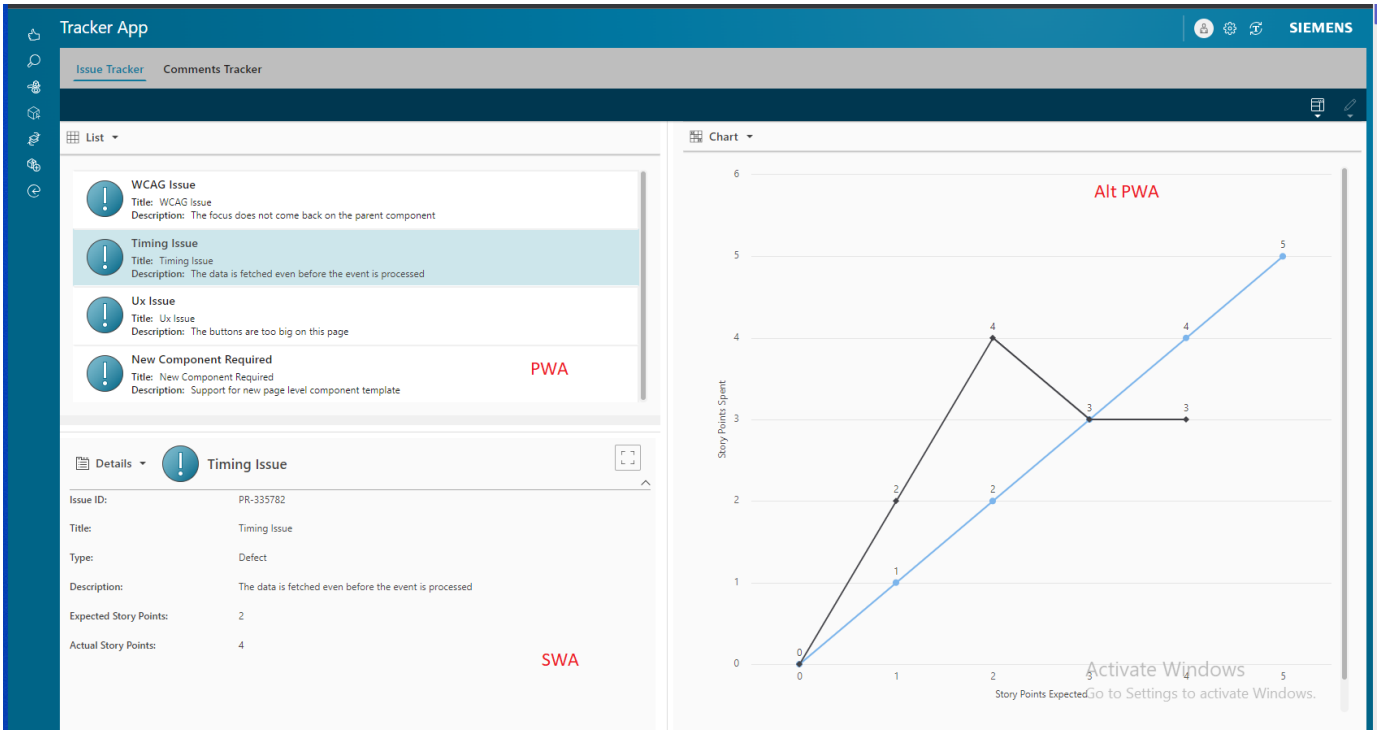
You need to define View/ViewModel with name given in action->inputData->viewTuple->alternatePWA. In its View(HTML) file you need to add AwCommandBar component so that it can host a place for Display toggles.

```
<aw-command-bar anchor='aw_altPWADisplay' context='props.subPanelContext'></
aw-command-bar>
---Alt PWA content goes here---
```

**Types of Alt PWA**

**1. AwStandardAlternatePrimaryWorkarea**

This component is responsible for rendering given Alternate PWA View. It passes children selection up in hierarchy using selectionData mechanism.




### Props received from the parent component hierarchy

| Input Prop | Description | Mandatory |
|---|---|---|
| searchState | The parent State | Mandatory |
| baseSelection | The base or implicit selection | Optional |
| selectionData | The selection data (additional object required for managing the selections in Alt PWA) which is passed down to child components | Optional |
| subPanelContext | subPanelContext received from prop and any additional data which needs to be passed down to child components | Mandatory |

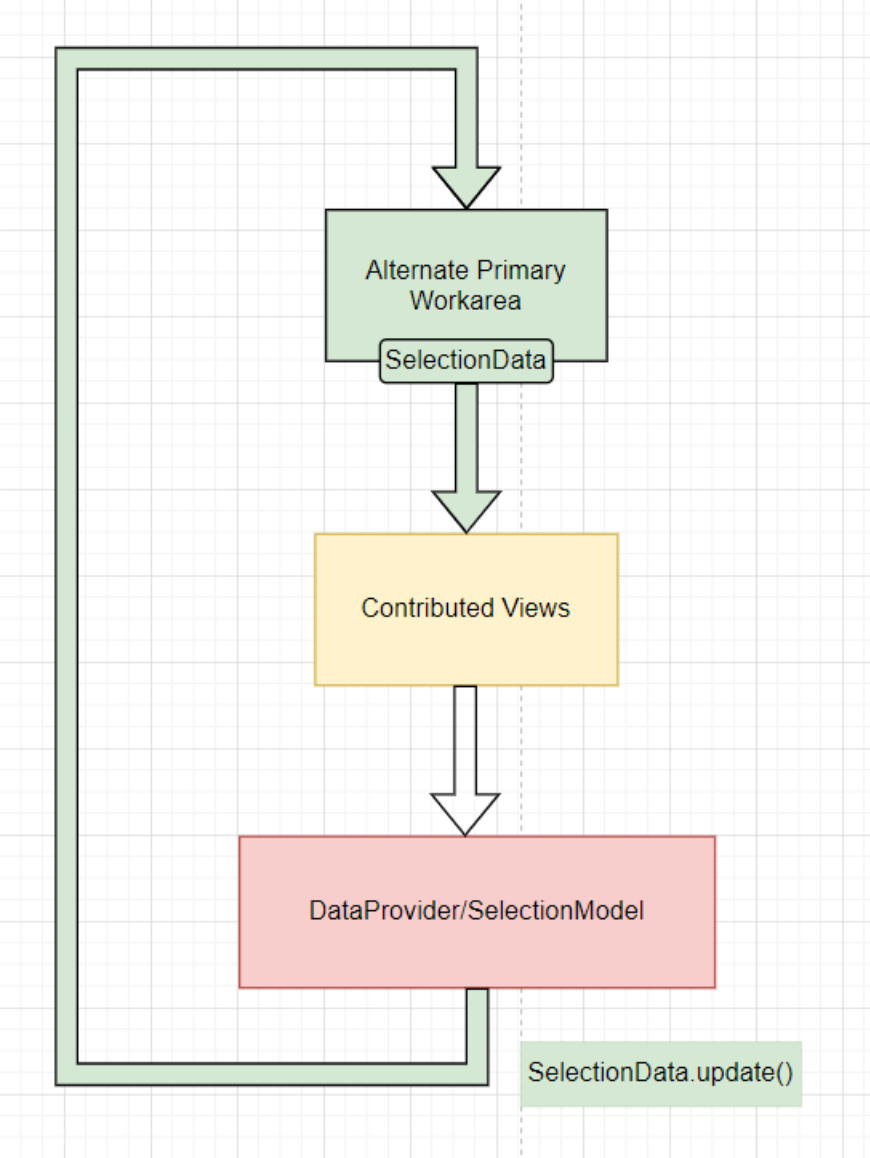
### Props sent down in the component hierarchy

| Output Prop | Description |
|---|---|
| subPanelContext.viewTuple | The View Tuple State |
| subPanelContext.selection | The selections (irrespective of the source from which they came i.e., PWA or baseSelection) which are passed down to child components |
| subPanelContext.context | The subPanelContext data as well as the searchState which is passed down to child components |
| subPanelContext.selectionData | The selection data (additional object required for managing the selections in SWA) which is passed down to child components |


Unpublished work. © 2025 Siemens

### Selection Management in Alternate PWA

Alt SWA is responsible for managing the selection within its child component hierarchy and updating the same information to the parent component hierarchy. The selection management is achieved via **selectionData**




To manage the selection within its various child components, Alt PWA creates its own selectionData and sends it to all its child components. It receives the update from its children and takes the responsibility of sending this information up in its hierarchy by invoking the update method available on the selectionData of its parent i.e., Workarea component. Before sending this information up, Alt PWA appends the source as secondary on the received selectionData. This information of source would be required by the parent component to orchestrate the selections properly.

## Location and Sublocation Configuration

A location and a sublocation are configured in states.json - which is a module level configuration file. Example:

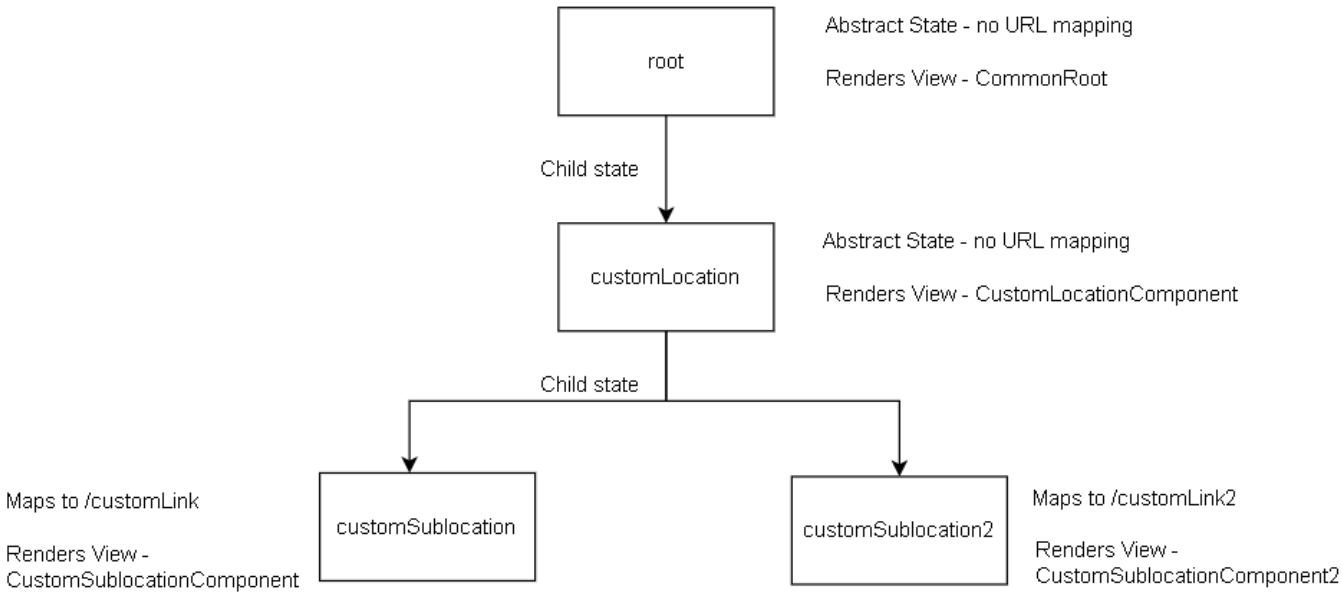
```
{
  "customLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/customConstants",
        "key": "customLocationTitle"
      }
    },
    "view": "customLocationComponent",
    "parent": "root"
  },
  "customSublocation": {
    "url": "/customLink",
    "data": {},
    "parent": "customLocation",
    "view": "CustomSublocationComponent"
  },
  "customSublocation2": {
    "url": "/customLink2",
    "data": {},
    "parent": "customLocation",
    "view": "CustomSublocationComponent2"
  }
}
```

The root state is defined in SWF as following:

```
{
  "root": {
    "abstract": true,
    "view": "CommonRoot"
  }
}
```

The example state definitions would create an hierarchy of states like shown below:


Siemens Web Framework
Unpublished work. © 2025 Siemens



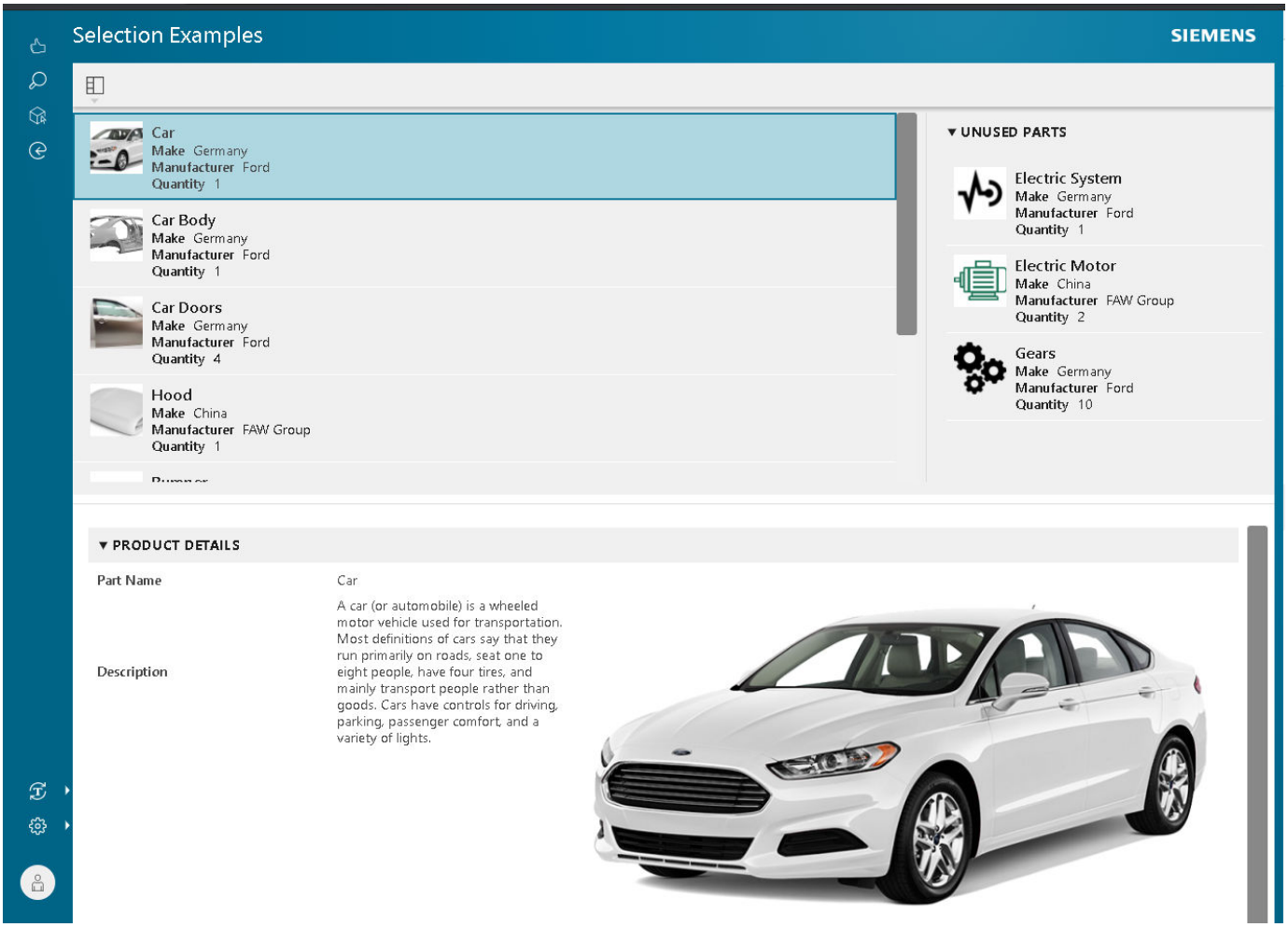

# Location and Sublocation Examples

## Example 1: With CommonLocation and Custom Sublocation using AwStandardSublocation

**States Definition**:

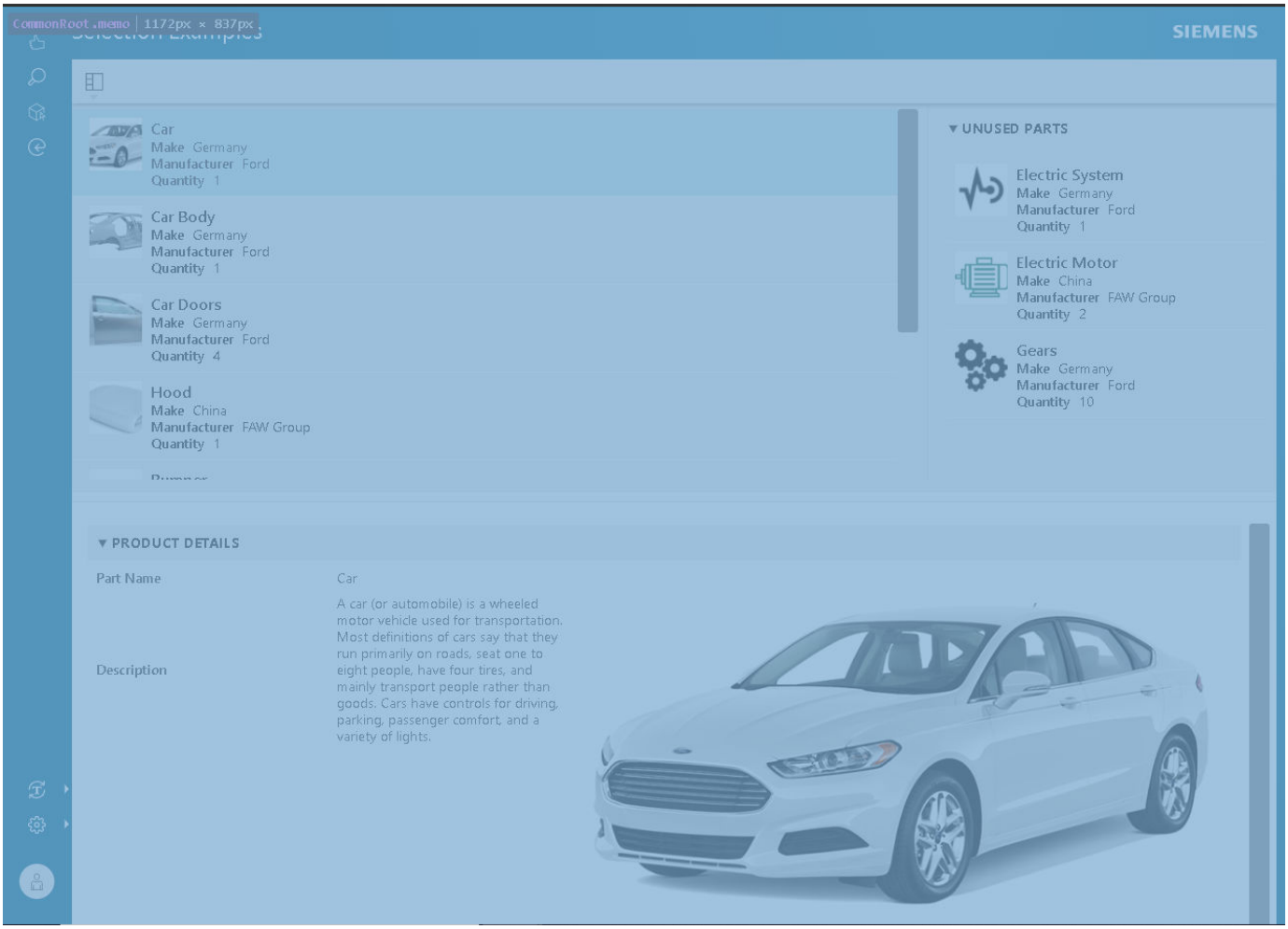
```
{
  "selectionExamplesCommonLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/selectionExamplesMessages",
        "key": "selectionExamplesLocationTitle"
      }
    },
    "view": "commonLocation",
    "parent": "root"
  },
  "selectionExample": {
    "url": "/selectionExample",
    "data": {},
    "parent": "selectionExamplesCommonLocation",
    "view": "SelectionExample"
  }
}
```

When URL is mapped to /selectionExample:

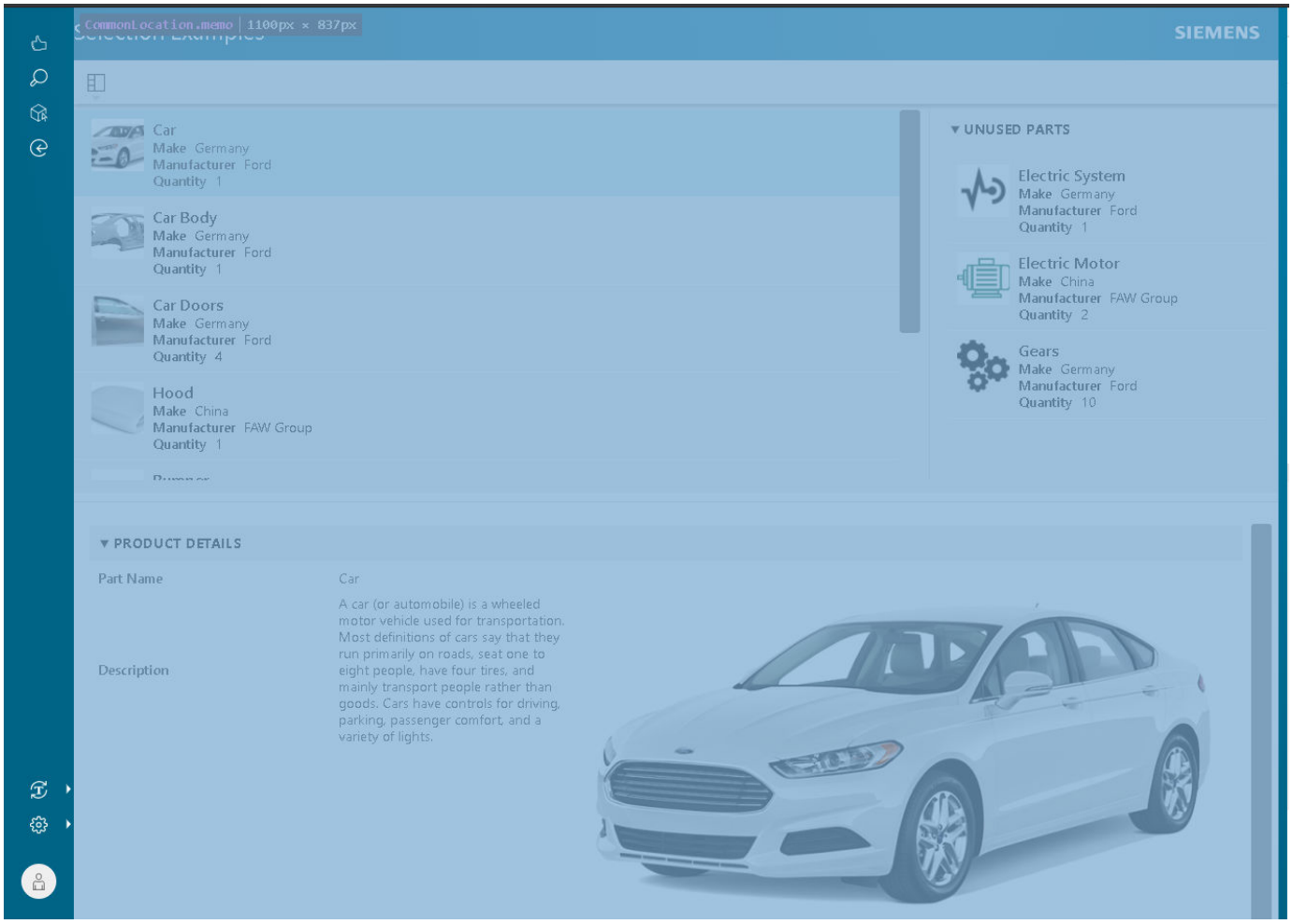



**Common Root - Defined in SWF**


Siemens Web Framework
Unpublished work. © 2025 Siemens



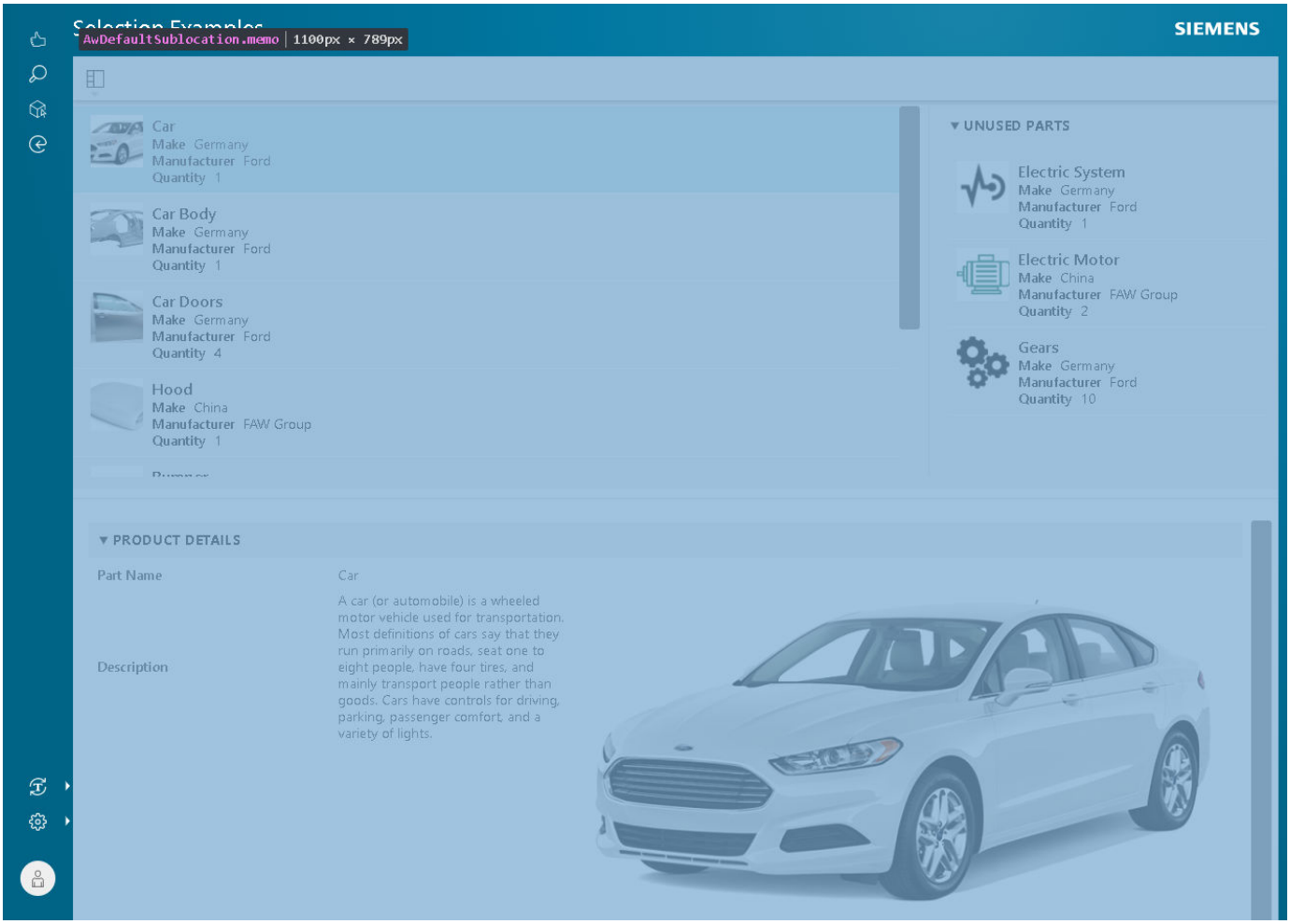

**Location Component - commonLocation**




**Sublocation Component - SelectionExample**



Unpublished work. © 2025 Siemens




## Example 2: With AwDefaultLocation and Custom Sublocation using AwStandardSublocation

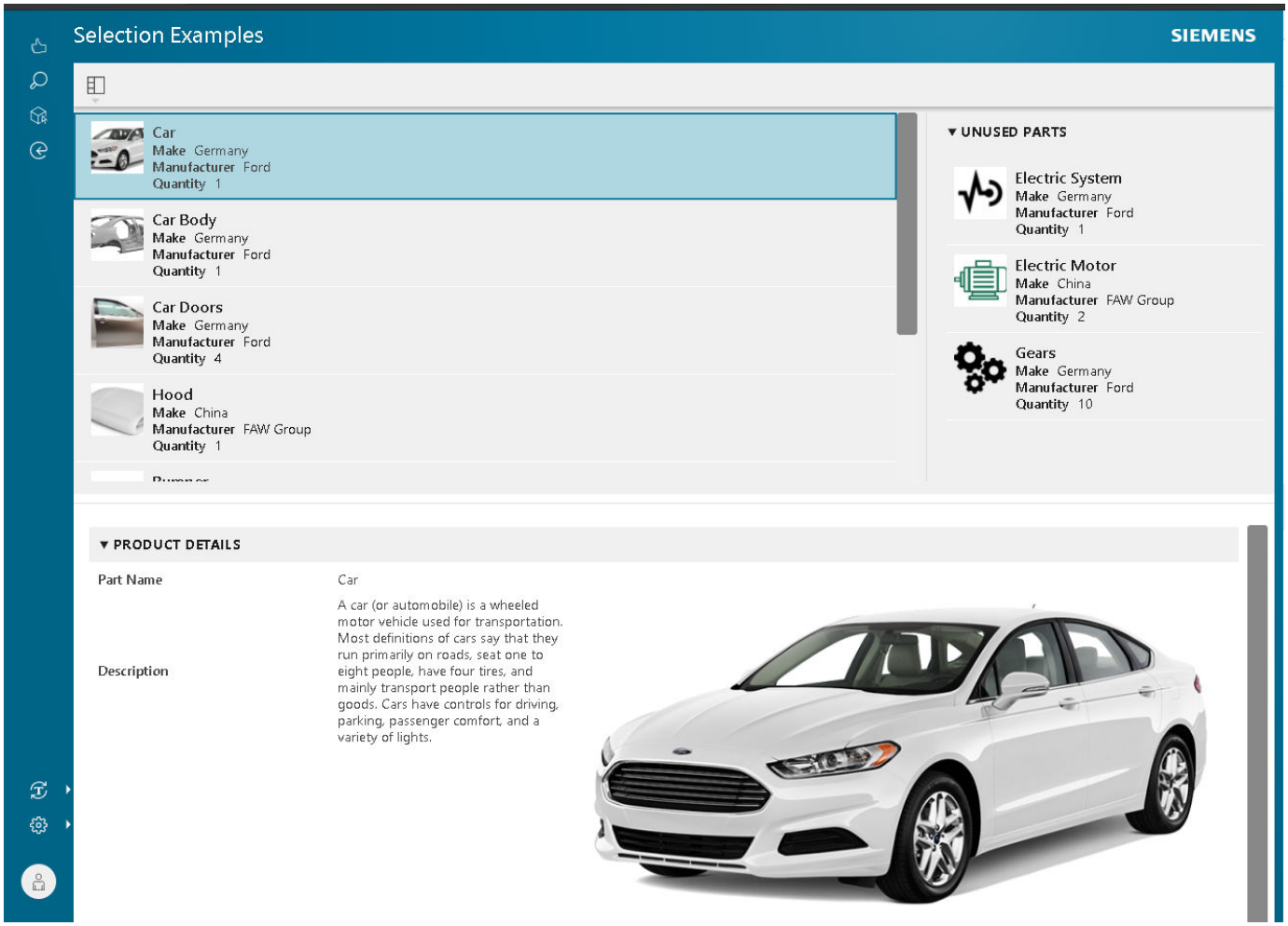
**States Definition**:

`type` as "location" maps the state to AwDefaultLocation

```
{
  "selectionExamplesCommonLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/selectionExamplesMessages",
        "key": "selectionExamplesLocationTitle"
      }
    },
    "type": "location",
    "parent": "root"
  },
  "selectionExample": {
    "url": "/selectionExample",
```

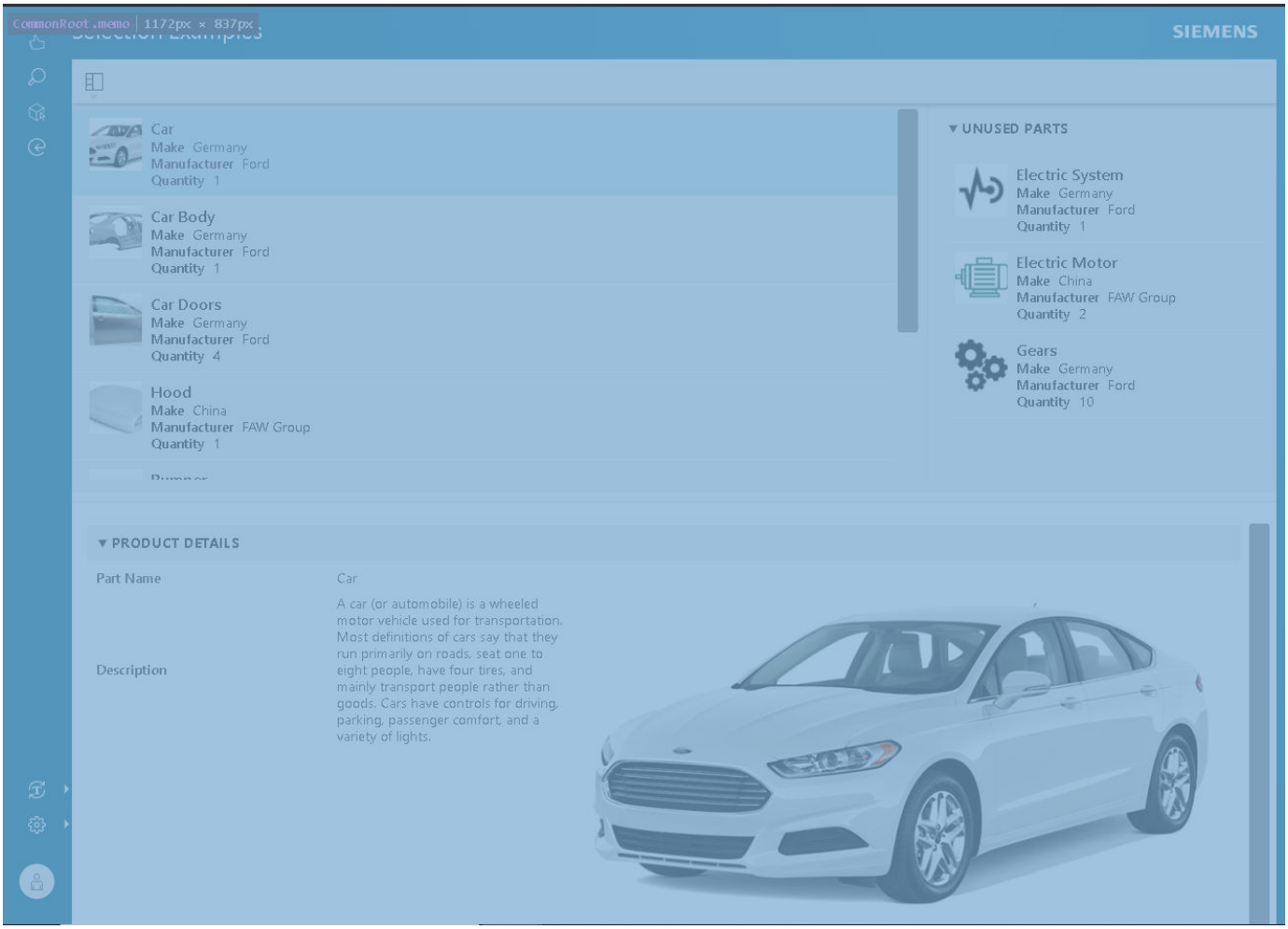
```
    "data": {},
    "parent": "selectionExamplesCommonLocation",
    "view": "SelectionExample"
  }
}
```

When URL is mapped to /selectionExample:

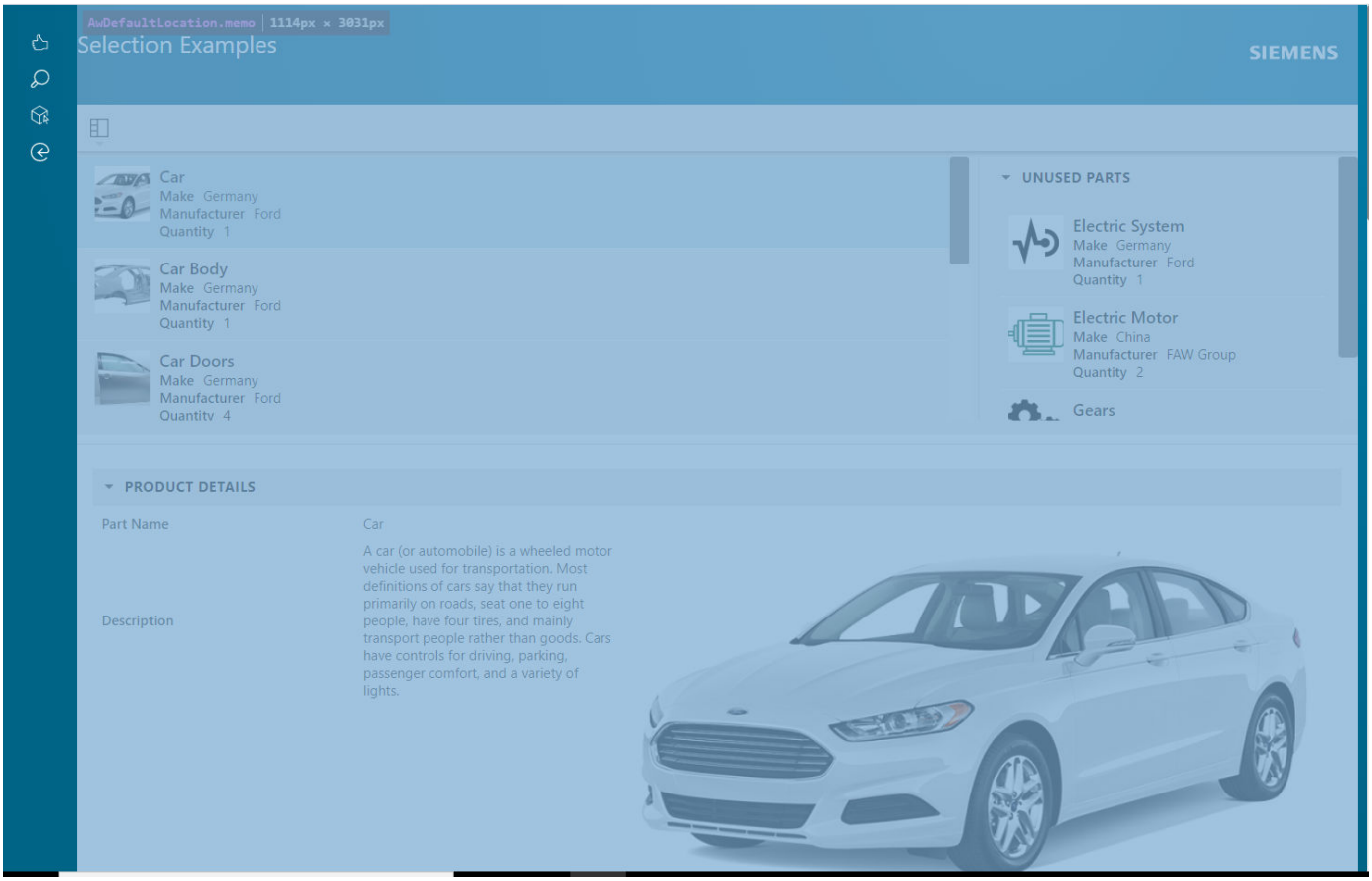



**Common Root - Defined in SWF**


Siemens Web Framework
Unpublished work. © 2025 Siemens

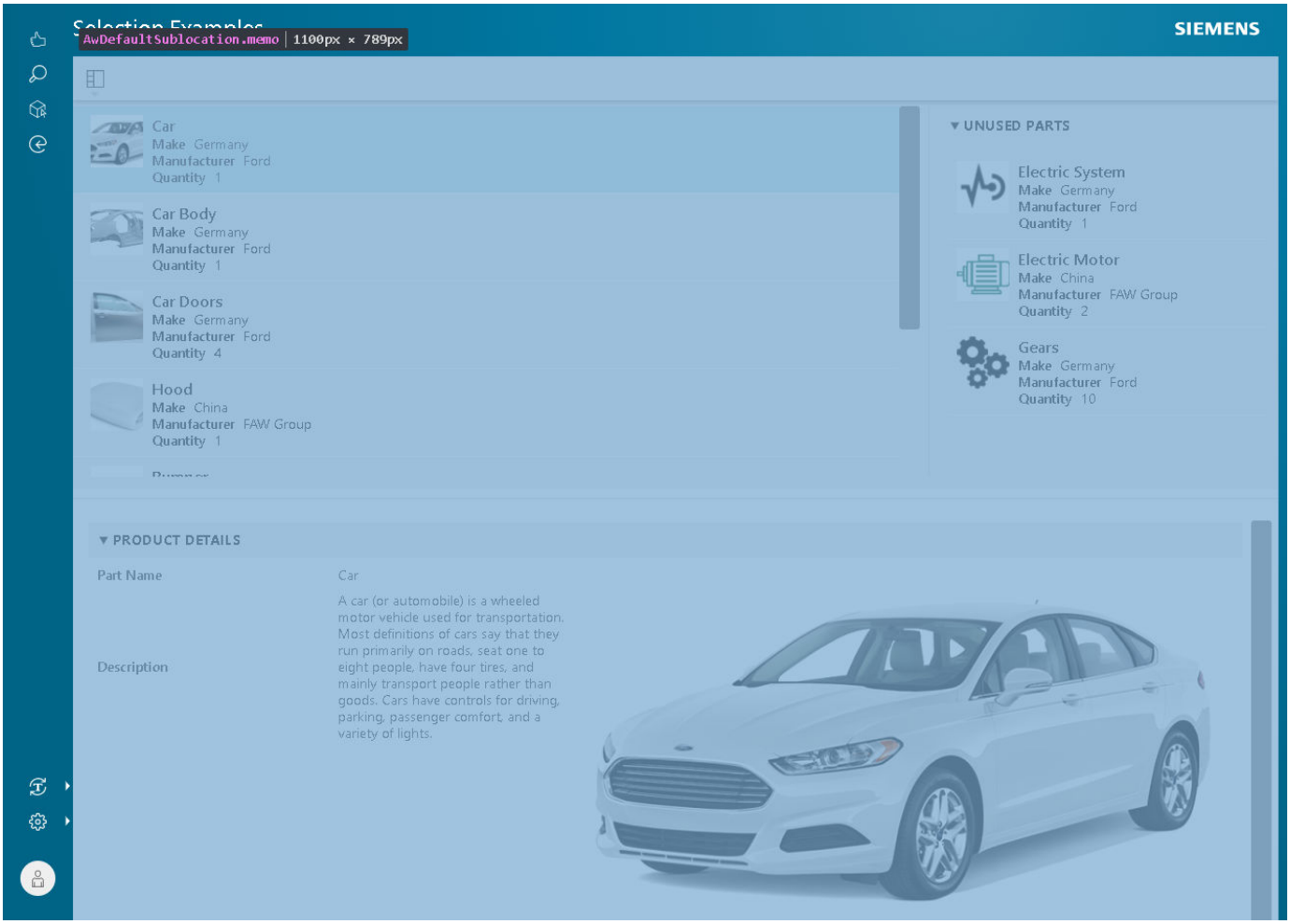



**Location Component - AwDefaultLocation**




**Sublocation Component - SelectionExample**


Siemens Web Framework
Unpublished work. © 2025 Siemens



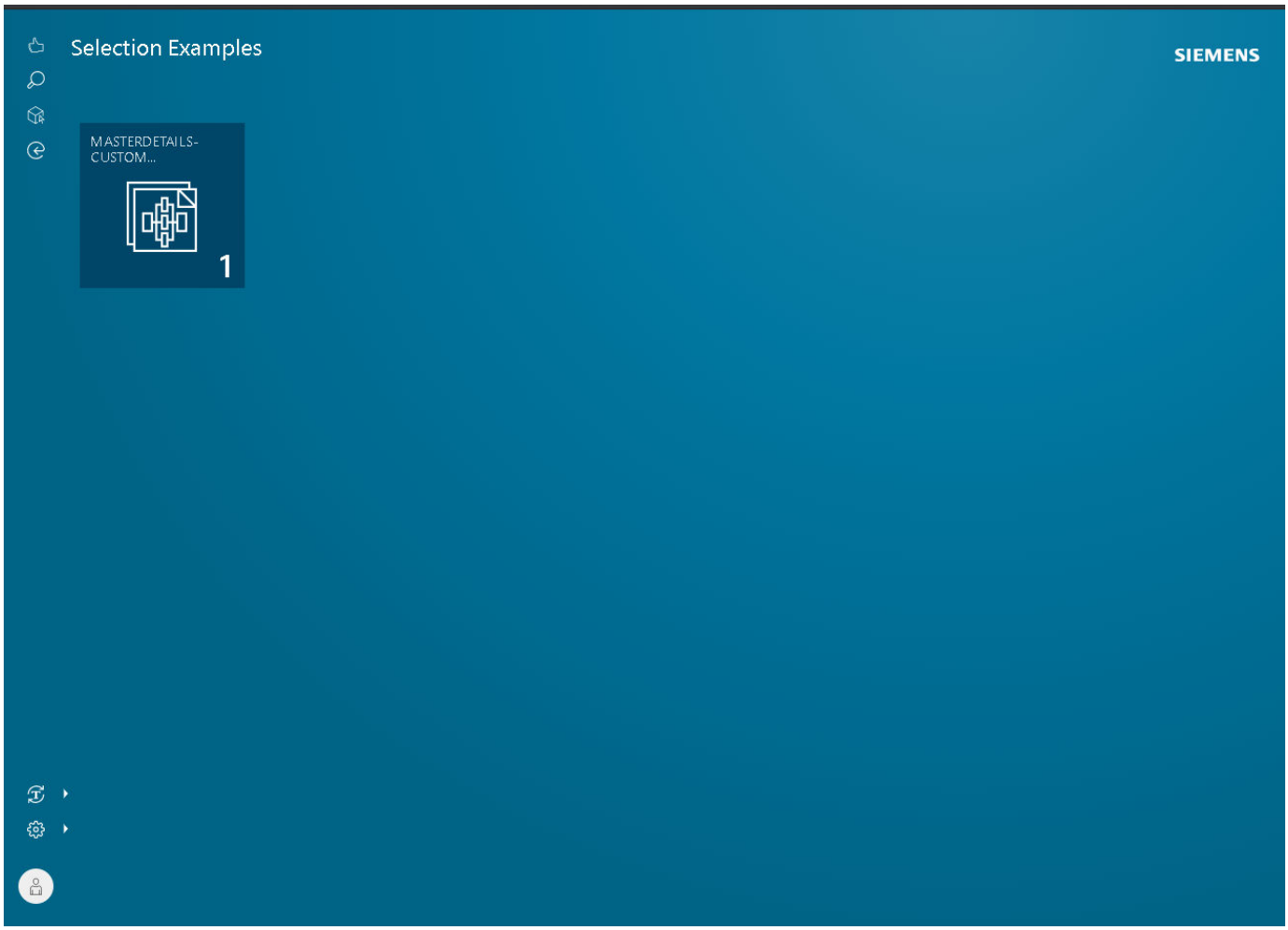

## Example 3: With Custom Location using AwDefaultLocation and Custom Sublocation

**States Definition**:

```
{
  "selectionExamplesLocation": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/selectionExamplesMessages",
        "key": "selectionExamplesLocationTitle"
      }
    },
    "view": "SelectionExamplesLocation",
    "parent": "root"
  },
  "selectionExamplesLandingPage": {
    "url": "/selectionExamples",
    "data": {},
    "parent": "selectionExamplesLocation",
    "view": "SelectionExamplesLandingPage"
```

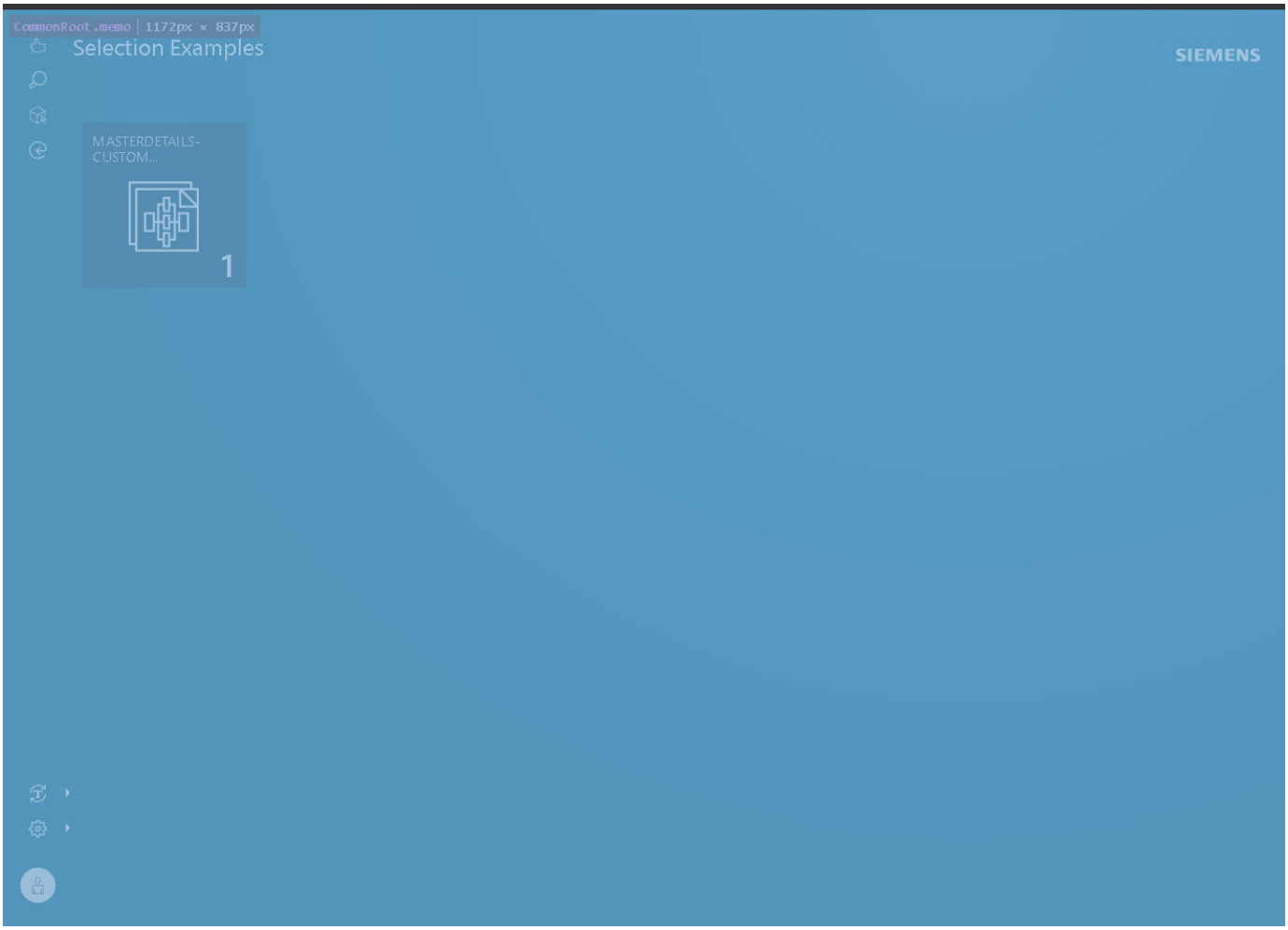
```
    }
}
```

When URL is mapped to /selectionExample:

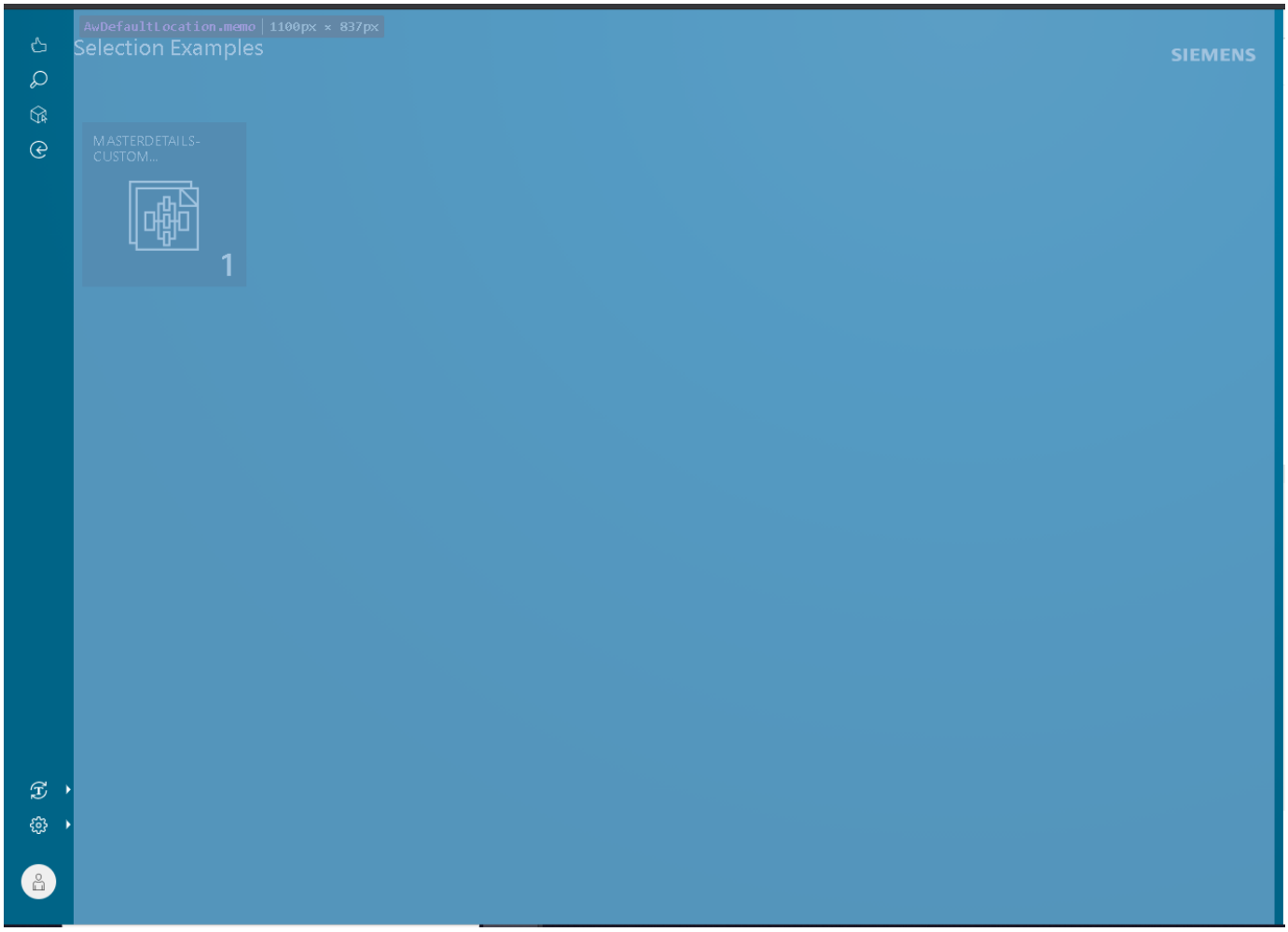



**Common Root - Defined in SWF**


Siemens Web Framework
Unpublished work. © 2025 Siemens

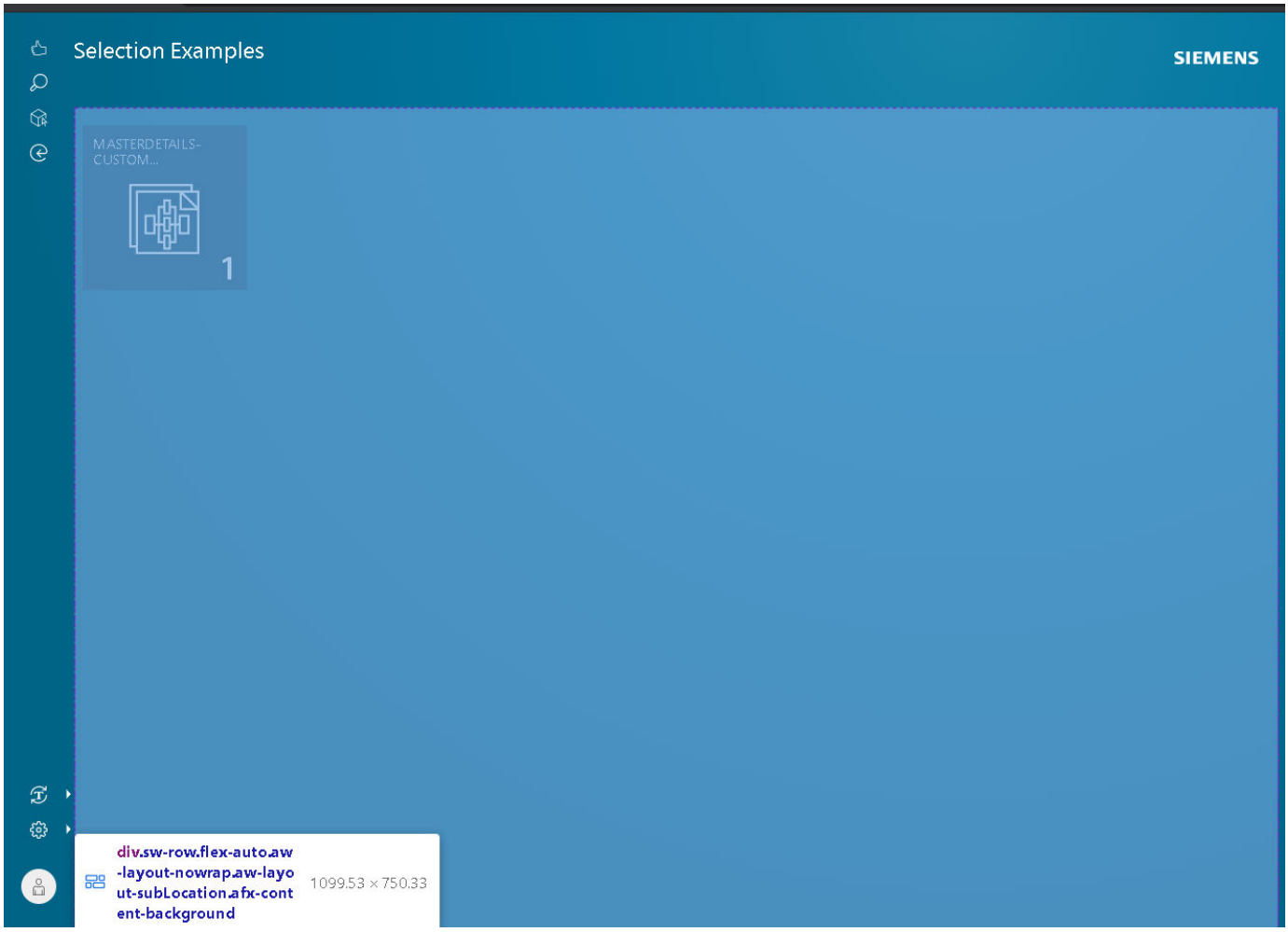



**Location Component - SelectionExamplesLocation**




**Sublocation Component - SelectionExamplesLandingPage**


Siemens Web Framework
Unpublished work. © 2025 Siemens



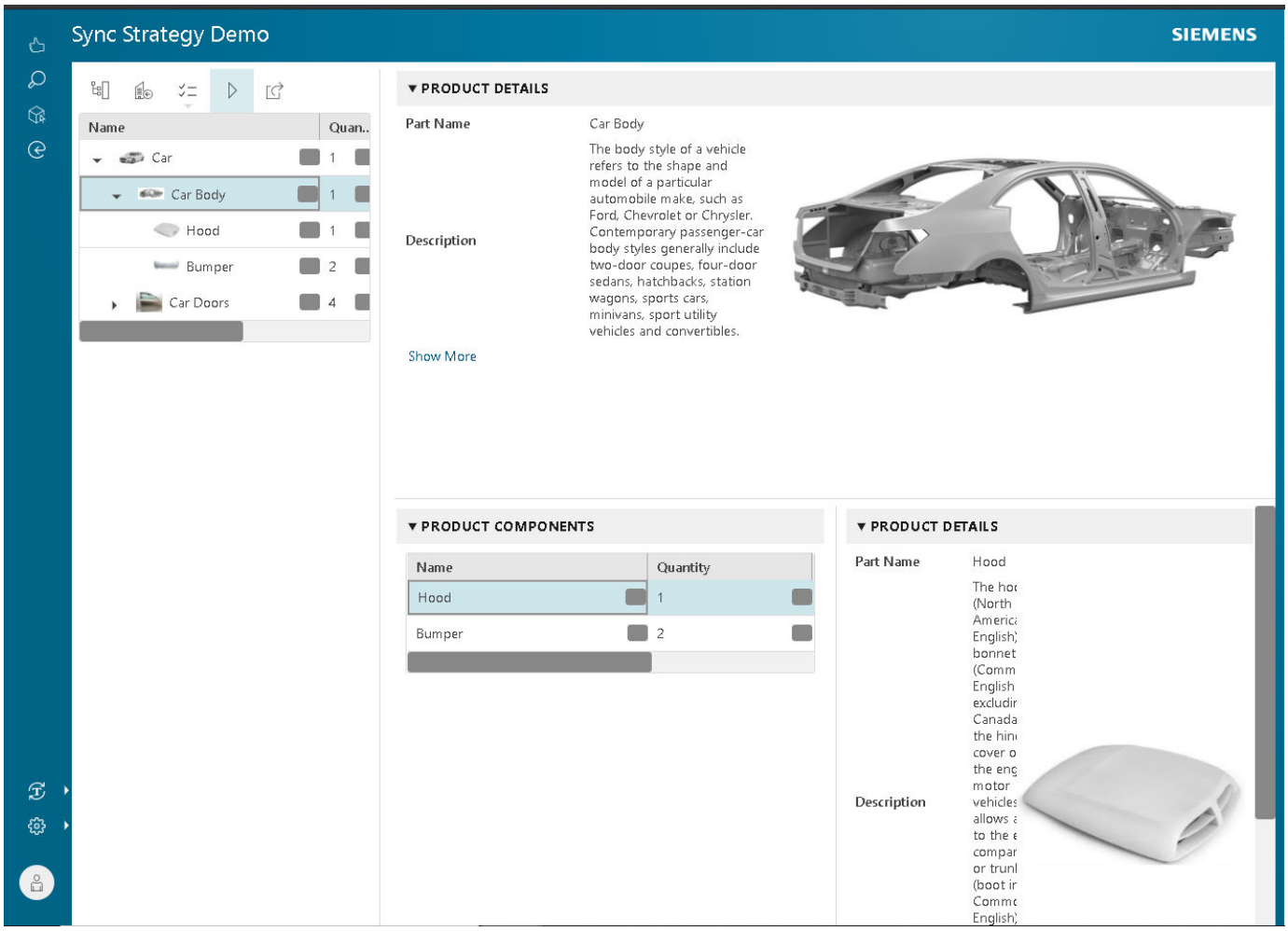

## Example 4: With CommonLocation and Custom Sublocation not using AwStandardSublocation

**States Definition**:

```
{
  "syncStrategyApp": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/locationAndSublocationMessages",
        "key": "syncStrategySubLocationTitle"
      }
    },
    "view": "commonLocation",
    "parent": "root"
  },
  "syncStrategyDemo": {
    "url": "/syncStrategyDemo",
    "data": {},
```

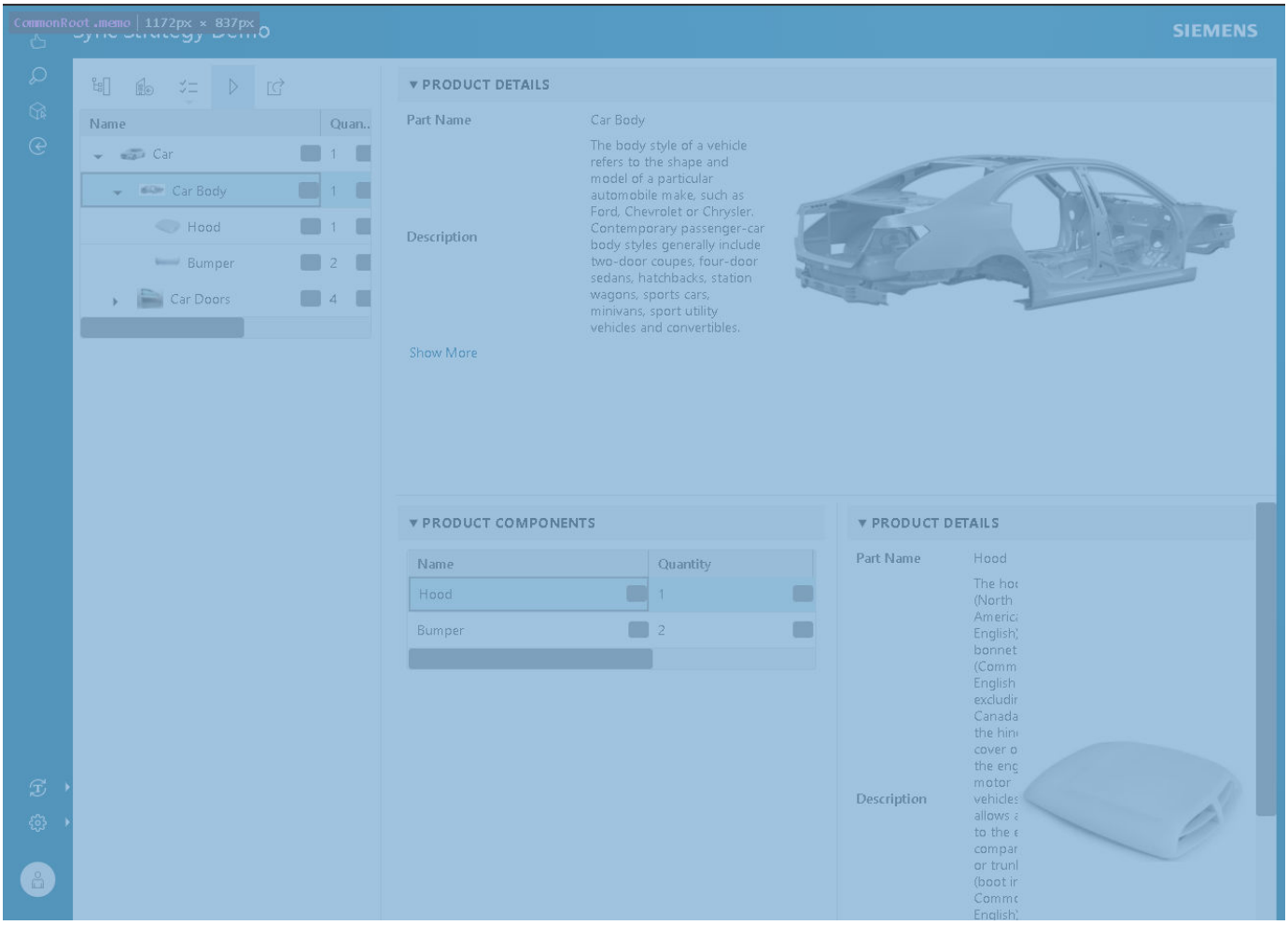
```
    "parent": "syncStrategyApp",
    "view": "syncStrategyDemo"
  }
}
```

When URL is mapped to /syncStrategyDemo:

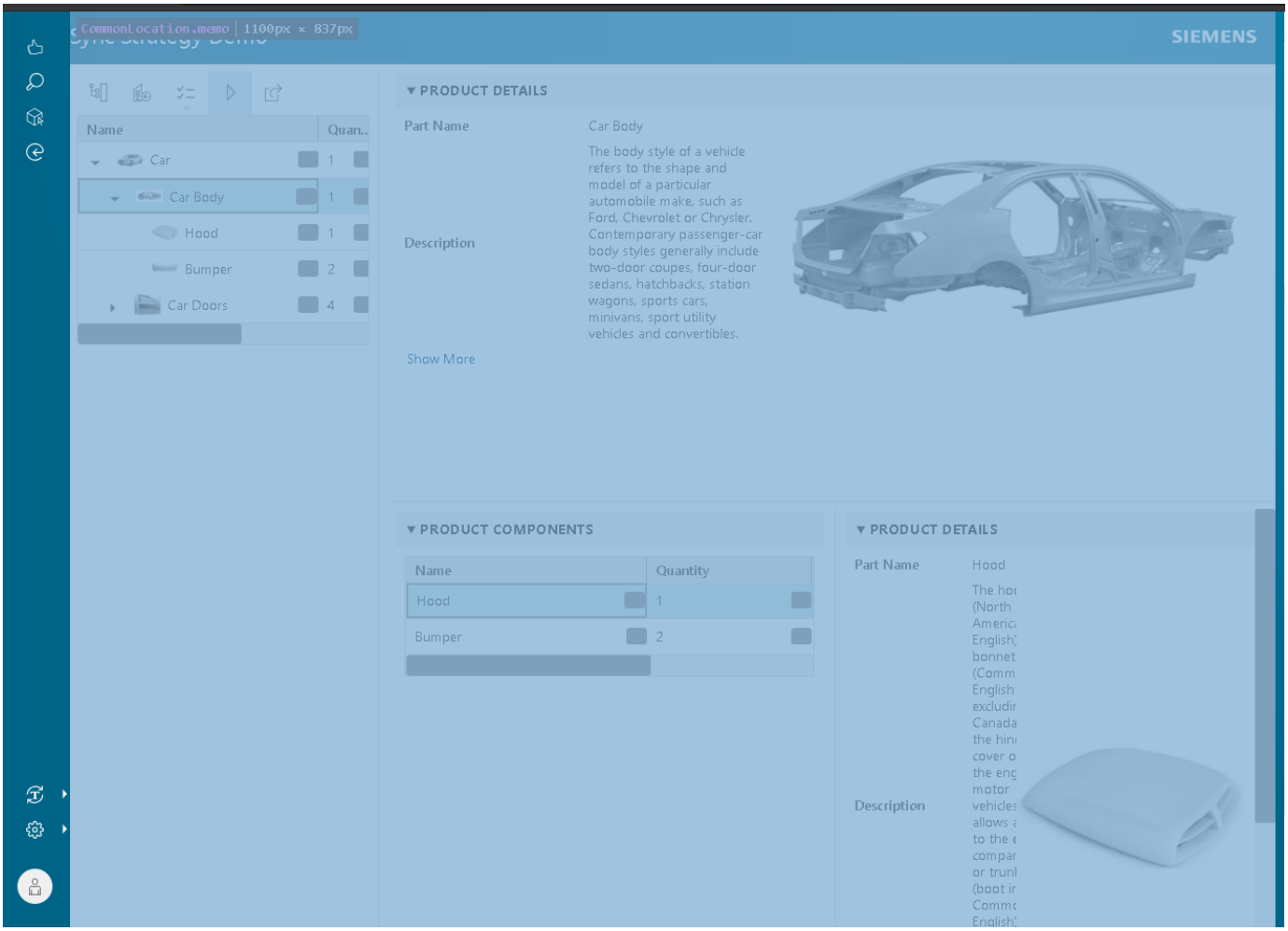



**Common Root - Defined in SWF**


Siemens Web Framework
Unpublished work. © 2025 Siemens

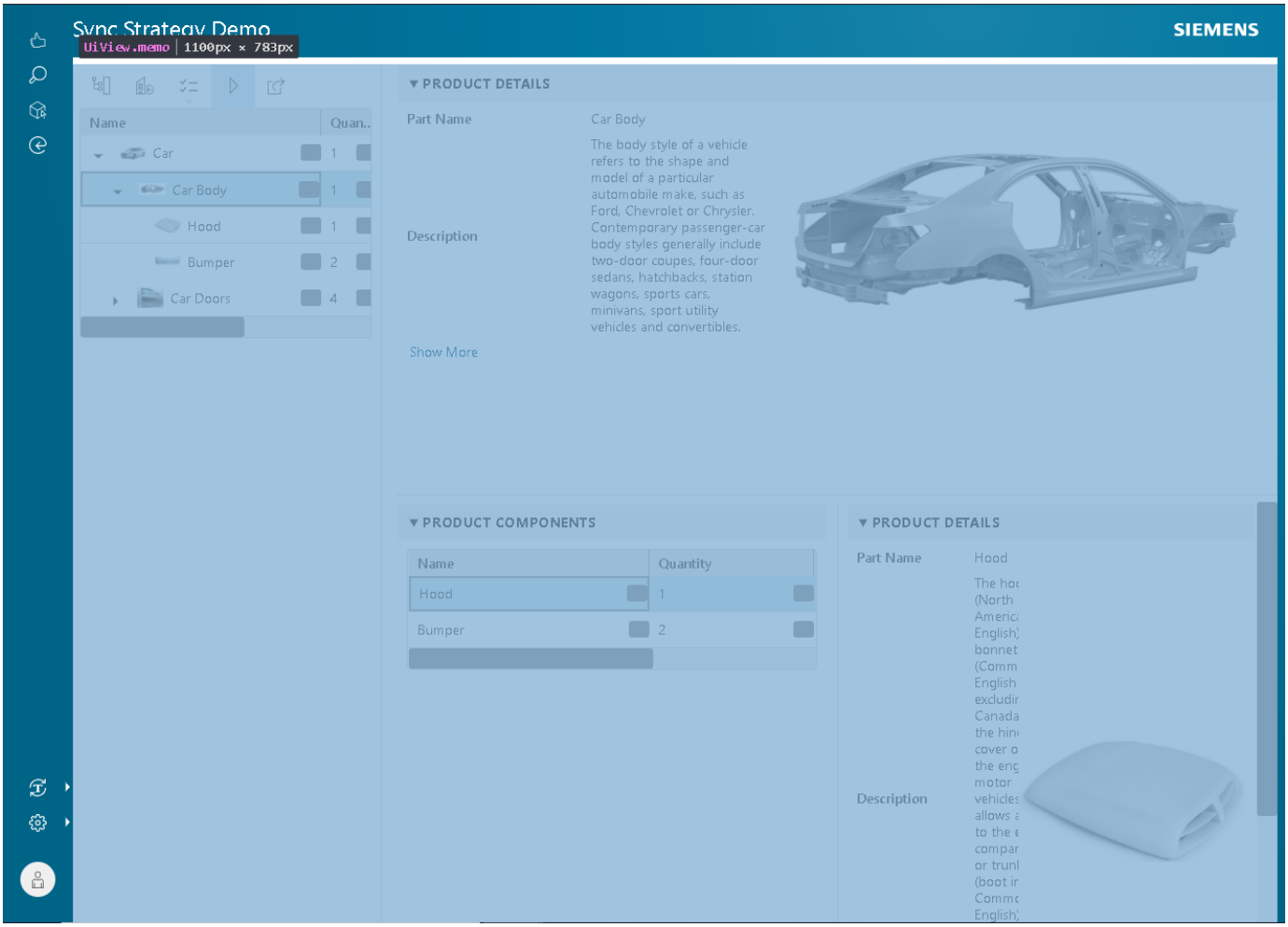



**Location Component - commonLocation**




**Sublocation Component - syncStrategyDemo**


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Example 5: With CommonLocation and AwStandardSublocation

**States Definition**: `view` as "AwStandardSublocation" maps the state to AwStandardSublocation

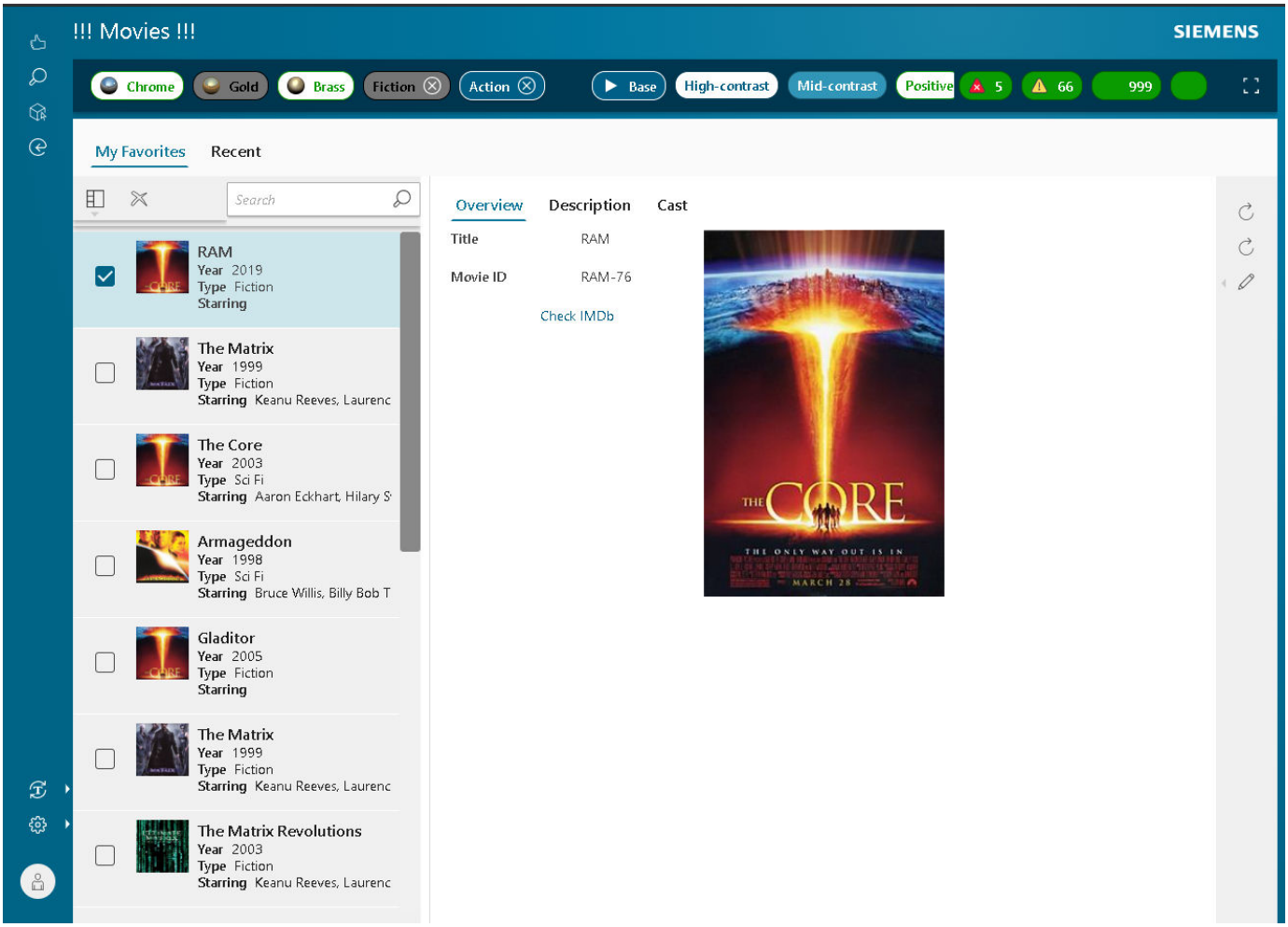
```
{
  "movies": {
    "abstract": true,
    "data": {
      "headerTitle": {
        "source": "/i18n/locationAndSublocationMessages",
        "key": "moviesLocationTitle"
      }
    },
    "view": "commonLocation",
    "parent": "root"
  },
  "myFavs": {
    "url": "/myFavs",
    "data": {
      "label": {
        "source": "/i18n/locationAndSublocationMessages",
```

```
        "key": "favoritesSubLocationTitle"
      },
      "nameToken": "com.siemens.splm.client.movies:favoritesSubLocation",
      "clientScopeURI": "",
      "viewBase": "favorites"
    },
    "params": {
      "d_uids": null,
      "s_uid": null
    },
    "parent": "movies",
    "view": "AwStandardSublocation"
  },
  "myRecent": {
    "url": "/myRecent",
    "data": {
      "label": {
        "source": "/i18n/locationAndSublocationMessages",
        "key": "recentSubLocationTitle"
      },
      "nameToken": "com.siemens.splm.client.movies:myRecentSubLocation",
      "clientScopeURI": "",
      "viewBase": "favorites"
    },
    "parent": "movies",
    "view": "AwStandardSublocation"
  }
}
```

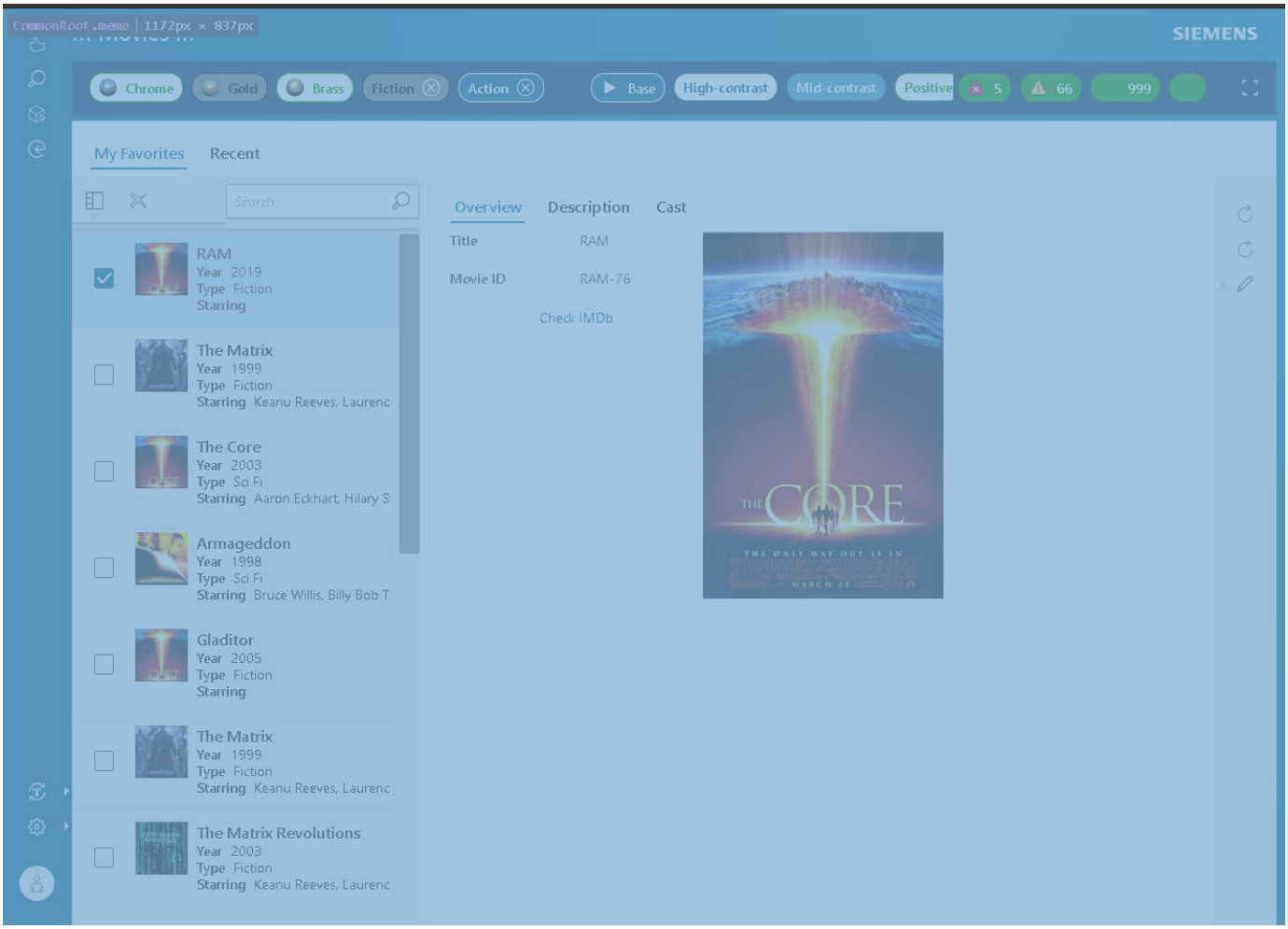
When URL is mapped to /myFavs:



Unpublished work. © 2025 Siemens



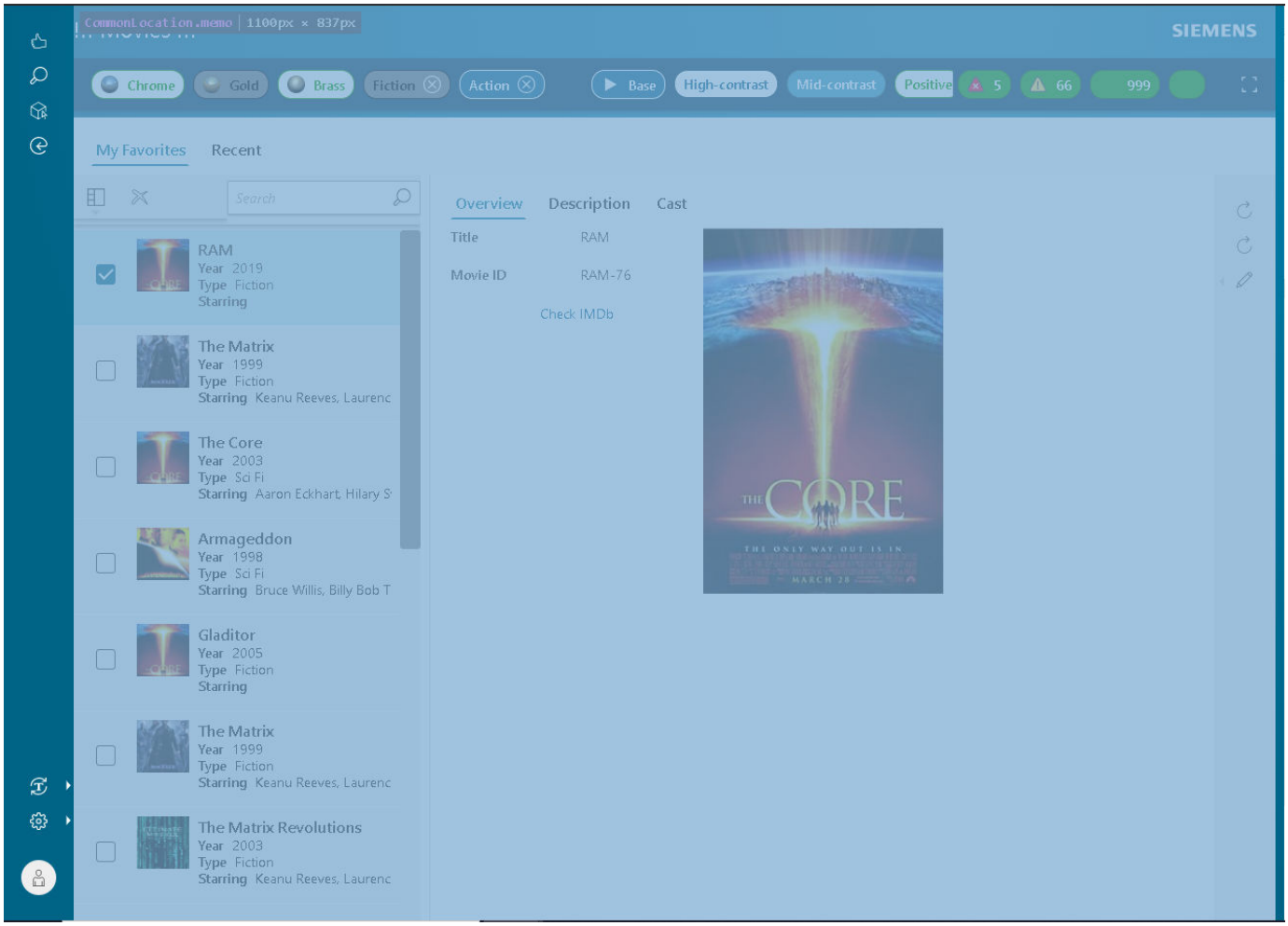

**Common Root - Defined in SWF**



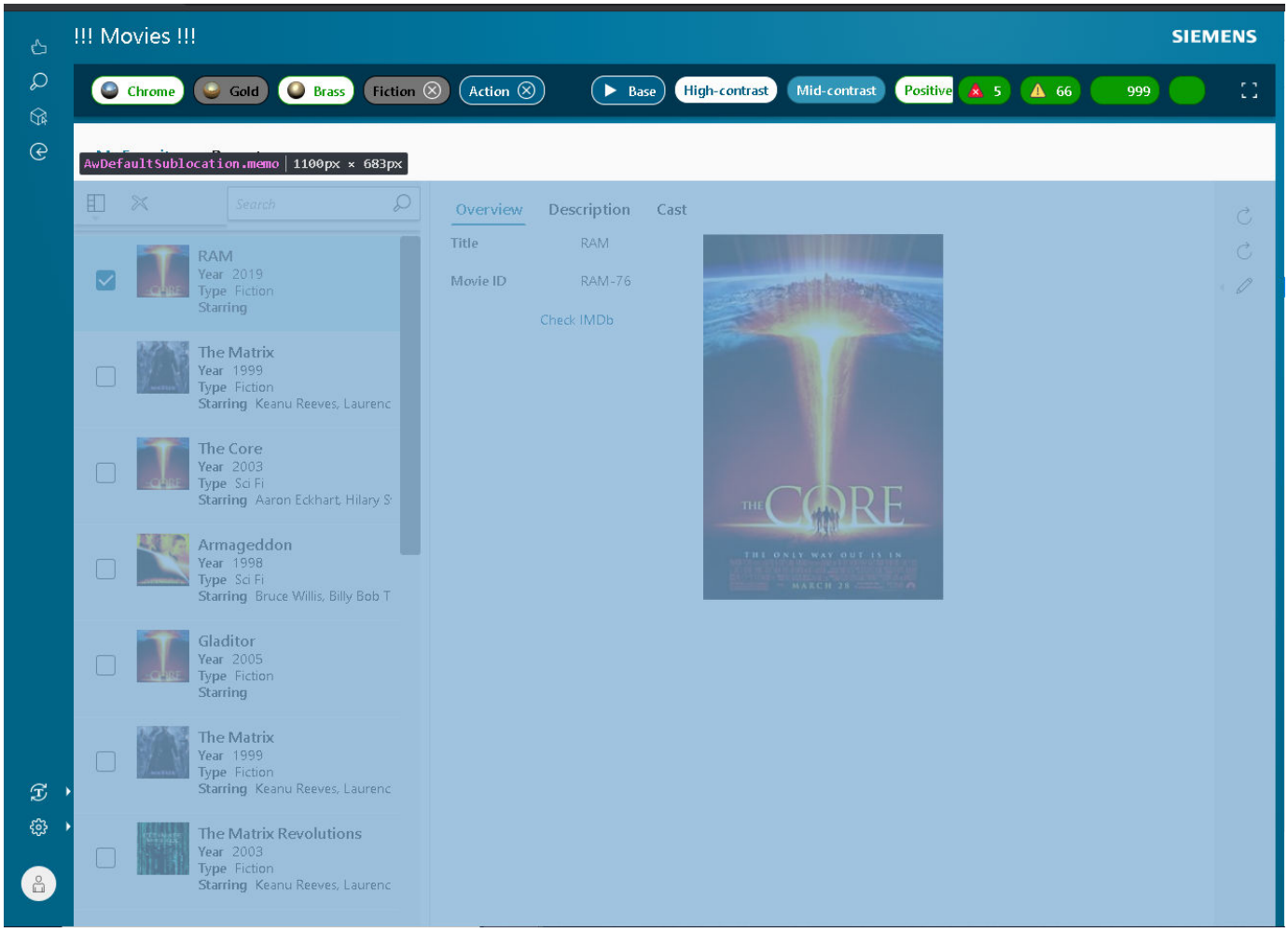

**Location Component - commonLocation**



Unpublished work. © 2025 Siemens




**Sublocation Component - AwStandardSublocation**




# Layout Elements

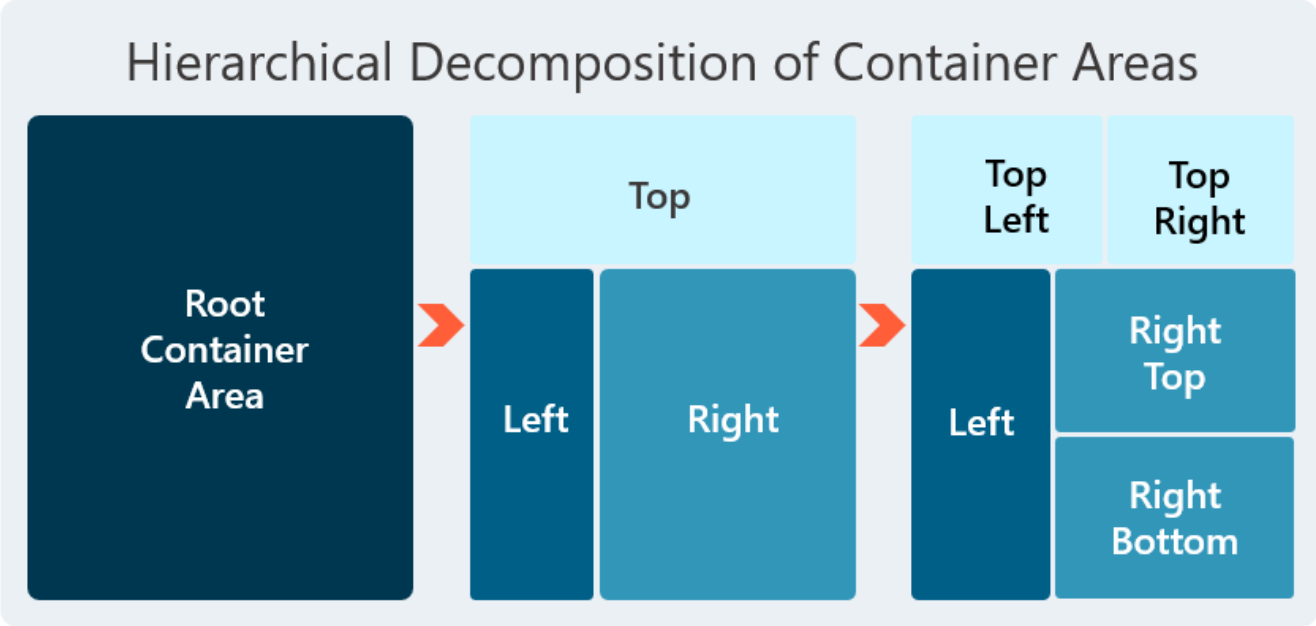
Declarative UI is a capability provided by the Siemens Web Framework Framework that allows for a concise and codeless definition of a system's user interface. A key goal of the Declarative UI is to provide a simple View Markup language that is semantically meaningful. For more information see: **Declarative UI Wiki**

This document defines the syntax and semantics of a subset of elements in the Declarative UI used to define the layout of other Declarative UI elements. The goal is to provide customized View Markup language elements that are simple and semantically meaningful. The custom elements should be layered such that for a given element/level the designer only sees its layout. The custom elements encapsulate and hide implementation details from the designer.

These Layout Elements define the position and size of container areas within the overall system display area. Other elements, generally referred to here as Content Elements, are expected to fill an entire container area.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Container areas are defined in a hierarchical fashion. That is, each container area can be decomposed into smaller container areas using the Layout Elements. This hierarchical breakdown of container areas continues until an area is defined that contains one or more “atom" elements (e.g. <aw-button\>, <aw-textarea\>), or contains one complex custom element (e.g. <aw-panel-body\>). Such complex elements define the layout for their content within their parent container area. See Section 2.4 for more information on the relationship between Layout Elements and Content Element.




## Overview

This section provides a high level description of the Layout Elements and discusses the common concepts that are used to define container areas. The element specifications in Section 3 assume the reader is familiar with these common concepts.

## Layout Elements

The Declarative UI Layout Elements consist of the follow four primary tags (1-4 below) and an additional tag (5 below) for testing:

1. <aw-row\> - This element is used to decompose the parent container area into smaller areas which are defined by the entire width of the parent container area and a given height for the new area. This element is defined in section 3.1.1.
2. <aw-column\> - This element is used to decompose the parent container area into smaller areas which are defined by the entire height of the parent container area and a given width for the new area. This element is defined in section 3.1.2.


Unpublished work. © 2025 Siemens

3. <aw-splitter\> - This element is used to define a standard user control that is positioned horizontally or vertically between two container areas. The user may use this control to dynamically resize the two areas and all sub-areas defined within those area. This element is defined in section 3.1.3.
4. <aw-include\> - This element is used to combine layouts defined in multiple data sets into a single layout specification. It also includes optional attributes for selecting different layouts for different target devises or display area sizes. These options are used to implement responsive UI designs. This element is defined in section 3.2.1.
5. <aw-dummy-content\> - This element is ONLY used for testing while developing a new layout. It allows the layout designer to add simple content to an area to help visualize the layout. This element is defined in section 3.3.1.

## Container Areas

Container areas are hierarchically decomposed. This means every container area defines a subarea of the area it is within (called the parent container or parent area). At the top level the container area is defined by the application and is typically a sub-area of the system display window.

A top level container area is called the “root" container area. A root container area is positioned and sized by the hosting application. The Declarative layout elements are then used to define the positions and sizes of the hierarchical decomposition of the root container areas. An application is free to define as many root container areas as needed. The decomposition of each root container is considered a different independent layout.

The following concepts apply to all container areas (with the exception of the root container which is fully controlled by the hosting application):

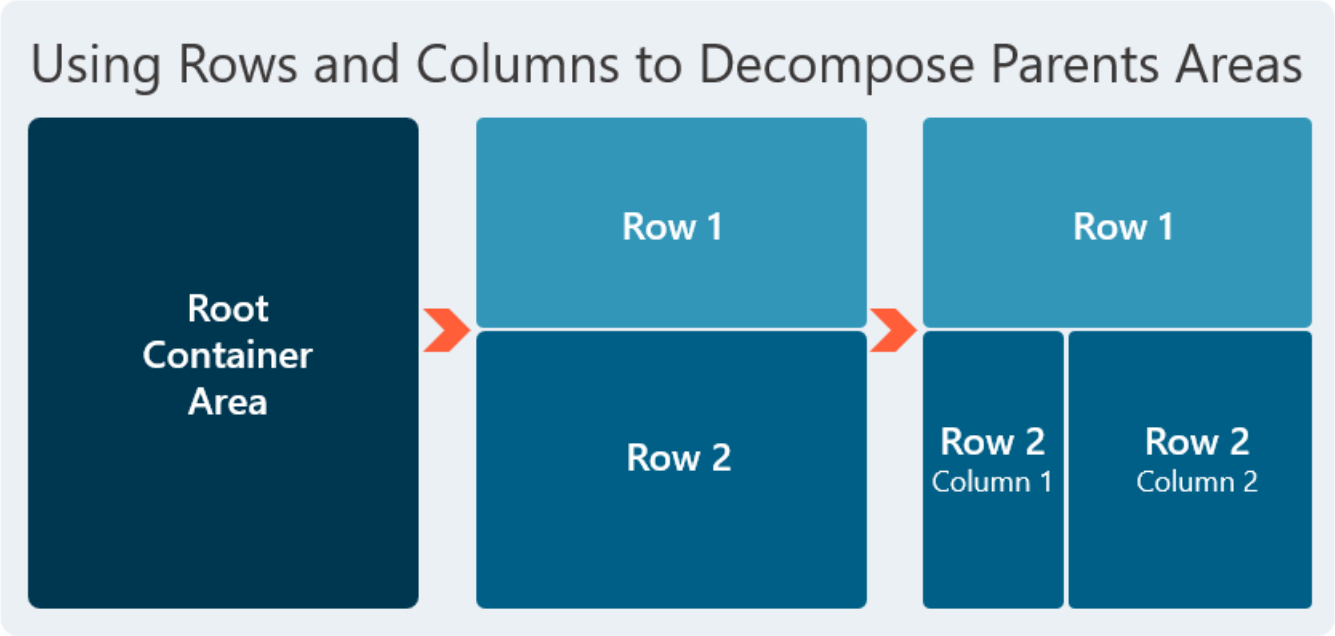
- Each container area is always fully within its parent area. Therefore container areas have no concept of view ports, scale, pan or scrolling. Any such operations are defined by the Content Elements within a container area.
- Container areas within a parent container do not overlap with each other. The application can define as many layouts as desired and may position those layouts however it wants (e.g. on top of each other). But each layout has no knowledge of such layering.
- Any area not defined within a parent area is assumed to contain no content (furthermore the Declarative UI has no mechanism to define content for these undefined areas). These empty areas are transparent and will display whatever is displayed in that area by the parent container.
- Container areas do not dynamically move. A standard splitter control may be defined to dynamically resize two adjacent areas. But the relative locations of the areas do not change and all of the above rules always apply. To dynamically move an area the application will load a new layout.

## Container Decomposition

A parent container area is decomposed into sub-area by either defining a set of rows or columns for the parent container. Each row or column within a parent container defines a sub-area which can be further


Siemens Web Framework
Unpublished work. © 2025 Siemens

decomposed using rows or columns where the new sub-area is now the parent container. Note that each parent container is decomposed with either a set of rows or a set of columns, not both.



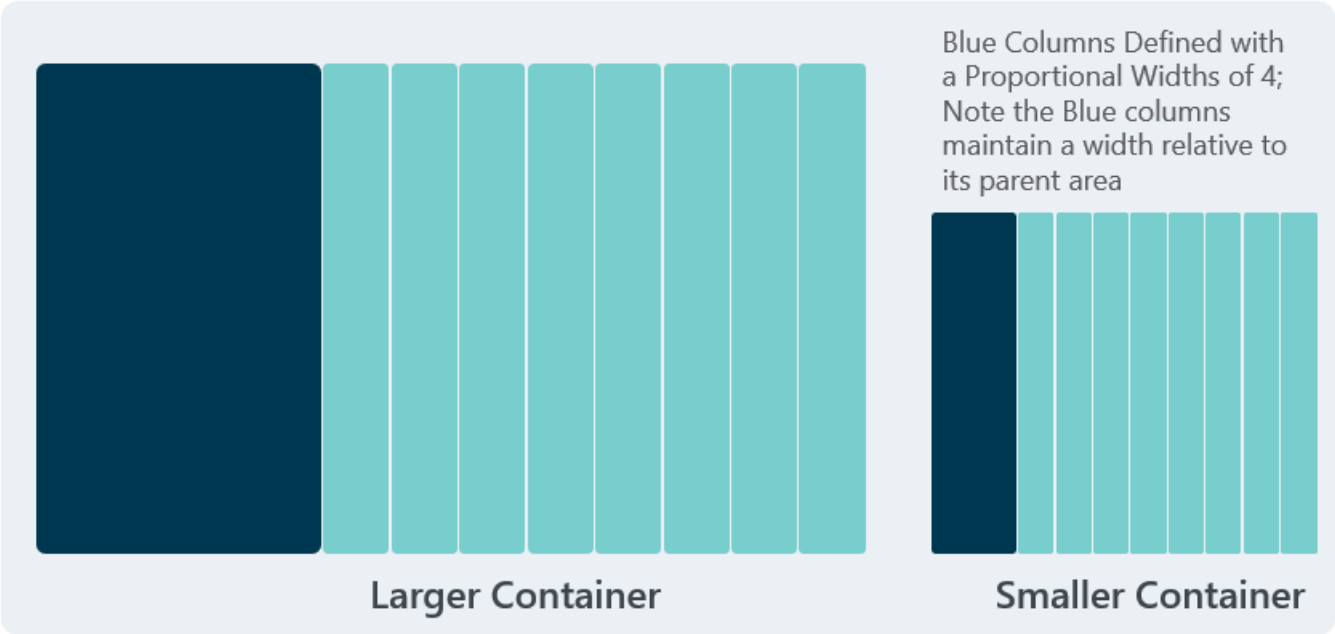

Rows are the same width as the parent container. The row element defines the height of the sub-area. For the purposes of this document the “size" of a row is given by the row’s height. Columns are the same height as the parent container. The column element defines the width of the sub-area. For the purposes of this document the “size" of a column is given by the column’s width.

The size of rows and column may be specified with proportional values or with fixed values or with the “fill" option. Rows and columns defined within a parent area may be specified with different types of values (proportional or fixed or “fill").

## Proportional Values

A parent area may be sub-divided by using rows or columns which have a size defined by a proportional value. Areas defined with proportional values will always have the same relative size regardless of the parent container size. If the parent container changes size for any reason then the row or columns defined for that area with proportional values update to their new relative size.

Proportional values are defined with integers between 1 and 12. A value of N means that the size of that row or column is **(N / 12)** of the size of the parent container. The values for all rows or columns within a parent area must sum to a value less than or equal to 12. If the values sum to less than 12 then the areas not specified are empty.




Note that the value 12 was selected because many layout tools are defined using a 12 column grid system. It has been found that sub-dividing areas in this fashion produce pleasing proportional layouts. For more information see: **Grid Theory** and **Golden Ratio in Web Designs**

## Fill Values

Optionally an area's size may be set to "fill". This means that the area will consume whatever space has not been defined by other areas. If multiple areas within the same row or column are defined using the "fill" option then the remaining space is divided equally between all of the fill area.

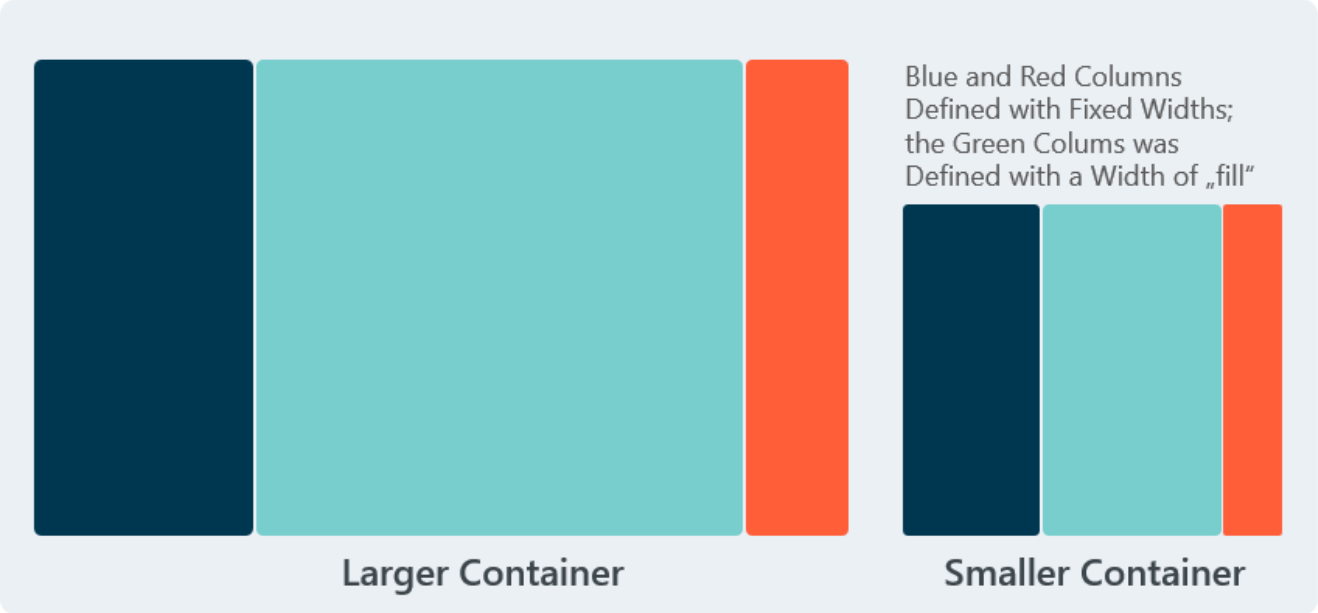
## Fixed Values

Optionally fixed values may be used to define row or column sizes. A fixed value is defined by an integer followed immediately by an "f". For instance, <aw-row width="4f"\> would mean that the row is 4 times a standard fixed width. So if the standard fixed width is 8 pixels then width="4f" would mean that the row is 32 pixels wide (See Section 2.3.3.1 for more information on standard fixed units).

When a parent area is resized for any reason then the rows or columns within that area defined with fixed values will remain the same size. Rows or columns defined with proportion values will then be proportional relative to the remaining area. Rows or columns defined with the "fill" option would then consume any undefined areas. If there are no fill areas and if the sum of the fixed and proportional values do not fully consume the space of the parent container then the areas not specified are empty.

If a parent area is resized smaller than the defined fixed sub-areas then the sub-areas will be truncated on the right and/or bottom to fit within the available space. Note that the proportional and "fill" areas would have first been reduced to the smallest possible size.


Siemens Web Framework
Unpublished work. © 2025 Siemens




Note the Blue and Red columns are the same width regarless of the size of the parent area

## Fixed Value Units

Standard fixed width and fixed height values are defined to be “1em". EM units are relative to the font size of the current area. That means, if the font size is 8px then “2f" results in a size of 16px. Note that by definition “em" units cascade through the layout hierarchy. Using EM units consistently within a layout allows the top level font size to proportionally control sizes of all areas defined with fixed units. This allows the application to change the font size at the top level (for instance to support different devices sizes) and then to have all areas within the layout automatically scale proportionally for that size.

## Fixed Value Usage Notes

Fixed values are used to support rows or columns that contain fixed size content. For instance, if a row is to contain header text or a column is to contain a toolbar then those sizes will be based on fixed font sizes or button sizes. Using proportional values for fixed sized content would be difficult because a layout never knows the actual size of the parent container.

In practice it is expected that a published list of fixed sizes will exist for specific types of content.

For example, a published value of `5f` could be the size required for Main Header Text. The approach of using multiples of some standard fixed size was adopted to avoid being locked into a fixed set of key words for standard sizes. This does not prevent us from defining synonyms for standard fixed sizes as needed in the future.

Fixed values should only be used when absolutely necessary due to the fixed size of the area's content (such as fixed sized buttons or font sizes). Using fixed sized areas makes it difficult to align elements in


Unpublished work. © 2025 Siemens

different areas and does not easily yield pleasing proportional layouts. Whenever possible, areas should be sized using the proportional value option.

## Content Elements

There are two types of Content Elements: 1) atom element which do not contain or reference other Declarative UI elements, and 2) Container Elements which may contain or reference other Declarative UI elements. Container Elements should not be confused with Container Areas. A Container Area defines an area in the display window that has a location, width and height. A Container Element, as defined below, defines the content for such an area.

Both types of Content Element are assumed to fill their entire parent container. They will reference standard padding and margin values for positioning within their parent container. They will define any required viewing operations such as scale and pan. They will define when scroll bars are used. They will fully define how they are positioned within their parent container.

Both types of Content Elements are also responsible for defining the internal layout of all elements that are used in their implementation. The internal layout is entirely encapsulated and hidden from its parent container.

The only difference is that future Container Elements may consider using Layout Elements to define the layout of the Declarative UI Elements that are referenced by the Container Elements. This will most likely require an enhancement to the current View Model for Container Elements but it is not expected to impact the definition of the Layout Elements.

## Layout Element Specifications

The following sections define the syntax and semantics of the Layout Elements and their attributes. Examples showing how the elements are combined together to define container areas are shown in Section 4.

Notes:

- All tags must be terminated with a closing tag. For instance:

```
<aw-row height="4"> . . . </aw-row>
```

- Layout Elements are custom HTML elements and use HTML syntax. Comments in HTML use the following syntax: `<!-- comment -->`

  For more information see: **HTML Syntax**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Container Decomposition Elements

This section defines the Layout Elements used to decompose a parent container area into rows or columns. It also defines the Layout Element used to define a standard splitter control to resize two adjacent rows or columns.

### <aw-row\>

The row element defines a sub-area within the parent area that is the full width of the parent area and a height defined by the row element. The vertical position of the row is determined by its attributes and any rows previously defined for the parent area. UI element library - Please refer to `aw-row` example in **Siemens Web Framework Storybook**.

## Row Attributes

| Attribute | Description |
|---|---|
| `height` | Defines the height of the row using proportional, fixed or “fill" value types as discussed in section 2.3. Optional: If not specified the value defaults to: height="1" |
| `justify` | Defines the vertical position of the row within the parent area. The possible values area: “top", “bottom", or “center". **Optional:** If not specified the value defaults to: justify="top" |
| `offset` | Defines an empty row above or below this row. When justify="top" or “center" the empty row is above this row. When justify="bottom" the empty row is below this row. **Optional:** If not specified then no empty row is defined. |
| `id` | Defines an optional name for this row. This name can be referenced in other layouts and is used to help transition from one layout to another. The rows with the same name in two transitional layouts are assume to represent the same container but with possibly different locations and sizes. The system can then use this information to transition between two layouts using different forms of animation. **Optional:** If not specified then no transitions can occur for this row. |
| `color` | Defines a background color for the row using any CSS color value. This attribute is only used during development to help visualize the layout. The actual background color for the area is specified by the area’s content. **Optional:** If not specified the value defaults to: color="transparent" |
| `wrap-style` | Defines a wrap style mechanism for the row. Possible wrap style values are "wrap", "nowrap" and "wrap-reverse". **Optional:** If not specified the value defaults to: "wrap" |

## Possible Row Content

A row may contain one of the following elements sets. The content cannot contain a mixture of elements from different sets.

Element Sets

- A list of <aw-row\> tags.
- A list of <aw-column\> tags.
- A single Container Element tag (e.g. <aw-panel\>).
- A list of atom tags (e.g. <aw-button\>). These elements are stacked horizontally within the row.
- A single <aw-include\> tag.
- Empty

## <aw-column\>

The column element defines a sub-area within the parent area that is the full height of the parent area and a width defined by the column element. The horizontal position of the column is determined by its attributes and any columns previously defined for the parent area. UI element library - Please refer to aw-column example in Siemens Web Framework Storybook.

## Column Attributes

| Attribute | Description |
|---|---|
| width | Defines the width of the column using proportional, fixed or “fill" value types as discussed in sections 2.3. Optional: If not specified the value defaults to: width="1" |
| justify | Defines the horizontal position of the column within the parent area. The possible values area: “left", “right", or “center". **Optional:** If not specified the value defaults to: justify="left" |
| offset | Defines an empty column to the left or right of this column. When justify="left" or “center" the empty column is to the left this column. When justify="right" the empty column is to the right of this column. Optional: If not specified then no empty column is defined. |
| id | Defines an optional name for this column. This name can be referenced in other layouts and is used to help transition from one layout to another. The columns with the same name in two transitional layouts are assume to represent the same container but with possibly different locations and sizes. The system can then use this information to transition between two layouts using different forms of animation. **Optional:** If not specified then no transitions can occur for this row. |
| color | Defines a background color for the column using any CSS color value. This attribute is only used during development to help visualize the layout. The actual background color for the area is specified by the area’s content. **Optional:** If not specified the value defaults to: color="transparent" |
| wrap-style | Defines a wrap style mechanism for the row. Possible wrap style values are "wrap", "nowrap" and "wrap-reverse". **Optional:** If not specified the value defaults to: "wrap" |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Possible Column Content

A column may contain one of the following elements sets. The content cannot contain a mixture of elements from different sets. Note that this is the same list available for rows.

Element Sets

- A list of <aw-row\> tags.
- A list of <aw-column\> tags.
- A single Container Element tag (e.g. <aw-panel\>).
- A list of atom tags (e.g. <aw-button\>). These elements are stacked vertically within the column.
- A single <aw-include\> tag.
- Empty

## <aw-splitter\>

The splitter element defines a standard user control that allows the user to dynamically resize adjacent rows or columns and all of the rows and columns in the sub-areas. A splitter element can only be placed between any two elements. When placed between two row elements a standard horizontal control is created to allow the user change the heights of the adjacent rows. When placed between two column elements a standard vertical control is created to allow the user change the widths of the adjacent columns. After a user resizes a row or column it will maintain the proportional or fixed sized behavior as given by its original definition. Of course proportional values may no longer be multiples of 1 12th of the parent area size but they will still update as proportionally defined areas. UI element library - Please refer to `aw-splitter` example in **Siemens Web Framework Storybook**.

## Splitter Attributes

| Attribute | Description |
|---|---|
| `min-size-1` | Defines the minimum size of the previous element before <aw-splitter\> in pixels. **Optional:** If not specified there is a default minimum size of 20 pixels. |
| `min-size-2` | Defines the minimum size of the next element after <aw-splitter\> in pixels. **Optional:** If not specified there is a default minimum size of 20 pixels. |
| `direction` | Direction parameter is optional and only needed if NOT between an <aw-row\> or <aw-column\>. (e.g. "vertical" or “horizontal"). **Optional:** It will default to vertical if not set and when used other than between an <aw-row\> or <aw-column\>. |

## Possible Splitter Content

A splitter element has no content.

## Layout Reference Element

This section defines an element used to combine layouts from multiple data sets into a single layout specification.

`Note` that this section refers to data sets. The actual implementation will require the name values to reference a logically paired View and a View Model defined within the **Declarative UI** framework. The framework's implementation will find and demand load the View and View Model and will insert the layout definition into the parent container. How (GET, REST, SOAService) and where (URL) the View and View Model artifacts are retrieved is configured by the "viewAndViewModelRepoConfiguration" configuration in an Siemens Web Framework Solution's kit.json (See the **Getting Started with Siemens Web Framework** guide and Siemens Web Framework documentation for details).

## `<aw-include\>`

The layout element is used to define the layout for the current parent container when the layout is defined in a different data set. UI element library - Please refer to `aw-include` example in **Siemens Web Framework Storybook**.

## Layout Attributes

| Attribute | Description |
|---|---|
| `name` | Defines a reference to a data set containing a layout definition. **Required:** A layout tag must reference a data set that contains the content for the parent area. |
| `when` | Defines a list of one or more condition name and layout name pairs. Each pair is separated by a comma (","). The condition name and layout name are separated by a colon (":"). For instance, the following example defines two condition and layout name pairs. The first pair has a condition named "c1" and a layout named "n1". The second pair has a condition named "c2" and a layout named "n2". when = "c1:n1, c2:n2" Each condition is tested in order. The layout named for the first condition that is true is selected and defines the layout for the current container area. If no condition evaluates to true then the layout named with the required "name" attribute defines the layout for the current container area. This means that when the "when" attribute is present, the "name" attribute defines the default layout. Named conditions can be used to select a layout based on the platform type or the parent container area size (or any other criteria we wish to implement). For instance, the grid system defined by ZURB Foundation (see **Zurb Foundation - Default Media Queries**) uses the named conditions: "Small", "Medium" and "Large". These names mean that the layout identified for "Large" is selected when the area size is >=1024, else the "Medium" layout is selected when the area size >=640 and if those conditions are not met then the "Small" layout is selected. `The set of named conditions are still to be defined.` **Optional:** If not specified then the named layout is always used. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Possible Layout Content

The <aw-include\> element has no content. But it does reference a data set that can contain one of the following elements sets. The content of the referenced data set cannot contain a mixture of elements from different sets. Note that this is the same list available for rows and columns.

Element Sets

- A list of <aw-row\> tags.
- A list of <aw-column\> tags.
- A single Container Element tag (e.g. <aw-panel\>).
- A list of atom tags (e.g. <aw-button\>). These elements are stacked vertically within the column.
- A single <aw-include\> tag.
- Empty

## Layout Development Element

This section defines an element only used during the development of a new layout. It is used to help visualize the layout and would be removed before actual deployment of the layout to an application.

## <aw-dummy-content\>

The dummy content element is used to define simple content for the current area.

## Dummy Content Attributes

| Attribute | Description |
|---|---|
| image | Defines the name of an image file (including the suffix). The image will stretch to fill the content area. Any file type support by the html tag <img\> may be used. The actual location of the file is defined by the specific implementation of the layout elements. **Optional:** If the image or text attributes are not given then default text is displayed as the content. |
| text | Defines a text string that is centered within the content area. |

Optional: If the image or text attributes are not given then default text is displayed as the content. |

Only one attribute should be used at a time. If both are given then the image attribute is used and the text attribute is ignored.

This element cannot have any content

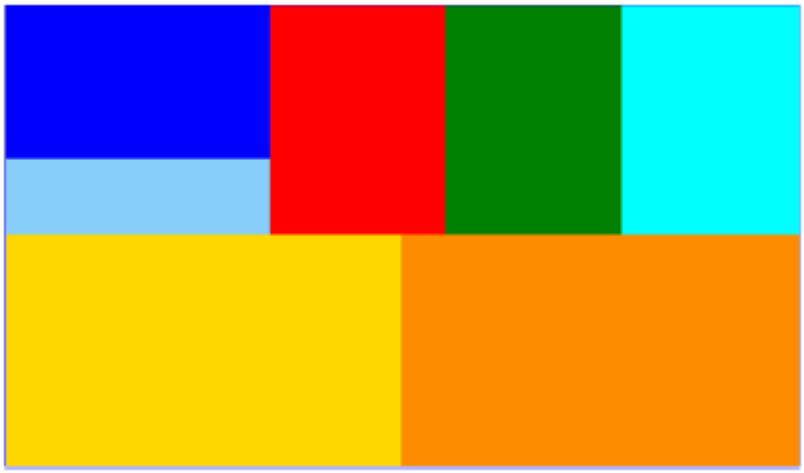

Unpublished work. © 2025 Siemens

## Examples

This section shows examples of how the Layout Elements and their Attributes combine together to define layout specifications. In these examples the optional color attribute is used to show the resulting area decomposition. The color attribute would not be used for actual layouts, since the content of those areas will define the display. The examples here focus on showing how the Layout Elements and their Attributes work, using easy to understand layouts. For examples using real SPLM application layouts see: Examples

## Proportional Values

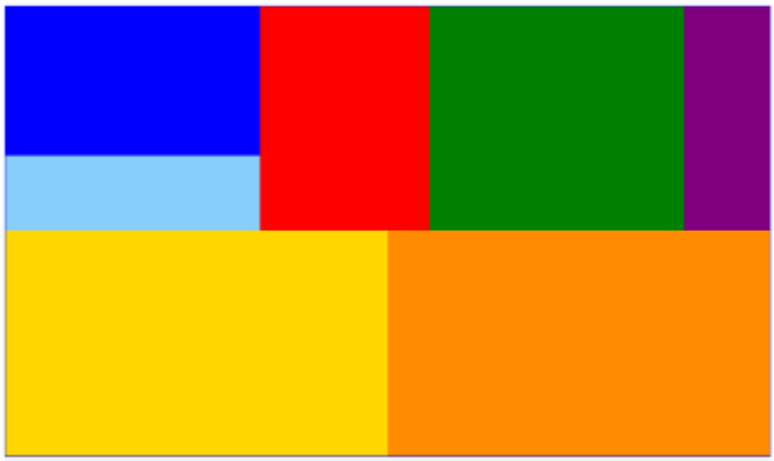
This example shows proportional values with nested rows and columns to decompose a container area. Also, shown is the ability to nest columns within columns, which also works for nesting rows within rows.



```
<aw-row height="6">
    <aw-column width="4">
        <aw-row height="8" color="blue"></aw-row>
        <aw-row height="4" color="lightskyblue"></aw-row>
    </aw-column>
    <aw-column width="8">
        <aw-column width="4" color="red"></aw-column>
        <aw-column width="4" color="green"></aw-column>
        <aw-column width="4" color="cyan"></aw-column>
    </aw-column>
</aw-row>
<aw-row height="6">
    <aw-column width="6" color="gold"></aw-column>
    <aw-column width="6" color="darkorange"></aw-column>
</aw-row>
```

Here is a similar example except one set of columns do not sum to a total width of 12. In this case the undefined area is empty and shows the content of the parent container, which is the purple column with width="8".


Siemens Web Framework
Unpublished work. © 2025 Siemens



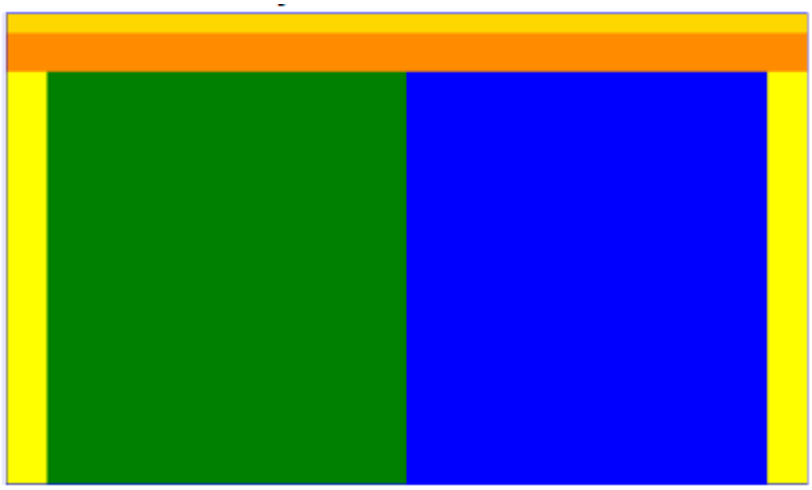
```
<aw-row height="6">
    <aw-column width="4">
        <aw-row height="8" color="blue"></aw-row>
        <aw-row height="4" color="lightskyblue"></aw-row>
    </aw-column>
    <aw-column width="8">
        <aw-column width="4" color="red"></aw-column>
        <aw-column width="6" color="green"></aw-column>
    </aw-column>
</aw-row>
<aw-row height="6">
    <aw-column width="6" color="gold"></aw-column>
    <aw-column width="6" color="darkorange"></aw-column>
</aw-row>
```

## Fixed Values

This example shows a top level parent area divided into three rows using fixed values. The bottom row then has three columns also using fixed values. Finally, the middle column in the bottom row has two columns divided into equal areas using proportional value.

Actual standard fixed width and height values are still to be defined.

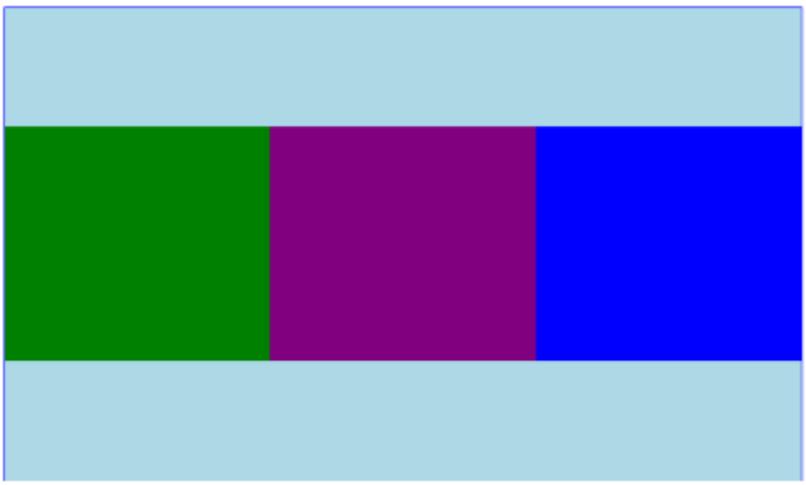

Unpublished work. © 2025 Siemens



```
<aw-row height="2f" color="gold"></aw-row>
<aw-row height="4f" color="darkorange"></aw-row>
<aw-row height="fill">
    <aw-column width="5f" color="yellow"></aw-column>
    <aw-column width="fill">
        <aw-column width="6" color="green"></aw-column>
        <aw-column width="6" color="blue"></aw-column>
    </aw-column>
    <aw-column width="5f" color="yellow"></aw-column>
</aw-row>
```

## Justification

This example shows a blue column that has been justified to the right. The green column has used the default left justification. Note that the purple area is empty and is showing the color of the parent row.

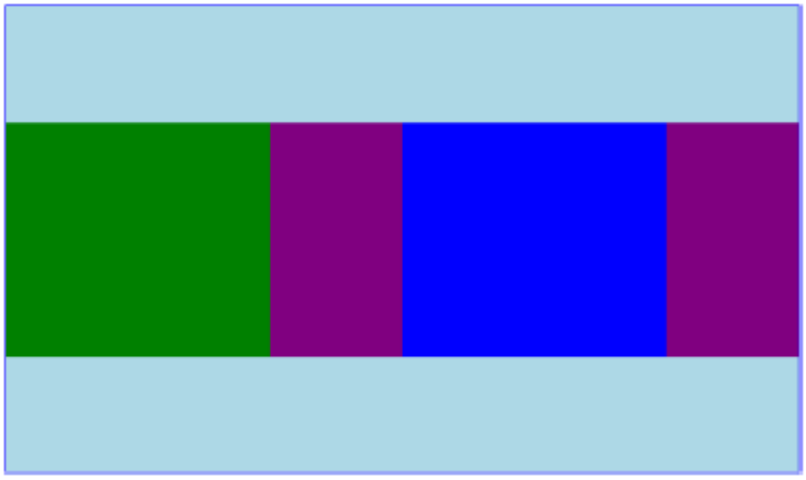


```
<aw-row height="3" color="lightblue"></aw-row>
<aw-row height="6" color="purple">
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    <aw-column width="4" color="green"></aw-column>
    <aw-column width="4" color="blue" justify="right"></aw-column>
</aw-row>
<aw-row height="3" color="lightblue"></aw-row>
```

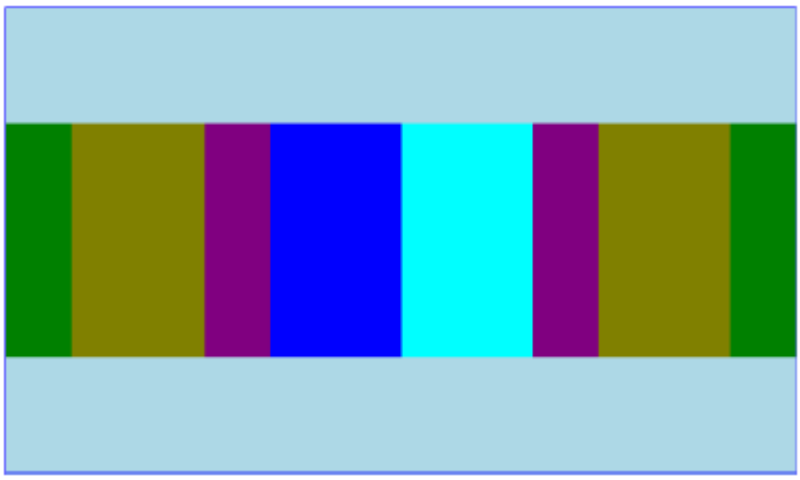
In this example the blue column has been justfied to the center. Note that it is centered in the area that remains after any left or right columns are have been justfied. The purple areas on the left and right of the blue column are empty.



```
<aw-row height="3" color="lightblue"></aw-row>
<aw-row height="6" color="purple">
    <aw-column width="4" color="green"></aw-column>
    <aw-column width="4" color="blue" justify="center"></aw-column>
</aw-row>
<aw-row height="3" color="lightblue"></aw-row>
```

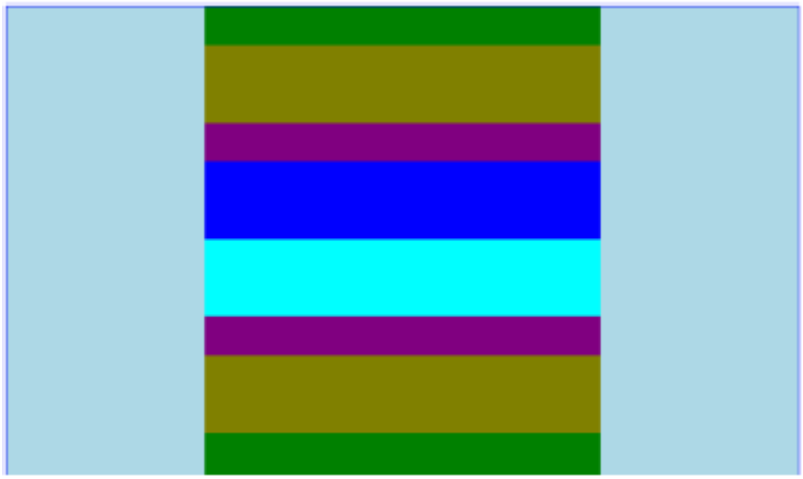
In this more complex example two columns are using the default left justification and two similarly colored columns are justified to the right. Then two columns are centered in the remaining area. The area not defined is empty and shows the parent row color of purple.

Note that rows and columns must be in order of their justification (for columns: left, center, right) or (for rows: top, center, bottom).



```
<aw-row height="3" color="lightblue"></aw-row>
<aw-row height="6" color="purple">
    <aw-column width="1" color="green"></aw-column>
    <aw-column width="2" color="olive"></aw-column>
    <aw-column width="2" color="blue" justify="center"></aw-column>
    <aw-column width="2" color="cyan" justify="center"></aw-column>
    <aw-column width="2" color="olive" justify="right"></aw-column>
    <aw-column width="1" color="green" justify="right"></aw-column>
</aw-row>
<aw-row height="3" color="lightblue"></aw-row>
```

This example shows the same layout as the previous example except the row and columns have been switched. Here the default justification is top and two similar rows have been justified to the bottom. Again the purple area is empty and shows the color of the parent column.



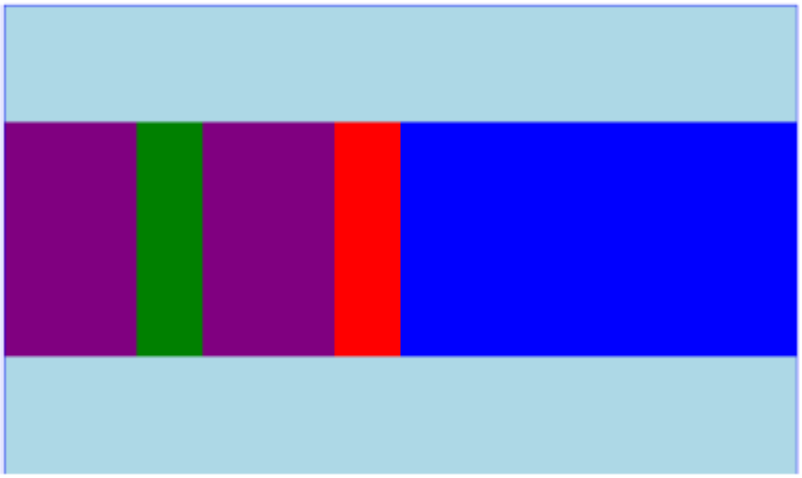
```
<aw-column height="3" color="lightblue"></aw-column>
<aw-column height="6" color="purple">
    <aw-row width="1" color="green"></aw-row>
    <aw-row width="2" color="olive"></aw-row>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    <aw-row width="2" justify="center" color="blue"></aw-row>
    <aw-row width="2" justify="center" color="cyan"></aw-row>
    <aw-row width="2" justify="bottom" color="olive" ></aw-row>
    <aw-row width="1" justify="bottom" color="green"></aw-row>
</aw-column>
<aw-column height="3" color="lightblue"></aw-column>
```

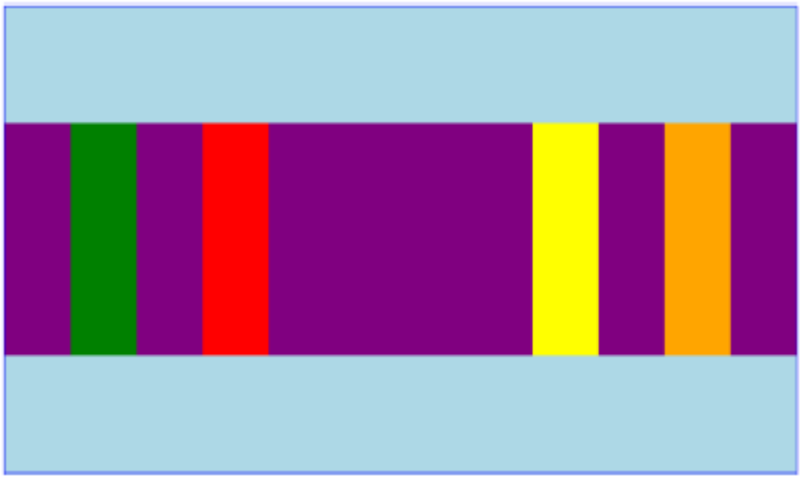
## Offsets

This example shows a green column that has been offset by an empty column of width 2. It is then followed by a red column that has also been offset by the same amount. The offset areas are empty and show the color parent row (purple). Offsets are a shorthand method for defining an empty row or column. Note that when using proportional values the width or height values PLUS the offset values must be less than or equal to 12.



```
<aw-row height="3" color="lightblue"></aw-row>
<aw-row height="6" color="purple">
    <aw-column width="1" offset="2" color="green"></aw-column>
    <aw-column width="1" offset="2" color="red" justify="right"></aw-column>
    <aw-column width="6" color="blue"></aw-column>
</aw-row>
<aw-row height="3" color="lightblue"></aw-row>
```

This example shows how offsets work with justification. When justified to the left or center the offsets are to the left of the column being defined. When justified to the right the offsets are to the right of the column being defined. In this example the yellow column that is justified right is listed first in the code, so it is shown first on the right. It has an offset of 1 which is the empty puple column on its right. The orange column follows in the code, so it is shown next on the right with its offset also on the right.



```
<aw-row height="3" color="lightblue"></aw-row>
<aw-row height="6" color="purple">
    <aw-column width="1" offset="1" color="green"></aw-column>
    <aw-column width="1" offset="1" color="red"></aw-column>
    <aw-column width="1" offset="1" justify="right" color="yellow"></aw-
column>
    <aw-column width="1" offset="1" justify="right" color="orange"></aw-
column>
</aw-row>
<aw-row height="3" color="lightblue"></aw-row>
```

This example shows the same layout as the previous example except the row and columns have been switched. Here the last two rows have been justified to the bottom.

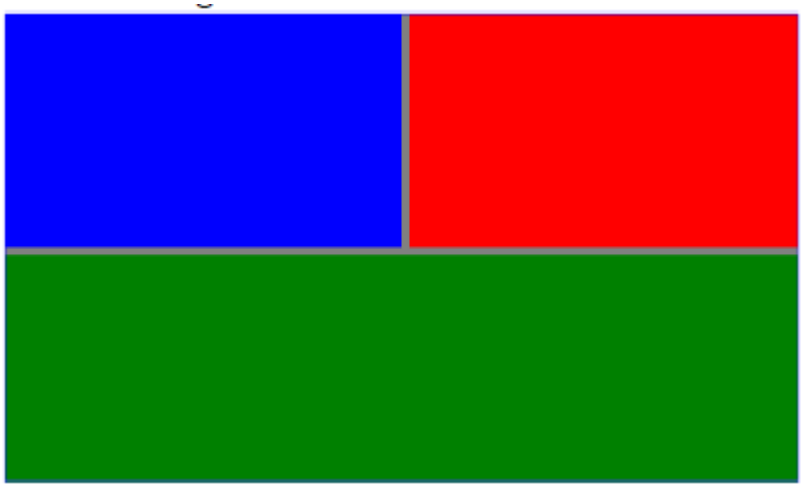
```
<aw-column height="3" color="lightblue"></aw-column>
<aw-column height="6" color="purple">
    <aw-row width="1" offset="1" color="green"></aw-row>
    <aw-row width="1" offset="1" color="red"></aw-row>
    <aw-row width="1" offset="1" justify="bottom" color="yellow"></aw-row>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    <aw-row width="1" offset="1" justify="bottom" color="orange"></aw-row>
</aw-column>
<aw-column height="3" color="lightblue"></aw-column>
```

## Splitters

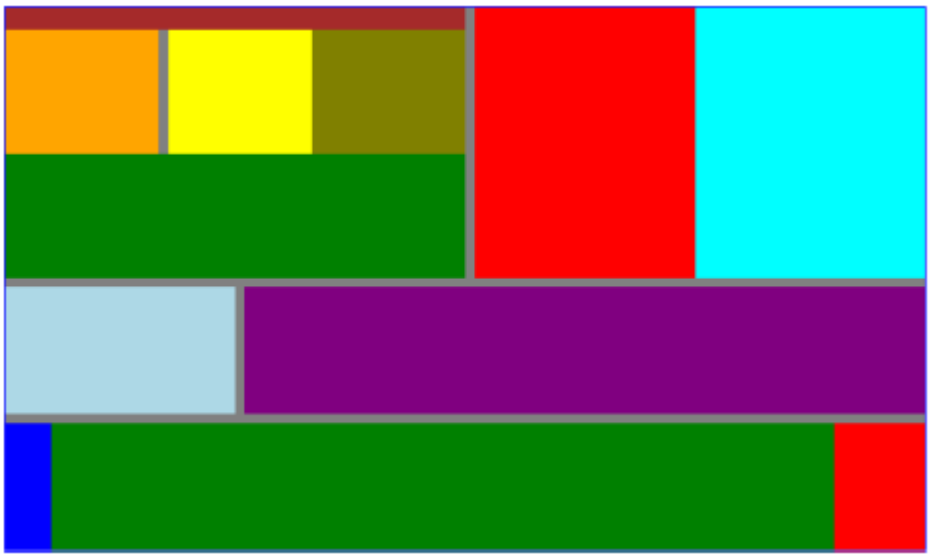
This example shows how to define two simple splitters. A vertical splitter is placed between the blue and red columns (the gray bar). And a horizontal splitter is placed between the top row and the bottom green row. Note that splitter's will actually be transparent and thus will display whatever is in the container that contains the elements being resized.



```
<aw-row height="3" color="blue">
    <aw-column width="6" color="blue"></aw-column>
    <aw-splitter></aw-splitter>
    <aw-column width="6" color="red"></aw-column>
</aw-row>
<aw-splitter></aw-splitter>
<aw-row height="6" color="green"></aw-row>
```

This shows the same rows and columns after the user has resized the areas using the splitter controls.

Here is a more complex example showing three vertical splitters and two horizontal splitters:



```
<aw-row height="6" color="blue">
    <aw-column width="6">
        <aw-row height="2f" color="brown"></aw-row>
        <aw-row height="fill">
            <aw-row height="6">
                <aw-column width="4" color="orange"></aw-column>
                <aw-splitter></aw-splitter>
                <aw-column width="4" color="yellow"></aw-column>
                <aw-column width="4" color="olive"></aw-column>
            </aw-row>
            <aw-row height="6" color="green"></aw-row>
        </aw-row>
    </aw-column>
    <aw-splitter></aw-splitter>
    <aw-column width="3" color="red"></aw-column>
    <aw-column width="3" color="aqua"></aw-column>
```

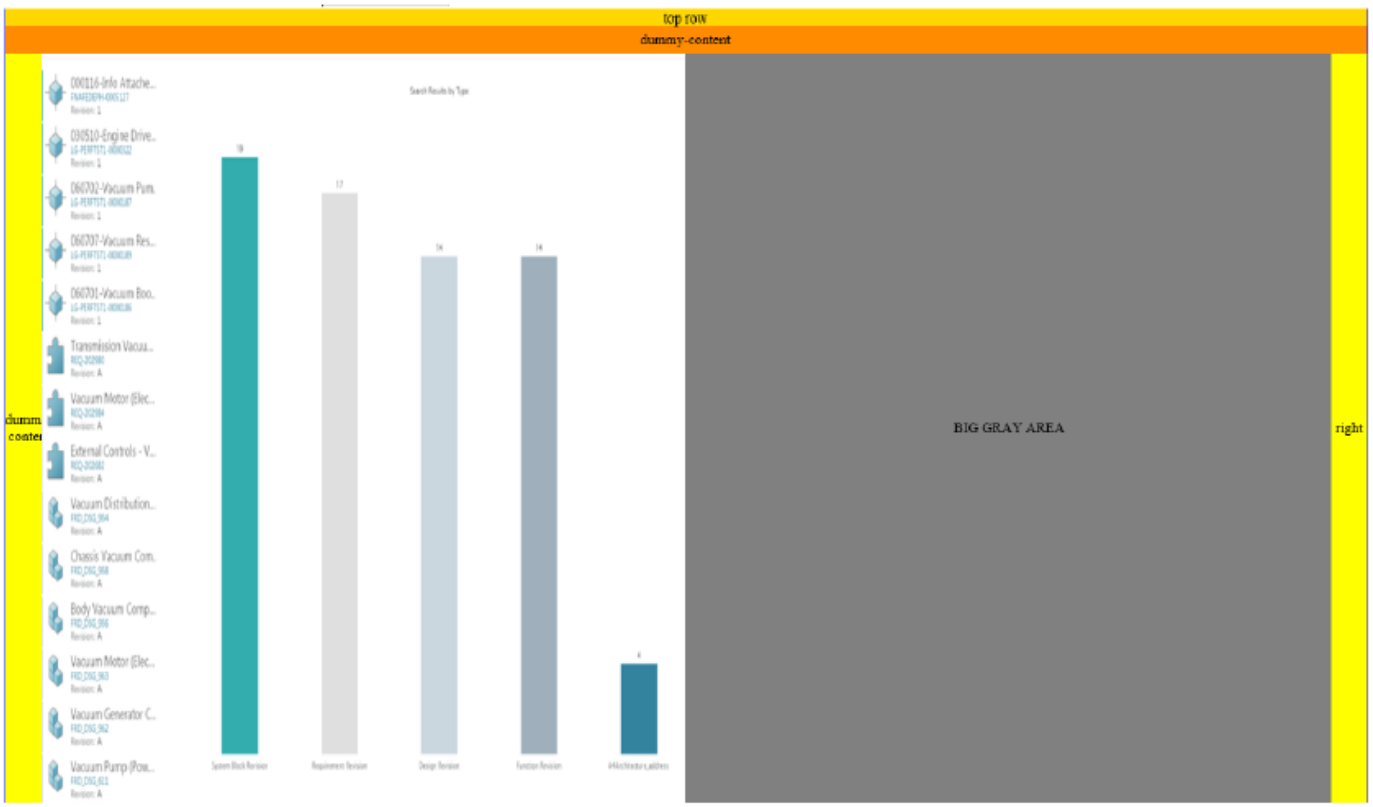

Siemens Web Framework
Unpublished work. © 2025 Siemens

```
</aw-row>
<aw-splitter></aw-splitter>
<aw-row height="3" color="pink">
    <aw-column width="3" color="lightblue"></aw-column>
    <aw-splitter></aw-splitter>
    <aw-column width="9" color="purple"></aw-column>
</aw-row>
<aw-splitter></aw-splitter>
<aw-row height="3" color="lightblue">
    <aw-column width="5f" color="blue"></aw-column>
    <aw-column width="fill" color="green"></aw-column>
    <aw-column width="10f" color="red"></aw-column>
</aw-row>
```

## Dummy Content

This example shows how may be used during layout development to show content within the container areas.

```
<aw-row height="2f" color="gold">
    <aw-dummy-content text"top row"></aw-dummy-content>
</aw-row>
<aw-row height="4f" color="darkorange">
    <aw-dummy-content></aw-dummy-content>
</aw-row>
<aw-row height="fill">
    <aw-column width="5f" color="yellow">
        <aw-dummy-content></aw-dummy-content>
    </aw-column>
    <aw-column width="fill">
        <aw-column width="6" color="green">
            <aw-dummy-content> image="results0.png"></aw-dummy-content>
        </aw-column>
        <aw-column width="6" color="gray">
            <aw-dummy-content> text="BIG GRAY AREA"></aw-dummy-content>
        </aw-column>
    </aw-column>
    <aw-column width="5f" color="yellow">
        <aw-dummy-content> text="right"></aw-dummy-content>
    </aw-column>
</aw-row>
```




## Multiple Data Sets

This examples show how the previous splitter example could have been defined in different data sets using the <aw-include\> element. The following code references three other sections of code which follow.

```
<aw-row height="6" color="blue"\>
    <aw-include name="toprow"></aw-include>
</aw-row>
<aw-splitter></aw-splitter>
<aw-row height="3" color="pink">
    <aw-include name="middlerow"></aw-include>
</aw-row>
<aw-splitter></aw-splitter>
<aw-row height="3" color="lightgreen">
    <aw-include name="bottomrow"></aw-include>
</aw-row>
```

`topRow`

```
<aw-column width="6">
    <aw-row height="2f" color="brown"></aw-row>
    <aw-row height="fill">
        <aw-row height="6">
            <aw-column width="4" color="orange"></aw-column>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            <aw-splitter></aw-splitter>
            <aw-column width="4" color="yellow"></aw-column>
            <aw-column width="4" color="olive"></aw-column>
        </aw-row>
        <aw-row height="6" color="green"></aw-row>
    </aw-row>
</aw-column>
<aw-splitter></aw-splitter>
<aw-column width="3" color="red"></aw-column>
<aw-column width="3" color="aqua"></aw-column>
```

`middleRow`

```
<aw-column width="3" color="lightblue"></aw-column>
<aw-splitter></aw-splitter>
<aw-column width="3" color="purple"></aw-column>
```

`bottomRow`

```
<aw-column width="5f" color="blue"></aw-column>
<aw-column width="fill" color="green"></aw-column>
<aw-column width="10f" color="red"></aw-column>
```

## Responsive Designs

The column and row based grid system used by Siemens Web Framework supports a way to re-arrange the grid based on the available width as a **responsive grid system** The page is always rendered as 12 column grid, irrespective of device size, howevre you can customize the size of your columns per device size. There are five different device sizes supported, namely extra small, small, medium, large, or extra large. Let's see how you can achieve the responsive layout.

## Responsive Layouts constructed with the `aw-row` and `aw-column`

- **Usecase** : Consider a page layout built with 6 columns, and you want to launch this page on different devices.
- **Challenge**: An extra large device has enough width to show all 6 columns properly, but imagine an extra small device trying to show all 6 columns. Each column would have very little width and it would be difficult to see contents.
- **Expectations**: You would rather prefer to see just one column on extra small device and would expect the other 5 columns to just wrap vertically with page sroll support, so you can still see the entire page contents.
- **Solution**: The aw-column supports when attribute to control the width of each column per device size


Unpublished work. © 2025 Siemens

Let's look look at below code snippet

```
<aw-column width="6" height="10f" color="red" when="xlarge: 2, large: 3,
medium: 4, small:6, xsmall:12"> </aw-column>
```

where,

- The default width for column is specified as 6 in proportionate unit, that means it would divide page into 2 columns. Remember we use 12 column grid, so when we divide total width in 6 equal parts, we can have just 2 columns.
- Now lets' understand what each of keys inside `when` attribute denotes
  - xlarge : extra large device (min-width: 1200px)
  - large : large device (min-width: 992px)
  - medium : medium device (min-width: 768px)
  - small : small device (min-width: 576px) and (max-width: 767px)
  - xsmall : extra small device (max-width: 575px)
- Now let's understand what each key value inside `when` attribute denotes
  - xlarge: 2 -> It means width for this column would be 2 on extra large device. Alternatively it means page can show 6 such columns on extra large device
  - large: 3 -> It means width for this column would be 3 on large device. Alternatively it means page can show 4 such columns on large device
  - medium: 4 -> It means width for this column would be 4 on medium device. Alternatively it means page can show 3 such columns on medium device
  - small: 6 -> It means width for this column would be 6 on small device. Alternatively it means page can show just 2 such columns on small device
  - xsmall: 12 -> It means width for this column would be 12 on extra small device. Alternatively it means page can show just single such column on extra small device

## Example

Let's take one more example to understand it better

```
<aw-row height="20f">
            <aw-column width="6" height="10f" color="red" when="xlarge: 2,
large: 3, medium: 4, small:6, xsmall:12">xlarge: 2, large: 3, medium: 4,
small:6, xsmall:12 </aw-column>
            <aw-column width="6" height="10f" color="blue" when="xlarge: 2,
large: 3, medium: 4, small: 6, xsmall:12">xlarge: 2, large: 3, medium: 4,
small:6, xsmall:12 </aw-column>
            <aw-column width="6" height="10f" color="green" when="xlarge:
2, large: 3, medium: 4, small: 6, xsmall:12"> xlarge: 2, large: 3, medium:
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
4, small:6, xsmall:12</aw-column>
            <aw-column width="6" height="10f" color="yellow" when="xlarge:
2, large: 3, medium: 4, small: 6, xsmall:12">xlarge: 2, large: 3, medium:
4, small:6, xsmall:12 </aw-column>
            <aw-column width="6" height="10f" color="pink" when="xlarge: 2,
large: 3, medium: 4, small: 6, xsmall:12">xlarge: 2, large: 3, medium: 4,
small:6, xsmall:12 </aw-column>
            <aw-column width="6" height="10f" color="grey" when="xlarge: 2,
large: 3, medium: 4, small: 6, xsmall:12">xlarge: 2, large: 3, medium: 4,
small:6, xsmall:12 </aw-column>
</aw-row>
```


Unpublished work. © 2025 Siemens

## Example in Storybook

Please refer to `aw-column` example in **Siemens Web Framework Storybook** and see *Column with responsive grid layout*.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Using custom stylesheets

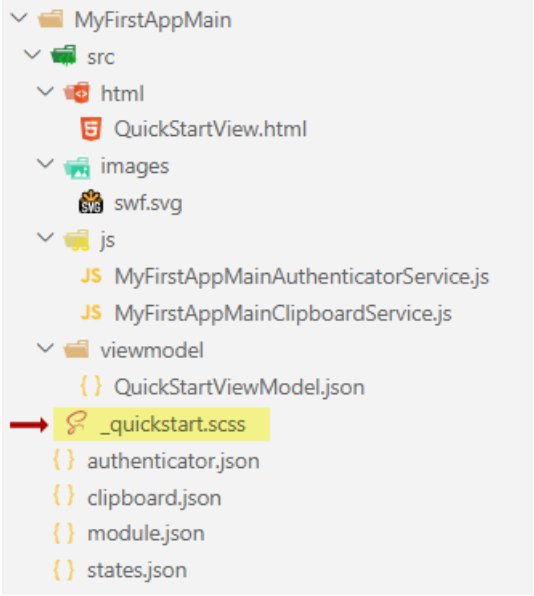
Siemens Web Framework comes with a rich collection of graphical controls, and out-of-the-box styling and theming to match the latest Siemens user experience guidelines. Therefore, typically you should not have to specify your own custom styles. However, there are certain situations in which this may be required.

## Adding a custom stylesheet to a module

When a Siemens Web Framework application is built, all the styles defined in all the modules included in the application are merged into a single CSS file. Styles are defined using **Sass**, a popular CSS extension language, which supports advanced features not available in CSS.

You can add your own styles as a `.scss` file to any Siemens Web Framework **module**, as follows:

1. Navigate to the `src` folder within your module.
2. Create a `.scss` file directly under the `src` folder, starting with an underscore (_).




3. Add your style rules inside this file.

When you build your application, the contents of your `.scss` file will be merged with the rest of the styles provided by Siemens Web Framework and other modules.

**Tip**

You can add more than one `.scss` file per module and they will be included in the build, but they *must* start with an underscore.

# Command Sub Panels

Similar to `aw-include`, the `aw-command-sub-panel` UI Element is used to load a component by specifying its panel id. It is also possible to configure navigations through panels similar to wizard steps.

Consider the following example:

```
<aw-command-panel caption="i18n.effectivity">
    <aw-command-sub-panel
       panel-id="ApsEffectivitySub"
       visible-when="data.activeView==='ApsEffectivitySub'"
       sub-panel-context="data.effectivity">
    </aw-command-sub-panel>
</aw-command-panel>
```

The `aw-command-sub-panel` component supports the following props:

- `panel-id`: The name of the component to display inside the sub panel (`ApsEffectivitySub` in this case).
- `visible-when`: Condition determining the visibility of the loaded component.
- `sub-panel-context`: Object used to pass the parent component state to the component specified through `panel-id`. The data passed through `sub-panel-context` would be available as props inside the loaded component.

## Implementing panel navigation with named events

**Note**

Using named events is generally not recommended as it limits reusage of a component in an application. You can use this method if you do not need to re-use your `aw-command-sub-panel` component or follow the example in the following section for an alternative which allows component reuse.

Consider the following example:

```
<aw-command-panel caption='i18n.commandPanelTitle'>
    <aw-command-sub-panel visible-when="data.activeView ===
'CommandSubPanel1'" panel-id='panel1'
        sub-panel-context='{data, dispatch}'>
    </aw-command-sub-panel>
</aw-command-panel>
```

And assume that the view of `panel1` is the following:

```
<aw-panel-body>
    <aw-panel-section caption='Properties'>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        <aw-panel-body>
            <aw-widget {...fields.id}></aw-widget>
        </aw-panel-body>
    </aw-panel-section>
    <aw-panel-section caption='Projects'>
        <aw-command-bar anchor='addProjectAnchor'
alignment='HORIZONTAL'><aw-command-bar>
    </aw-panel-section>
</aw-panel-body>
```

Let's suppose you want to switch to another panel called `panel2` from `panel1` when clicking a command contributed in the `addProjectAnchor` command bar.

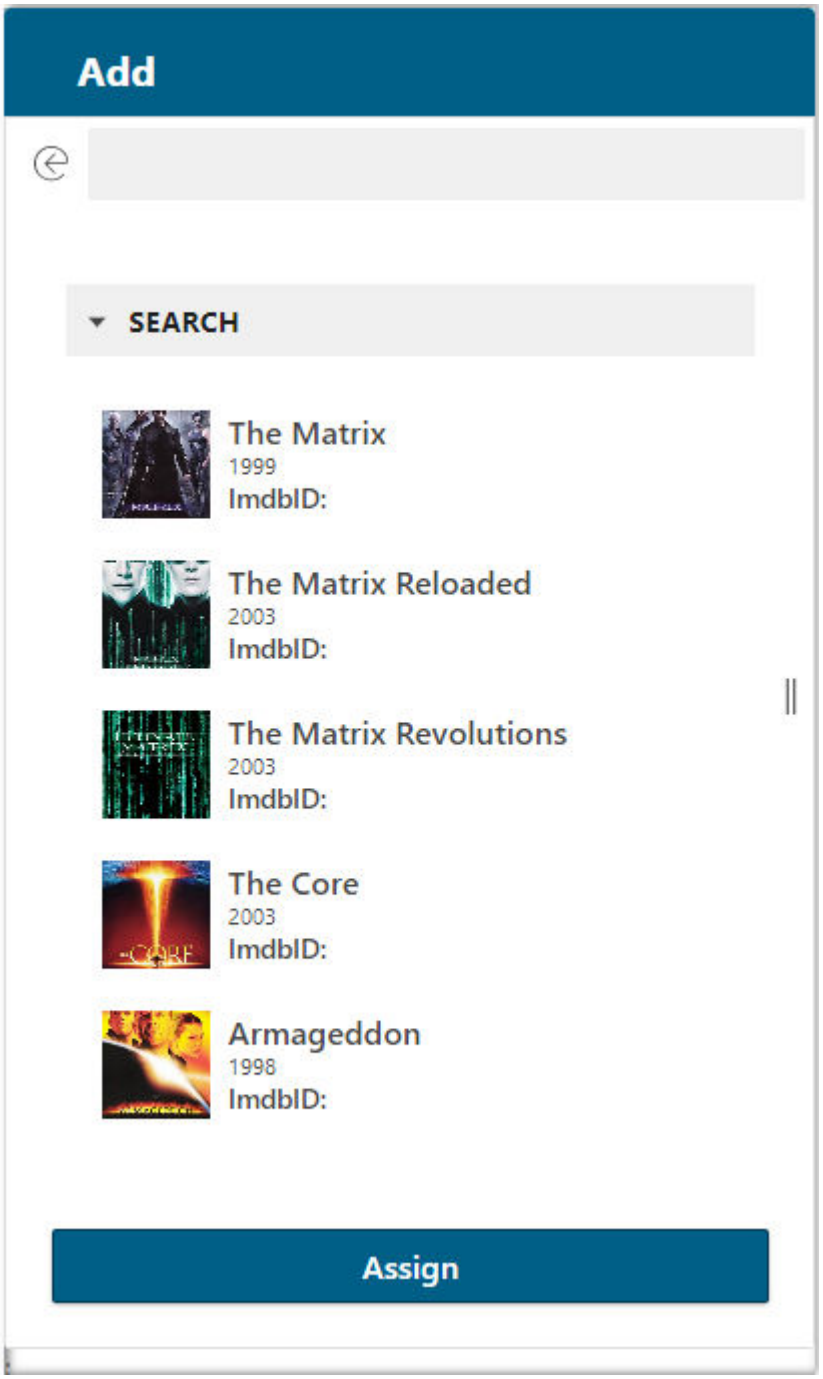
To achieve this, you have to fire the `awPanel.navigate` event as follows:

```
{
    "name": "awPanel.navigate",
    "eventData": {
        "destPanelId": "panel2",
        "supportGoBack": true,
        "title":"add"
    },
    "excludeLocalDataCtx": true
}
```

This event is listened by the parent `aw-command-panel` component, which will automatically hide `panel1` and load `panel2` in its place.

Additionally, a back button will automatically appear in the header of `panel2` and user can use is button to navigate back to `panel1`:


Unpublished work. © 2025 Siemens




If for some reason you need to navigate back to the parent panel by clicking on another command instead of using the back button, you need to fire the `awPanel.navigate` event again with the following data:

```
{
    "name": "awPanel.navigate",
    "eventData": {
        "destPanelId": "panel1",
        "backNavigation": true
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    },
    "excludeLocalDataCtx": true
}
```

## Implementing panel navigation without named events

Consider the following example:

```
<aw-command-panel caption='i18n.commandPanelTitle'>
    <aw-command-sub-panel
        show-when="data.sharedData.activeView === 'CommandSubPanel1'"
        panel-id='CommandSubPanel1'
        sub-panel-context='{sharedData:data.sharedData}'>
    </aw-command-sub-panel>
    <aw-navigate-panel
        title='back'
        show-when="data.sharedData.activeView === 'CommandSubPanel2'"
        panel-id='CommandSubPanel2'
        sub-panel-context='{sharedData:data.sharedData}'
        action='backAction'>
    </aw-navigate-panel>
</aw-command-panel>
```

In this case, the `aw-navigate-panel` component is to encapsulate the navigation behavior while allowing data to be shared across sibling sub panels, without using named events. This can be accomplished by defining the data to share as **atomic data** in the parent ViewModel, like this:

```
{
    "data": {
        "sharedData": {
            "initialValues": {
                "activeView": "CommandSubPanel1"
            },
            "meta": {}
        }
    }
}
```

Whenever you need to perform a navigation, you need to set the active view in the shared data using **inverse data flow**. This can be done from any JSFunction action calling a method like the following:

```
export const goToCommandSubPanel1 = ( dataProvider, sharedData ) => {
    let sharedDataVal = sharedData.value;
    sharedDataVal.selectedObjects = dataProvider.selectedObjects;
    sharedDataVal.activeView = 'CommandSubPanel1';
    sharedData.update( { ...sharedDataVal } );
};
```

In this case, the method updates `sharedData` by calling its `update` method.

# Table

The **aw-splm-table** component provides a high-performance grid control that provides:

- consistent user experience
- advanced features to manage data, like filtering, sorting, hierarchical and matrix view
- advanced single and bulk editing capabilities

The **Working with Tables** tutorial explains how to configure a table component step-by-step. Additionally, a lot of examples covering all the functionality provided by this component are provided in Storybook.

For the reference documentation concerning aw-splm-table, see the following pages:

- **aw-splm-table** the component reference documentation.
- **GridOptions** the object used to configure the main settings of a table.
- **DataProviderConfiguration** the object used to configure a data provider, that is used to populate a table with data.
- **ColumnProviderConfiguration** the object used configure a column provider, that is used to define which columns will be displayed in a table.

## FAQ

**Q. How can table cells be put into edit mode?**

**A.** SPLM table will go into edit mode if the editHandlerStateChange or plTable.editStateChange event is published with eventData that has a state of 'starting' and dataSource matching/containing the tables dataProvider. For details on how to use this event, review the details in the 'Supported Events for PL Table' section. You can also look at the "Editable table with various property types" example in Storybook for aw-splm-table.

**Q. I have modified the properties of a view model object on the client and the table is not reflecting these changes. How can I get the table to reflect these changes?**

**A.** SPLM Table has a watcher on the loadedVMObjects array in the viewModelCollection of the DataProvider. This watcher will not be fired if a property within one of the loaded view model objects is modified since the object's reference in the loadedVMObjects array remains unchanged. In this situation, the event 'gridid' + '.plTable.clientRefresh' can be published to make SPLM Table re-render all cells in the table with the latest data for each loaded view model object. You can find more details on the usage of the "clientRefresh" event in the 'Supported Events for PL Table' section.

**Q. How can I have SPLM Table render cells with my own custom implementation?**


Siemens Web Framework
Unpublished work. © 2025 Siemens

**A.** An example of how to implement your own cell in SPLM table can be found in Storybook. For aw-splm-table, refer to the "Custom cell template" example.

**Q. The height of my table is too small. How can I increase the maximum height of the table?**

**A.** By using the maxRowsToShow gridOption, you can set the number of rows you would like the table to display before showing a vertical scrollbar.

**Q. The column headers are blank on my table. How can I get them to show up?**

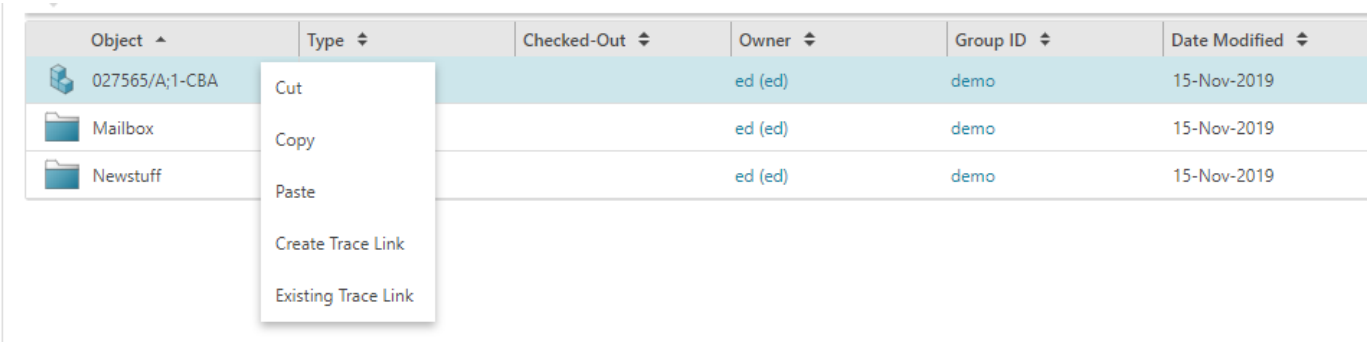
**A.** The text displayed in the column header is set via displayName property on each column in the columnConfig. Update the column in the columnConfig so that it has this property set to the desired value.

**Q. Where can I find details about the supported options of SPLM Table?**

**A.** Please see the 'Supported Options for PL Table' section. If you are converting from aw-table you can also refer to the 'Unsupported Options for PL Table when converting from AW Table' to find rationale and alternative options to use with SPLM table.

**Q. What is context menu and how can I enable it on Table?**

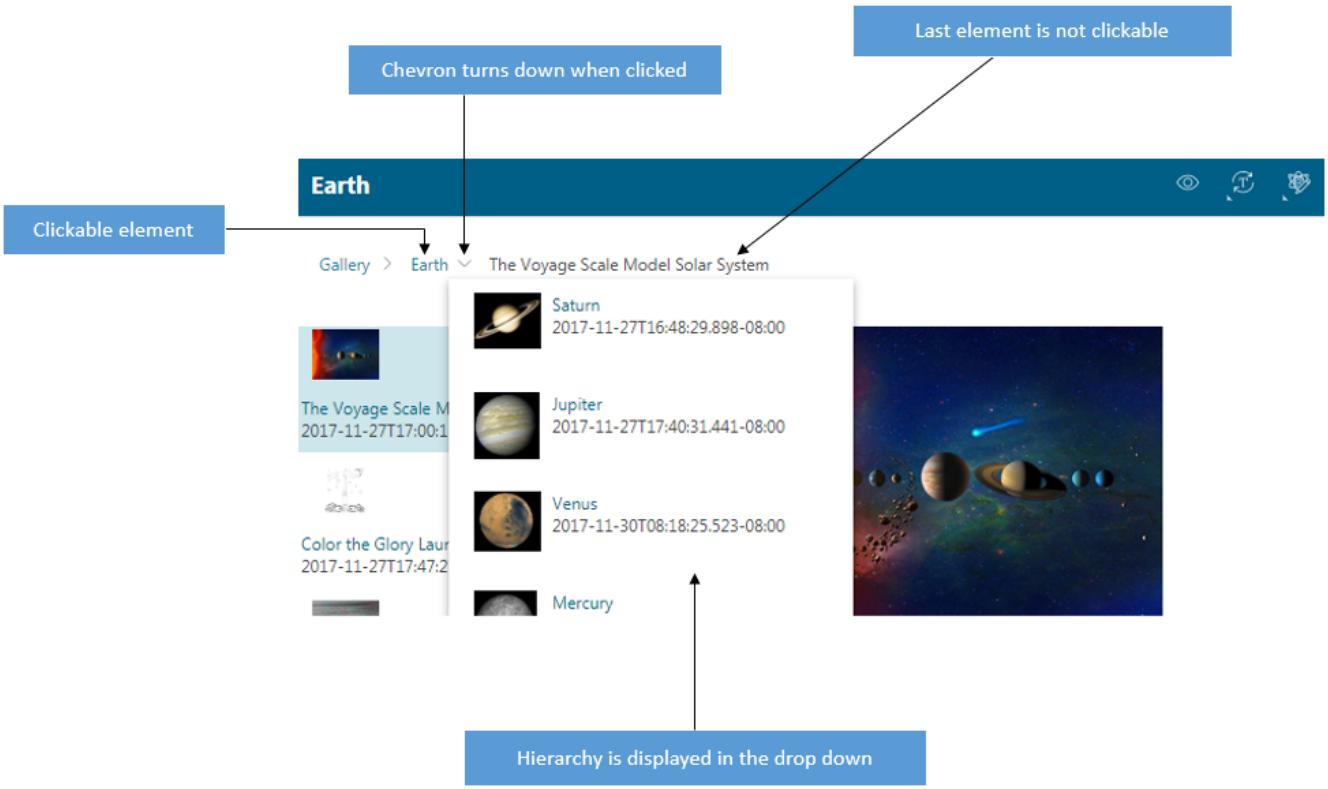
**A.** Context menu is a popup, that opens on right click, showing a list of commands. This is only supported on configuration of aw-splm-table. Please visit **How to enable context menu** to configure.




# Breadcrumb

`aw-navigate-breadcrumb` is a UI Element that is used to show users where they are within an application or content. It is represented as a list of Links that can be used to navigate within the scope of the breadcrumb separated by chevrons that can also be clicked to provide additional naviation.

Breadcrumbs tell you where you are in a BOM, Assembly or folder structure. This type of breadcrumb displays the hierarchy to allow a user to traverse or change direction as to what they are trying to accomplish.




## Usage

The `aw-navigate-breadcrumb` component can be used as follows:

```
<aw-navigate-breadcrumb crumbs="data.crumbs"
popuplist="AwDefaultBreadCrumbList">
</aw-navigate-breadcrumb>
```

- The `crumbs` prop must be set to a list of crumb configuration objects, provided as an array that can be constructed with any ordinary action.
- The `popuplist` prop must be set to the name of a component which will be loaded in a popup when the chevron is clicked. The following props will be passed to the component specified via the `popuplist` prop:
  - `selectedCrumb`, the currently-selected crumb object.
  - `chevronPopup`, a reference to the popup containing the component displayed.

Each crumb configuration object included in the array specified via the `crumbs` property is an object similar to the following:

```
{
    "clicked": false,
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "displayName": "Home",
    "showArrow": true,
    "primaryCrumb": true,
    "objectsCountDisplay": " (0 Objects)",
    "onCrumbClick": (crumb, event) => { /* ... */ }
}
```

Where:

- `displayName` : The name of the crumb.
- `objectsCountDisplay`: Optional text to display next to the name of the crumb.
- `primaryCrumb`: Whether the crumb is primary or not. Generally, the first crumb is the primary crumb and a different set of css is applied.
- `scopedUid`: The uid of the object the crumb is representing.
- `showArrow`: If this attribute is true, the chevron is shown at the right side of the crumb.
- `onCrumbClick`: An inline function which will be invoked when a crumb is clicked. The selected crumb and the onclick event would be passed as parameter.

# Data Flow Patterns

When it comes to data flow, there are three fundamental patterns that should be used in your application:

- The **uni-directional flow** of data When data must be shared across various components, you need to *lift the state up* to the nearest common parent. From there, data gets passed down as read-only to all child components. There should be no (or negligible) dependency on anything global or singletons to facilitate the reuse of components.
- The **inverse flow** of data In the uni-directional flow of data, typically, the data is read-only. But if the use case requires the ability to update this data from the child components, this indicates the need for an inverse data flow to be applied. The inverse data flow pattern ensures that the **atomic data** is created and passed down so that child components can update it. Additionally, any component within the same component hierarchy can observe changes and react accordingly.
- The **complex inverse flow** of data The complex inverse data flow is similar to the inverse data flow, but it deals with passing complex data around complex component hierarchies.

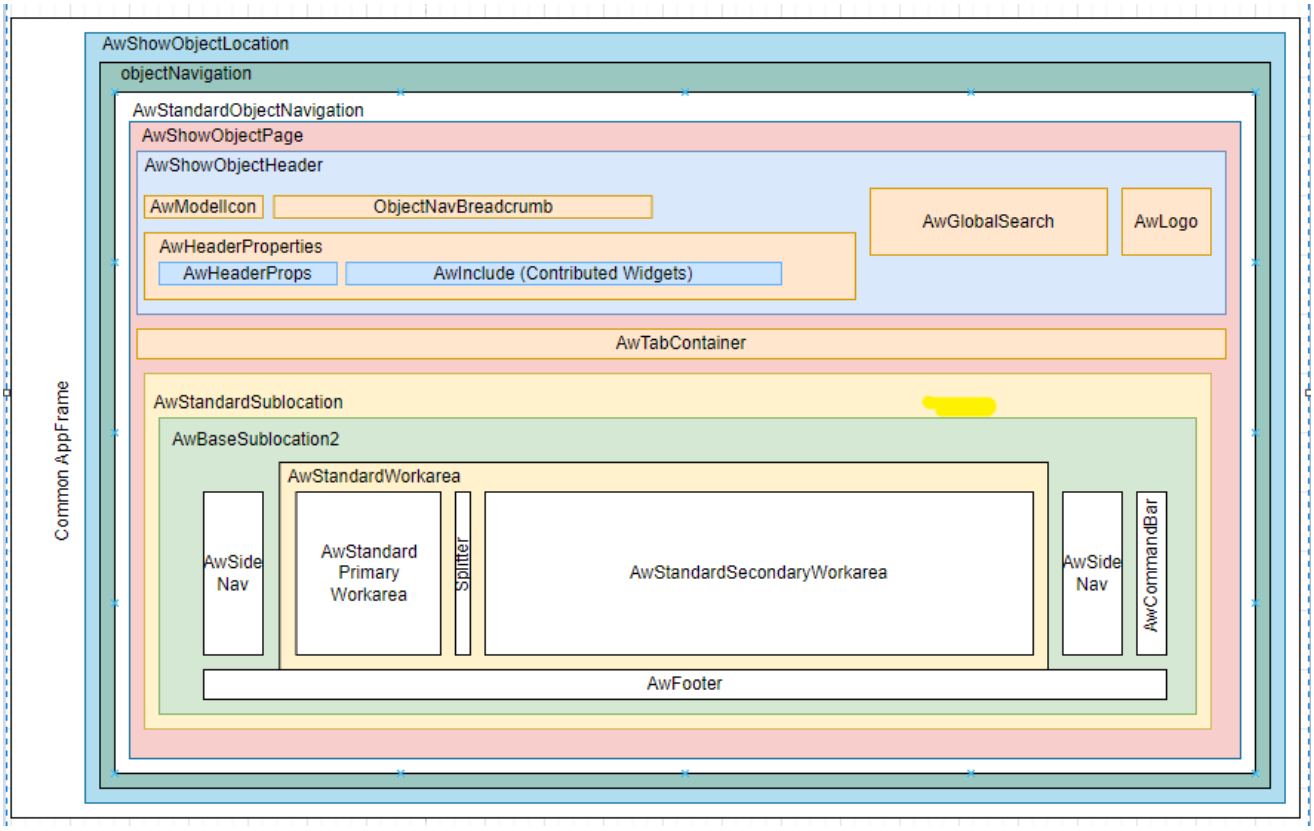
## How to choose the right data flow pattern

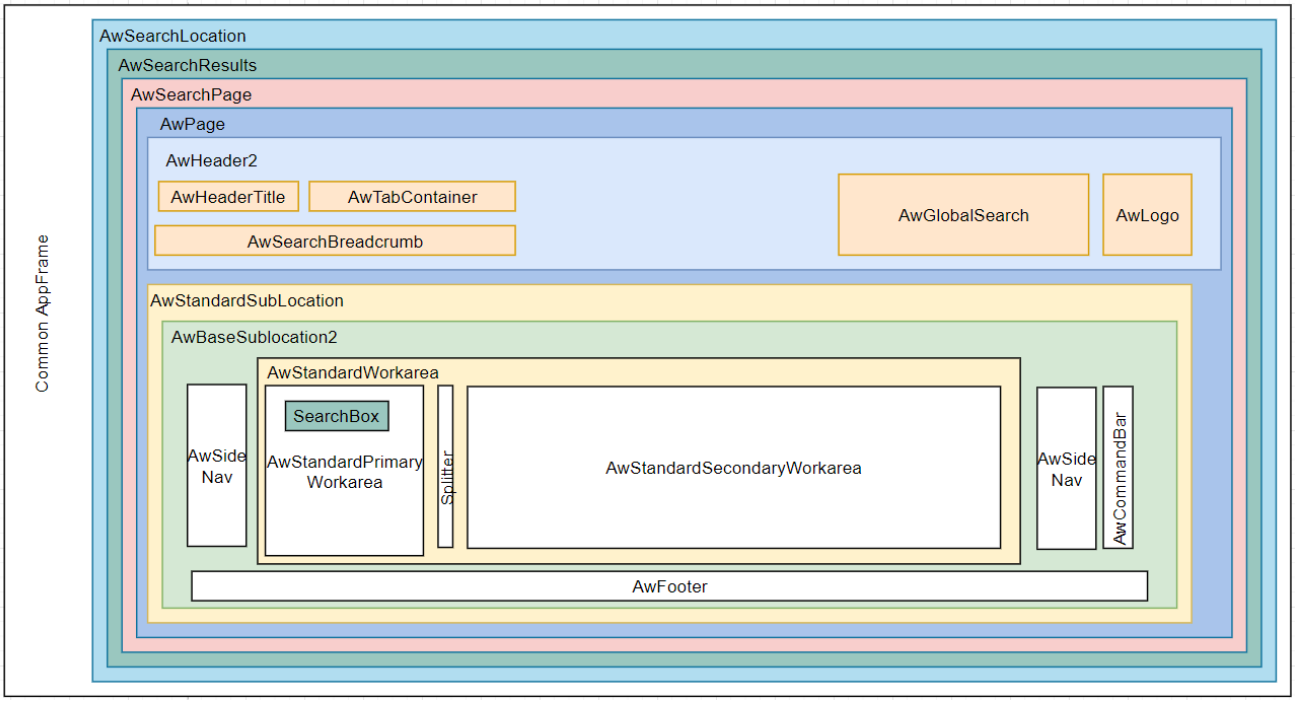
There are three steps to decide which is the best data flow pattern for a particular use case:

1. Understand the component hierarchy An essential step is to know the component hierarchy, i.e. what different components would be interacting and where they belong in the overall component


Unpublished work. © 2025 Siemens

hierarchy of your application. The easiest way to do this is to represent the component hierarchy in a diagram like the following:





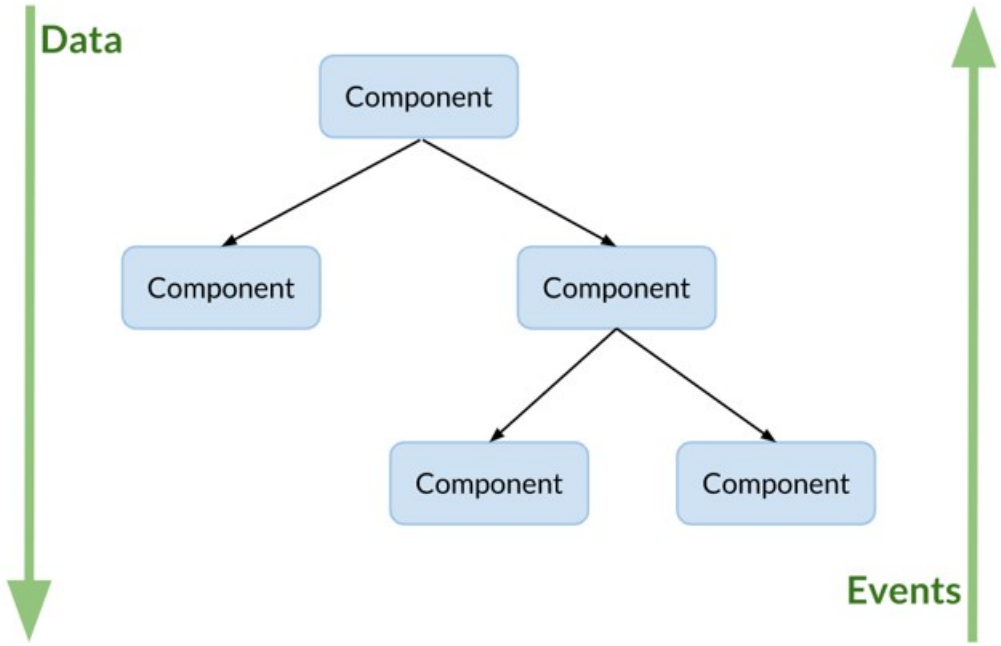
Siemens Web Framework
Unpublished work. © 2025 Siemens

2. Understand which properties (props) are passed to components Once you know the hierarchy of the components, the next step consists in determining the props that are being passed among these components. Of the properties that are being passed, understand:
   - What needs to be strictly read-only
   - Which would need to support for updates and by which components
3. Understand what state is being shared among various components In this step, you need to determine the component-level state across the various components identified in your hierarchy. The basic rule of thumb is that the data needs to be hoisted to the least common parent of the components which need to share the data.

## Uni-directional Data Flow

When following this pattern, data flow in one and only one way. More specifically:

- Data flows from parent to child in the form of props.
- Actions are triggered by the props passed to components.
- Actions can update the state of the component through a controlled API only.
- The rendered visual representation of a component is solely based on the internal logic dependent on the currently-passed properties and component-level state.




Note that:

- Data cannot flow the other way, i.e. no *two-way data binding* is possible.


Unpublished work. © 2025 Siemens

- State is always owned by one component. Any data that is affected by this state can only affect the components below it, i.e. its children.
- Changing the state of a component never affects its parent, its siblings, or any other component in the application; just its child components.
- To share state among different components, you must *lift it up* to the common containing parent component.

### Advantages

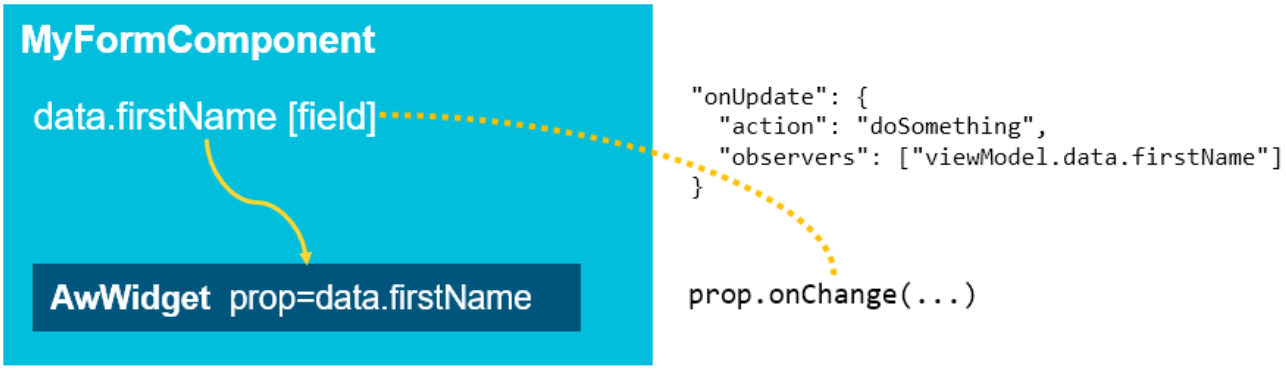
This data flow pattern has the following main advantages:

- It is less error-prone, as you have more control over your data
- It is easier to debug, as you know what data is coming and from where
- It is more efficient, as the library already knows what the boundaries are of each part of the system

## Inverse Data Flow

The easiest way to explain inverse data flow is to think of a simple form component. This component (MyFormComponent in the following diagram) holds the form state, e.g. a `firstName` view model property.

The `firstName` view model property is defined in the `data` section of MyFormComponent and it is automatically converted to a **field** object and passed down to the AwWidget component:




When the property is changed (for example because the user types in a textbox rendered by AwWidget), AwWidget invokes the `onChange()` callback which was defined at field creation (which occurred *within the parent component*). In this case, even though the value of `firstName` is changed from the child AwWidget component, the data is actually mutated in its parent, where the `onChange()` method was defined.


Siemens Web Framework
Unpublished work. © 2025 Siemens

The parent component can react to changes triggered by child components by observing the property and triggering an action via the `onUpdate` hook.

**Note**

The example discussed above shows how inverse data flow is used within internal framework components (e.g. AwWidget). For an example of inverse data flow that is entirely managed at application level, see **this atomic data example**.

## Complex Inverse Data Flow

With **unidirectional data flow**, data only flows in one way, from a parent component to its children, via props. Additionally, the component **state** is only owned by a single component and can only be updated by the component alone.

This has two key implications:

- If the child component must modify the data passed from a parent component, it must do through a callback function also passed from the parent component that allows update. This is referred to as **inverse data flow**
- Any data or state that must be shared between sibling or distant cousin components, must be lifted up to the lowest common parent/ancestor of the components sharing the data.

Complex Inverse data flow takes the basic inverse data flow situation and builds upon it by dealing with the following:

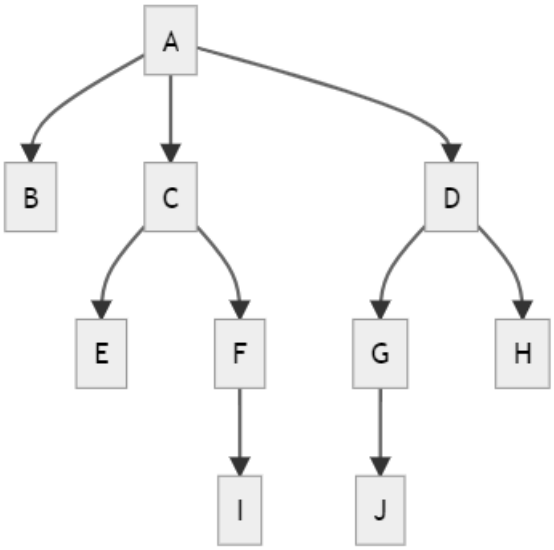
- Complex state The state is more complex than just a simple view model property, but rather an object that references one or more levels of other objects. In other words, the object **references** are nested to form a hierarchy of data as a network or graph of objects. These individual objects may contain scalar properties or other references to objects and so on. So for any given data state hierarchy, there could be any number of properties at any level of the hierarchy that may be required to be passed to components as **fields**:

```
const a = {
   b: [
      { x: "foo", y: true, z: 42},
      { x: "bar", y: false, z: 17},
   ],
   c: "someString",
   d: { m: 14, n: [ "ready", "steady", "go"]}
}
```

- Complex component hierarchies The child component hierarchies are more complex. As you decompose the page to smaller and smaller components following good **SOLID** principles, you may find your component hierarchy getting wider and deeper. As those component hierarchies get wider and deeper and you must share data between more components, you will discover that you have to hoist more and


Unpublished work. © 2025 Siemens

more data to common parent components. You are not just passing the state just one child component, but rather to multiple child components. Also, those child components may be acting as a pass-thru and




are further passing the data to their children and so on:

- Property decomposition Another complexity is selectively passing down only a subset of the original passed-in data to child components. For example, assume you passed down data a from the example above to component `C`. Inside `C` render function it is looping over the `a.b` array and for each object in the array it is passing down only the object in the array to the child components `E` and `F`. In this way, you are decomposing the data as you traverse down the component hierarchy. This is especially true if the small parts of the data needs to be displayed and modified by property widgets through fields.
- Updates to any part of the data affect the whole This means that the original state owned by the root component is **immutable**. Any modification to any part of the nested data structure will result in a completely new object with the modification. This way if even if a small part of the nested object is modified anything dependent on any part of the object will get re-rendered.

## Data Communication

There are 5 declarative ways by which we can share data between different viewmodels.

- **application context**
- **Events**
- **subPanelContext**
- **Props**
- **Atomic Data**

Understand the usage of above approaches in practical world.

**WHEN TO USE WHAT**(refer **subPanelContext** for more details)


Unpublished work. © 2025 Siemens

| View | Usage | Ctx | subPanelContext | Best Practice |
|---|---|---|---|---|
| with aw-include | Communication between two Viewmodels which are in parent child relation | Do not Use Me | Use Me | subPanelContext + Events(if required) |
| with aw-include | Communication between two Viewmodels which are not in parent child relation | Use Me | NA | Ctx + Events(if required) |
| without aw-include | Communication between two Viewmodels | Use Me | NA | Ctx + Events(if required) |

## subPanelContext

`subPanelContext` is used to share context specific information between ViewModels which are connected using `aw-include` or it is often specified as a prop for certain **components**. It can be used declaratively to access shared context information inside child ViewModel (specifically inside data section, actions, conditions etc.) as explained in the following sections.

### Usage of subPanelContext in ViewModels

Consider the following usage of the `aw-include` UI element:

```
<aw-include name="teamForm" sub-panel-context="data"></aw-include>
```

...where `data` contains the following:

```
{
    "data": {
        "firstName": {
            "displayName": "First Name",
            "type": "STRING",
            "dbValue": "MS Dhoni"
        }
    }
}
```

In the included ViewModel (`teamFormViewModel.json` in this case), it is possible to access `subPanelContext` in expressions in the `data` section, as follows:

```
{
    "data": {
        "firstName": {
            "displayName": "First Name",
            "dbValue": "{{subPanelContext.dbValue}}"
```

```
        }
    }
}
```

Also, it is possible to access `subPanelContext` in expressions used in actions and conditions, like this:

```
{
    "actions": {
        "getSubPanelContext": {
            "actionType": "JSFunction",
            "method": "buttonAction",
            "inputData": {
                "data": "{{subPanelContext}}"
            },
            "deps": "js/showcaseButtonActions"
        }
    },
    "conditions": {
        "showLabelText1": {
            "expression": "subPanelContext.title.isEditable === true"
        }
    }
}
```

### Usage of subPanelContext in View

Similarly, considering the following usage of `aw-include`:

```
<aw-include name="teamForm" sub-panel-context="{title:'Team Details'}"></aw-
include>
```

It is also possible to access `subPanelContext` in the child View (`teamFormView.html` in this case) like this:

```
<aw-panel-section caption="{{subPanelContext.title}}"></aw-panel-section>
```

### FAQ

`Q.` when should I use `subPanelContext`?

`A.` `subPanelContext` used to communicate between two ViewModels in parent child relationship. Parent viewmodel can use the `subPanelContext` to pass on same data/property to all its child viewmodels.

`Q.` How the update for data inside the `subPanelContext`?


Siemens Web Framework
Unpublished work. © 2025 Siemens

A. `subPanelContext` in child level should contain the reference of some `data.xxx` model about it's parentNodes. So any changes about these referenced model should be updated in `subPanelContext`.

Q. What are some best practices to use `subPanelContext`?

A. Never overwrite the `subPanelContext` reference with a new object. This would break the chain between parent and child sharing the same subPanelContext.

`Error Case`: if we create new `subPanelContext` object at child level, then any change for model inside `subPanelContext` from parent cannot pass to child. Best practices: try to re-use the same `subPanelContext` if available otherwise create a new one.

## Atomic data

There are different **data flow patterns** that can be implemented among the components interacting with each other in your application.

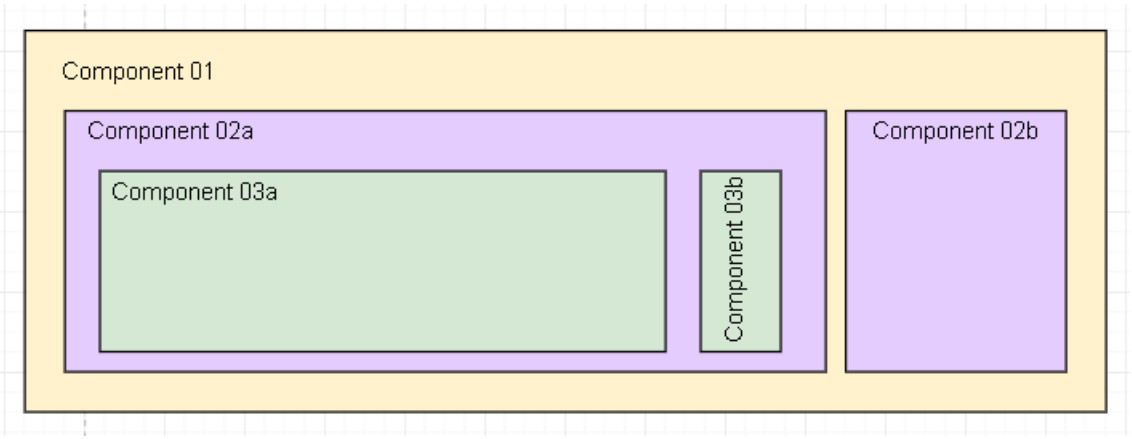
As a general rule, whenever sibling child components need to share data, the approach is to *lift the state up* and store it within the closest common parent. This facilitates the sharing of data among the sibling components and can be extended for complex interactions among components.

The term atomic data identifies a data structure and an approach to help deal with these **inverse flow** requirements.

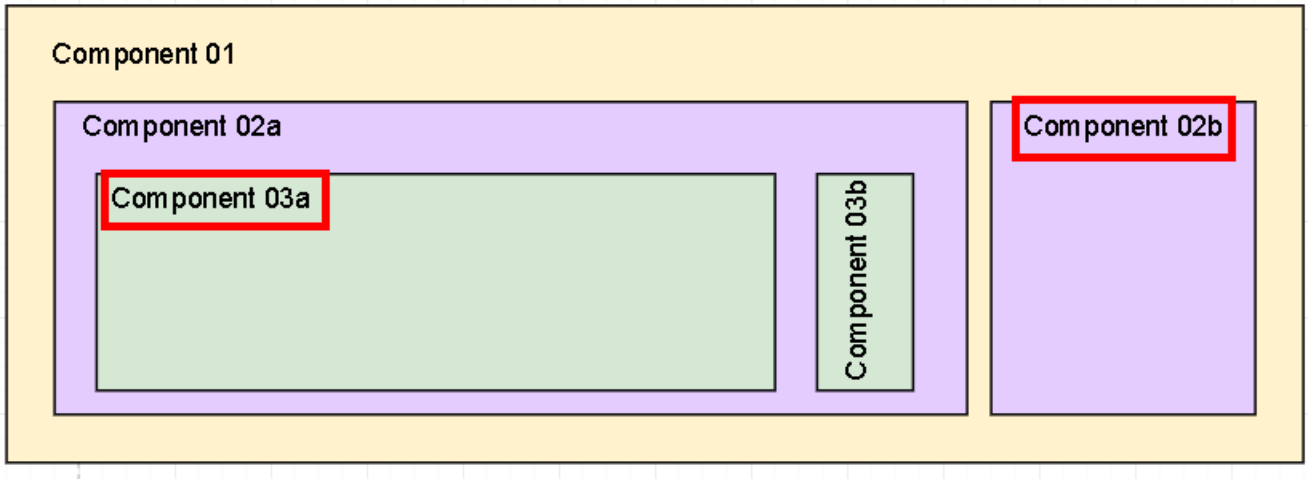
Atomic data allows you to define a free-flowing data structure in the parent component which can then be shared with its child components, and it provides a safe API that child components can use to update this data.

### How atomic data works

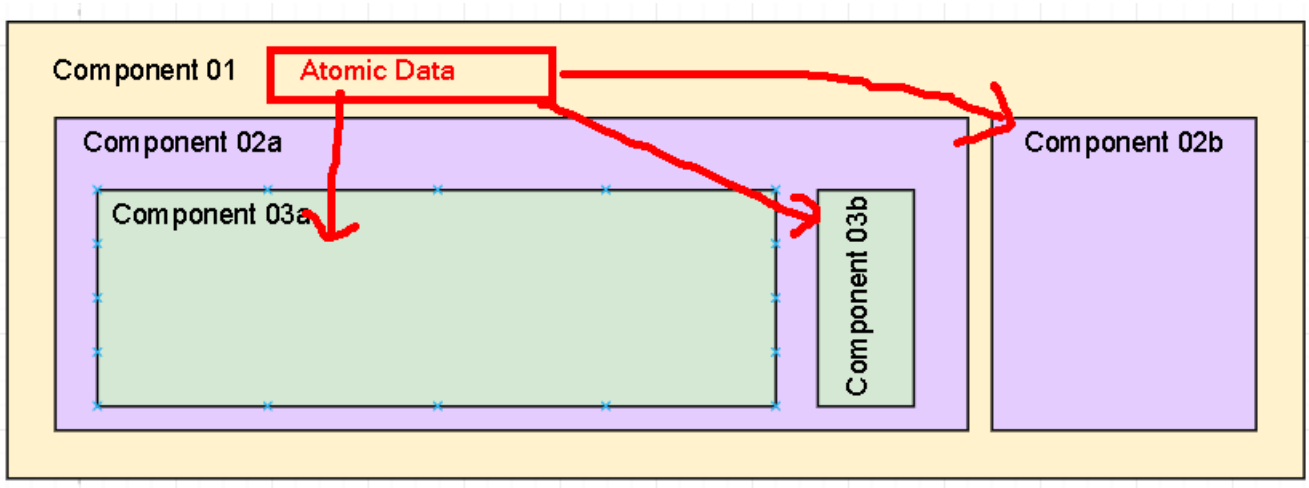
Consider the following component hierarchy:



In this case, if you need to share data between any two components, for example, `Component 03a` and `Component 02b`, you have to lift this common data up to the closest common parent (`Component 01` in this case).



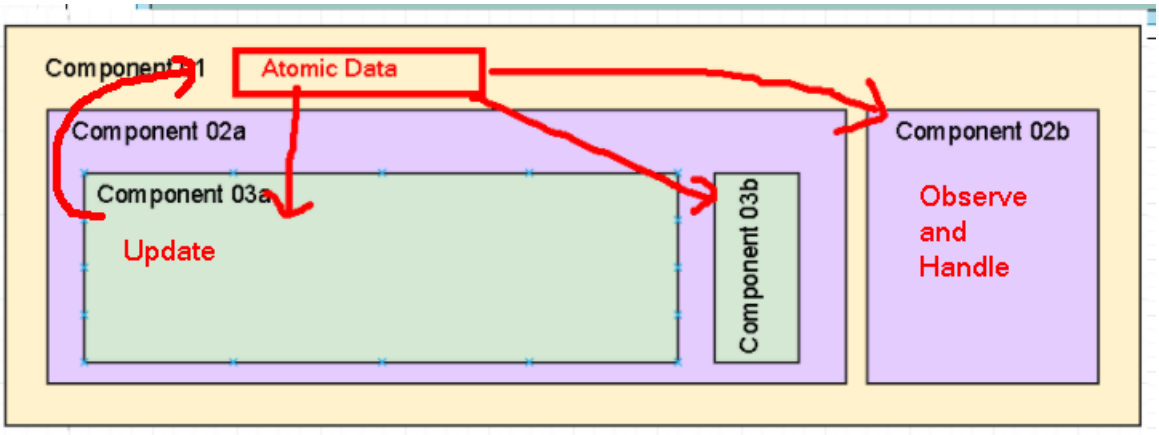

Within this common parent, you can define an atomic data object and share it down through the component hierarchy via props:



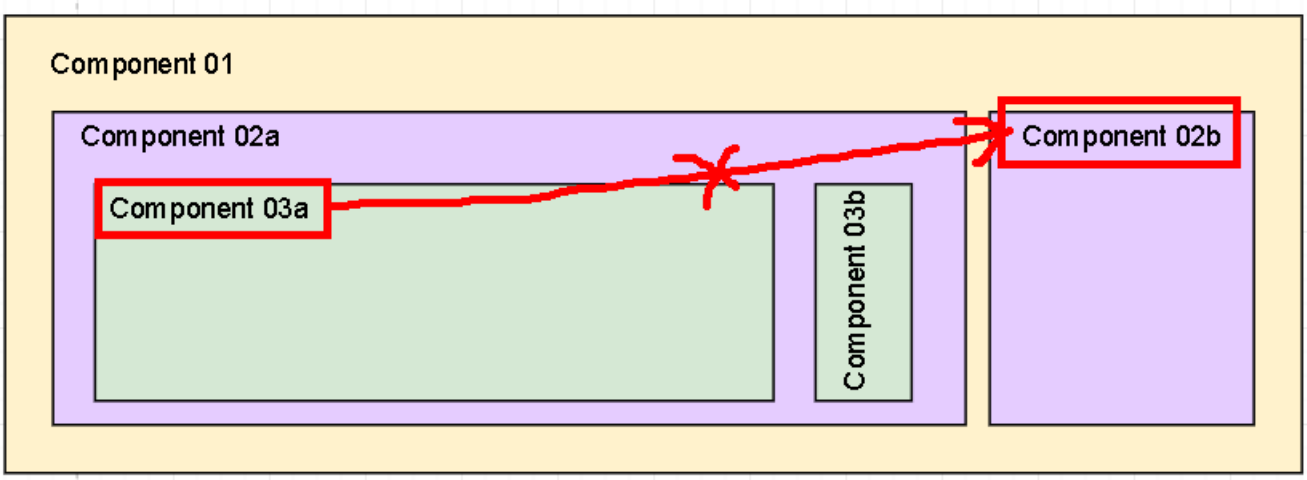

In this case:

- the component that needs to update this shared data (for example `Component 03a`) can safely update it using atomic data update API
- the component that needs to react to these modifications (for example `Component 02b`) can observe this data and get notified whenever it changes.


Siemens Web Framework
Unpublished work. © 2025 Siemens




The key point here is that there is no direct interaction happening between two components across the component hierarchy: it always happens through the closest common parent.




## Structure of atomic data objects

Atomic data is a free-flowing object that can be defined in the `data` section of the view model of a component. This object can be as complex as needed and it can include other objects.

By default only one **field** for the entire object is created, but if you want you can define which properties within your nested data structure must be converted into fields by specifying a `meta` property within the atomic data object definition.

Overall, there are two pieces of atomic data definition:

- `initialValues` Here you can specify the initial value for your complex object (even an empty object).
- `meta` Here you can specify which properties in your nested data structure must be converted into fields.

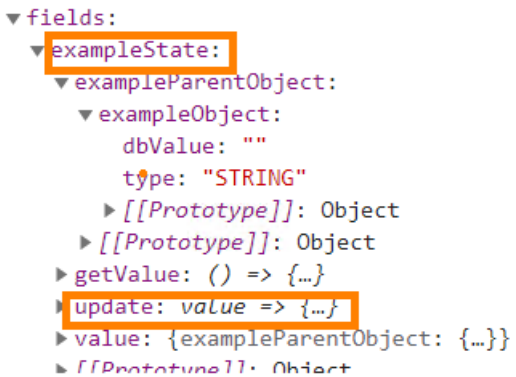
**Note**

Atomic data is called *atomic* because the entire nested data structure is treated as a single immutable object. An update to any property in the structure will result in an entirely new object being created (deep-cloned) with the new modified value substituted in the new object.

Consider the following example of an atomic data object with an empty data section:

```
{
    "data": {
        "exampleState": {
            "initialValues": {
                "exampleParentObject": {
                    "exampleObject": {
                        "dbValue": "",
                        "type": "STRING"
                    }
                }
            },
            "meta": {}
        }
    }
}
```

For this example, the following structure will be created by the framework:



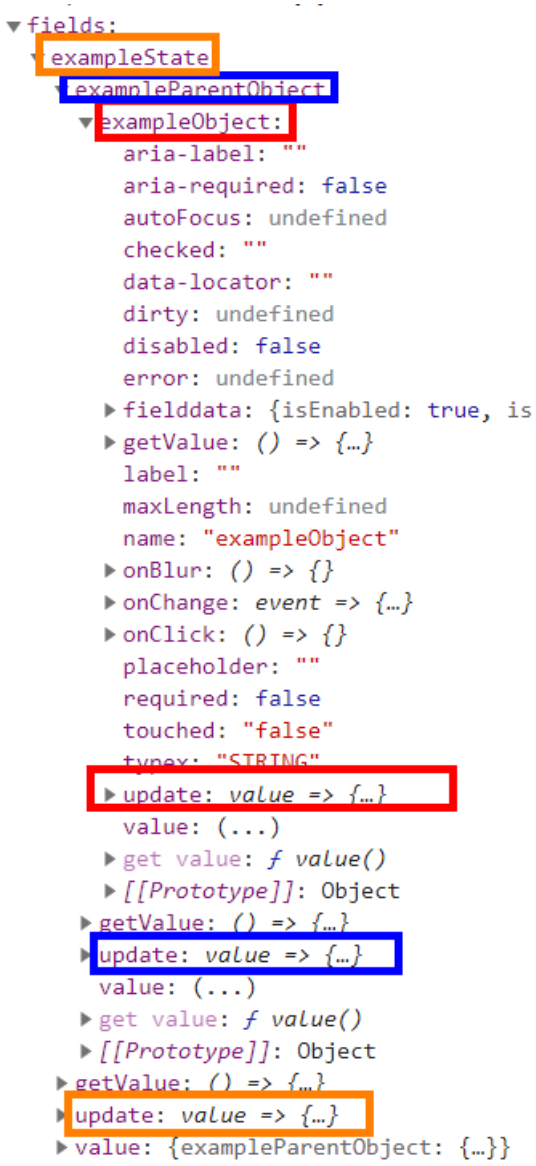

Consider now the following example containing a populated meta section:

```
{
  "data": {
    "exampleState": {
      "initialValues": {
        "exampleParentObject": {
          "exampleObject": {
            "dbValue": "",
            "type": "STRING"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            }
          }
        },
        "meta": {
          "exampleParentObject": {
            "exampleObject": {
              "field": "true",
              "type": "STRING"
            }
          }
        }
      }
    }
  }
```

This example shows that the intermediate nodes of the meta section are also converted to fields object walking back up the nested data structure to the root. These intermediate fields allow invoking the `update` callback function at any level:




**Note**

The `meta` section must be defined statically in the view model, and cannot be modified at run time.

## Updating atomic data

You can update atomic data in a declarative way only from the component in which it was defined. This can be done using action `outputData` configuration:


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
"syncStateFromUrl": {
    "actionType": "JSFunction",
    "method": "syncStateFromUrl",
    "inputData": {
        "provider": "{{props.provider}}",
        "searchState": "{{data.searchState}}"
    },
    "outputData": {
        "searchState": "searchState"
    },
    "deps": "js/awSearchSublocationService"
},
```

You can also update atomic data objects in child components (that's the main use case for atomic data) by:

- passing the atomic data object down the component hierarchy as a prop
- calling the `update` method available on the object from JavaScript code

Consider the following example, in which an atomic data object (`data.searchState`) is passed to the `aw-search-sublocation` component as a prop:

```
<aw-search-sublocation exist-when='data.searchState'
    provider='props.provider'
    sub-panel-context='{pageContext,...sublocationContext}'
    search-state='data.searchState'      You, seconds ago • Uncc
    base-selection='props.baseSelection'>
</aw-search-sublocation>
```

and then it is modified from JavaScript code:

```
"loadSearchData": {
    "actionType": "TcSoaService",
    "serviceName": "Internal-AWS2-2019-06-Finder",
    "method": "performSearchViewModel4",
    "inputData": {
        "searchInput": {
            "maxToLoad": "{{function:getDefaultPageSize}}",
            "maxToReturn": "{{function:getDefaultPageSize}}",
            "providerName": "{{props.subPanelContext.searchState.provider}}",
            "searchCriteria": "{{props.subPanelContext.searchState.criteria}}",
            "searchFilterMap6": "{{function:createFilterMap}}"
        },
        "inflateProperties": false,
        "noServiceData": false
    },
    "outputData": {
        "totalFound": "totalFound",
        "totalLoaded": "totalLoaded",
        "searchResults": "{{json:searchResultsJSON}}",
        "cursor": "cursor",
        "selectFirstObject": "{{function:processSelectionOfFirstObject}}",
        "output": "{{function:processOutput}}"
    },
    "deps": "js/awSearchService"
},
```

Atomic Data "search State" received as props in the child component. It is passed to a SOA call input.

Need to update the Atomic Data "search state" from the server response, hence a function is written to massage the atomic data

```
export const processOutput = ( serverResponse, searchData ) => {
    const newSearchData = { ...searchData.getValue() };
    newSearchData.totalFound = serverResponse.totalFound;
    newSearchData.totalLoaded = serverResponse.totalLoaded;
    newSearchData.endIndex = serverResponse.endIndex;
    newSearchData.searchFilterCategories = serverResponse.searchFilterCategories;
    searchData.update( newSearchData );
};
```

- "getValue()" function would give you the latest data.
- You should always use the getValue() function to avoid any stale data issue,

1. Get the latest value.
2. Mutate it according to your business logic.
3. Pass the updated value (newSearchData) to the Updater function.

```
export const processOutput = ( serverResponse, searchData ) => {
    searchData.totalFound = serverResponse.totalFound;
    searchData.totalLoaded = serverResponse.totalLoaded;
    searchData.endIndex = serverResponse.endIndex;
    searchData.searchFilterCategories = serverResponse.searchFilterCategories;
    searchData.update( searchData );

};
```

Wrong pattern, never mutate the "Atomic Data" directly without using the "getValue" function.

**Note**

When using JSX code, note that you must reference atomic data objects via the `fields` object:

```
import AwSearchSubLocation from 'viewmodel/AwSearchSubLocationViewModel';

export const searchLocationRenderFunction = ( { subPanelContext, viewModel, fields, formProp } ) => {
    return (
        <AwSearchSubLocation searchState={fields.searchState}></AwSearchSubLocation>
    );
};
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Complex atomic data example

Suppose that you want to implement the following complex data structure:

```
{
    "searchObject": {
        "searchString": false,
        "searchObjects": [
            {
                "id": "dis-1",
                "colorObjects": [
                    // These VMOs will be populated programmatically
                    { ...ViewModelObject1 },
                    { ...ViewModelObject2 }
                ],
                "colors": [
                    // These VMPs will be populated programmatically
                    { ...ViewModelProperty1 },
                    { ...ViewModelProperty2 }
                ]
            }
        ],
        "contextObject": ViewModelObject3 // VMO populated at runtime
        "contextColor": ViewModelProperty4, // VMP populated at runtime
        "searchHeader": "My Search"
    }
}
```

The corresponding metadata definition would be the following:

```
{
  "searchObject": {
    "initialValues": {
      "searchString": false,
      "searchHeader": "Initial Header"
    },
    "meta": {
      "searchObjects": [
        {
          "colorObjects": [
            {
              "field": "true",
              "type": "viewModelObject"
            }
          ],
          "colors": [
            {
              "field": "true",
              "type": "viewModelProperty"
            }
          ]
        }
```

```
      ],
      "contextObject": {
        "field": "true",
        "type": "viewModelObject"
      },
      "contextColor": {
        "field": "true",
        "type": "viewModelObject"
      }
    }
  }
}
```

**Tip**

For a more detailed example on using atomic data, see **this atomic data example**.

## Frequently asked questions

### How can I specify where fields are created within an atomic data object?

To configure a field, within the `meta` section you need to specify an object containing the following properties:

- `field`, set to `true`
- `type`, set to `viewModelProperty` or `viewModelObject`

For example:

```
{
    "meta": {
        "contextColor": {
            "field": "true",
            "type": "viewModelObject"
        }
    }
}
```

**Note**

It is possible to define fields only for ViewModelProperty or ViewModelObject objects.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Can I configure an Array as atomic data?

No, the root of an atomic data object must always be an object. The following example is incorrect and it will not work:

```
{
    "data": {
        // The root level object searchObject cannot be defined as an Array.
        "searchObject": [
            {
                //...
            }
        ]
    }
}
```

### How can I be notified when an atomic data object changes?

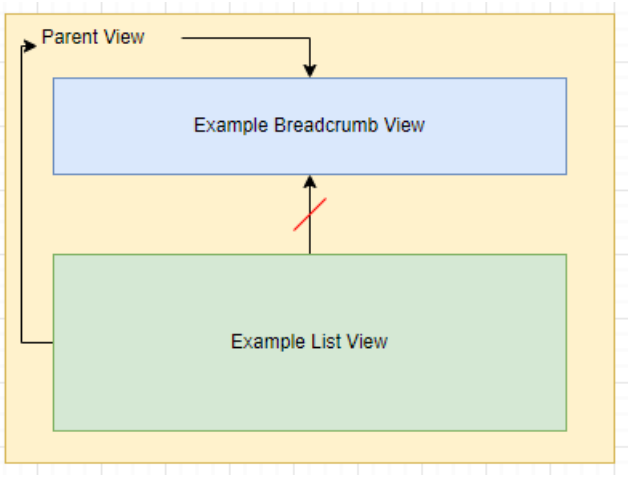
You can configure an **onUpdate hook** to observe changes to a specific atomic data object and configure an action to be executed when its value changes.

## Atomic data example

This is a detailed example to illustrate the usage of **atomic data**.

#### Requirements

Consider an interaction between two comonents, a list displaying items and a breadcrumb that shows the total count of the items in the list:



### Approach

To facilitate the communication between two sibling components, we need to lift the state up to the closest common parent and leverage the concept of atomic data. Direct communication between the two components makes them hard to re-use, while communication via clear prop-based interfaces makes them easy to be reused inside any parent component as long as their interface is honored.

The parent component here would define a common state which will be shared via props to both the child components (`Example List View` and `Example Breadcrumb View`).

When there is a change happening in `Example List View`, the common parent state needs to be updated, and the `Example Breadcrumb View` needs to listen to this state update and take the required action.

Now, remember that the **unidirectional flow** is read-only. For the child component `Example List View` to be able to update the parent state, we need to implement the **inverse flow** pattern, which can be achieved through **atomic data**.

#### Step 1: define atomic data in the parent component

In the `Parent View` component, define an atomic data object as follows:

```
{
    "data":{
        "sharedParentState":{
            "initialValues":{
                "totalFound" : 0
            },
            "meta": {}
        }
    }
}
```

#### Step 2: pass the atomic data object down from parent to child components

In the `Parent View` component, send the atomic data object down to the child components using the `parent-state` prop.

```
<aw-example-breadcrumb-view parent-state="data.sharedParentState"></aw-
example-breadcrumb-view>
<aw-example-list-view parent-state="data.sharedParentState"></aw-example-
list-view>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Step 3: update the atomic data object from the child component

In the `Example List View` component, update the atomic data object from JavaScript code (the `processOutput` function referenced in the `customAction` action):

```
{
    "actions":{
        "customAction": {
            "actionType": "JSFunction",
            "method": "customAction",
            "inputData": {
            },
            "outputData": {
                "output": "{{function:processOutput}}"
            },
            "deps": "js/customService"
        }
    },
    "functions": {
        "processOutput": {
            "functionName": "processOutput",
            "parameters": [
                "{{data}}",
                "{{props.parentState}}"
            ]
        }
    }
}
```

```
export const processOutput = ( data, parentState) => {
    const newParentState = { ...parentState };
    newParentState.totalFound = data.totalFound;
    parentState.update( newParentState );
};
```

### Step 4: Observe changes to the atomic data object in the `Example Breadcrumb View` child component

In the `Example Breadcrumb View` component, observe the updates on the atomic data object using an `onUpdate` **lifecycle hook**, and execute an action as needed:

```
{
  "lifecycleHooks": {
    "onUpdate": [
      {
        "action": "showTotalCount",
        "observers": [
          "props.parentState.totalFound"
        ]
      }
```

```
    ]
  },
  "actions": {
    "showTotalCount": {
      "actionType": "JSFunction",
      "method": "customActionToShowTotalCount",
      "inputData": {
        "totalFound": "{{props.parentState.totalFound}}"
      },
      "outputData": {
        "crumbsToShow": "crumbsToShow"
      },
      "deps": "js/customService"
    }
  }
}
```

# Selection Management

Selection refers to how users indicate specific items they intend to take action on.

In Siemens Web Framework, selection is associated with components such as aw-list, aw-splm-table, aw-splm-table tree, and aw-listbox. Users can select a specific item from the available list of items that the **Data Provider** fetches.

## selectionData

To announce the selection from a child component (having the source of the selection) to its component hierarchy, **selectionData** is used.

## selectionModel

A **selectionModel** contains a list of UIDs that are selected and has other internal state information such as multi-select state, selection mode, and selection status.

## How selection works with Data Provider

When a user interacts with components, for example, user selects an item in aw-list, the data provider updates selectedObjects for a new selection, which is provided by selectionModel.

Selections can be updated programmatically. Data providers support the firing selection change event using code.

```
eventBus.publish( dataProviderName + '.selectionChangeEvent',
{selectedObjects: objectToSelect, selectedUids:itemUidToSelect} );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Event Name | EventData | EventData Description |
|---|---|---|
| ⟨dataproviderName⟩.selectionChangeEvent | "selectedObjects":"ViewModelObject", "selectedUids":"ViewModelObject", "scope": "dataCtxNode " | "selectedObjects": The selected object's viewModelObject, "selectedUids": The selected object's viewModelObject,sometimes it's the same with selectedObjects, "scope": Data of this scope |

Please also refer to **selectionData**.

## Using the `selected` context and `uid` property

All items managed by data providers should have a uid property that uniquely identifies them.

When an item is selected by the user or programmatically, the selected context property is set to the selected data provider item. You can use this context to determine the value of the currently-selected item, but it is recommended not to change the value of this context directly.

## Updating the current selection in the URL using the s_uid state parameter

Every time an item is selected in the primary area, the `s_uid` parameter of the current sublocation will be added or updated automatically to the `uid` of the selected item (so that primary selection is preserved in case of a browser page reload).

The sublocation checks for the presence of a `s_uid` state parameter. If this state parameter is defined, its value will be used to pre-select the corresponding data provider item when retrieved. Specifying `s_uid` parameter in the URL of all sublocation states defined in your `states.json` files for which you want to manage selection is required.

The `s_uid` parameter can effectively be used to select items programmatically on page load. You can also select items programmatically by reloading the current state and specifying a `s_uid` parameter.

## Defining s_uid routing parameter in states.json

```
{
  "myFavs": {
    "url": "/myFavs",
    "data": {
      "label": {
        "source": "/i18n/locationAndSublocationMessages",
        "key": "favoritesSubLocationTitle"
      },
      "nameToken": "com.siemens.splm.client.movies:favoritesSubLocation",
```

```
      "clientScopeURI": ""
    },
    "params": {
      "s_uid": null
    },
    "parent": "movies",
    "view": "AwStandardSublocation"
  }
}
```

### References

- **List with Data Providers**
- **Selection summary**
- **Data provider**
- **Locations and sub location pattern**

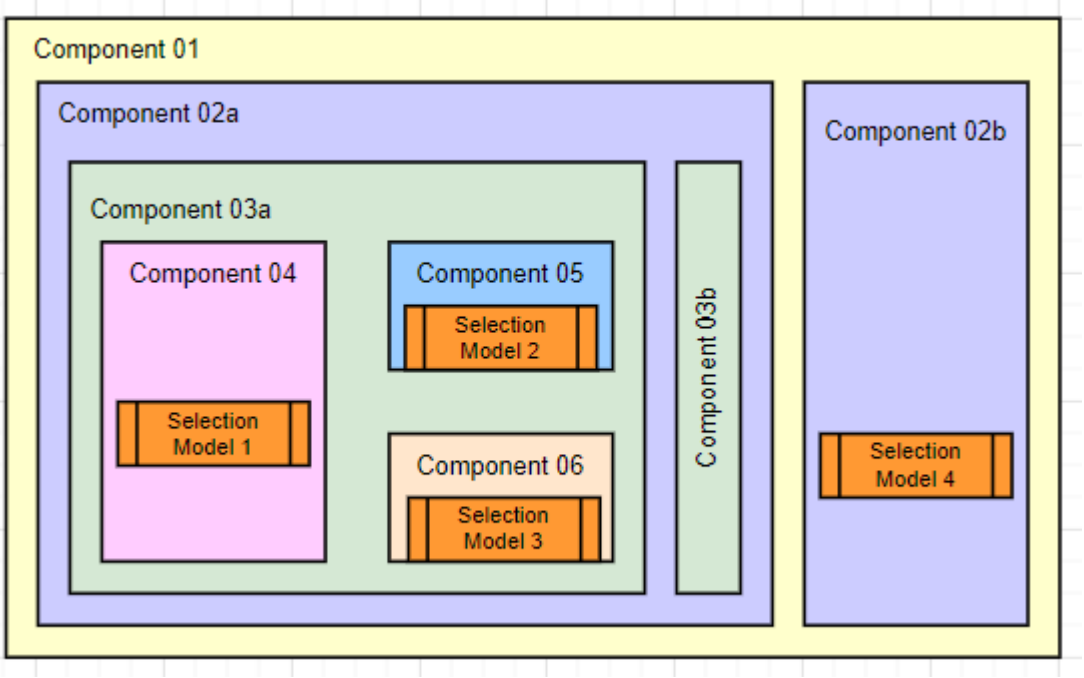
## selectionData

`selectionData` is a mechanism to manage the selection updates happening anywhere within the component's child hierarchy. It facilitates the component to get the complete handle over the selection changes and take the appropriate actions as required, for example - sharing the selection information to sibling components, firing events with selection information received, or updating the information on **global AppCtx**, etc.

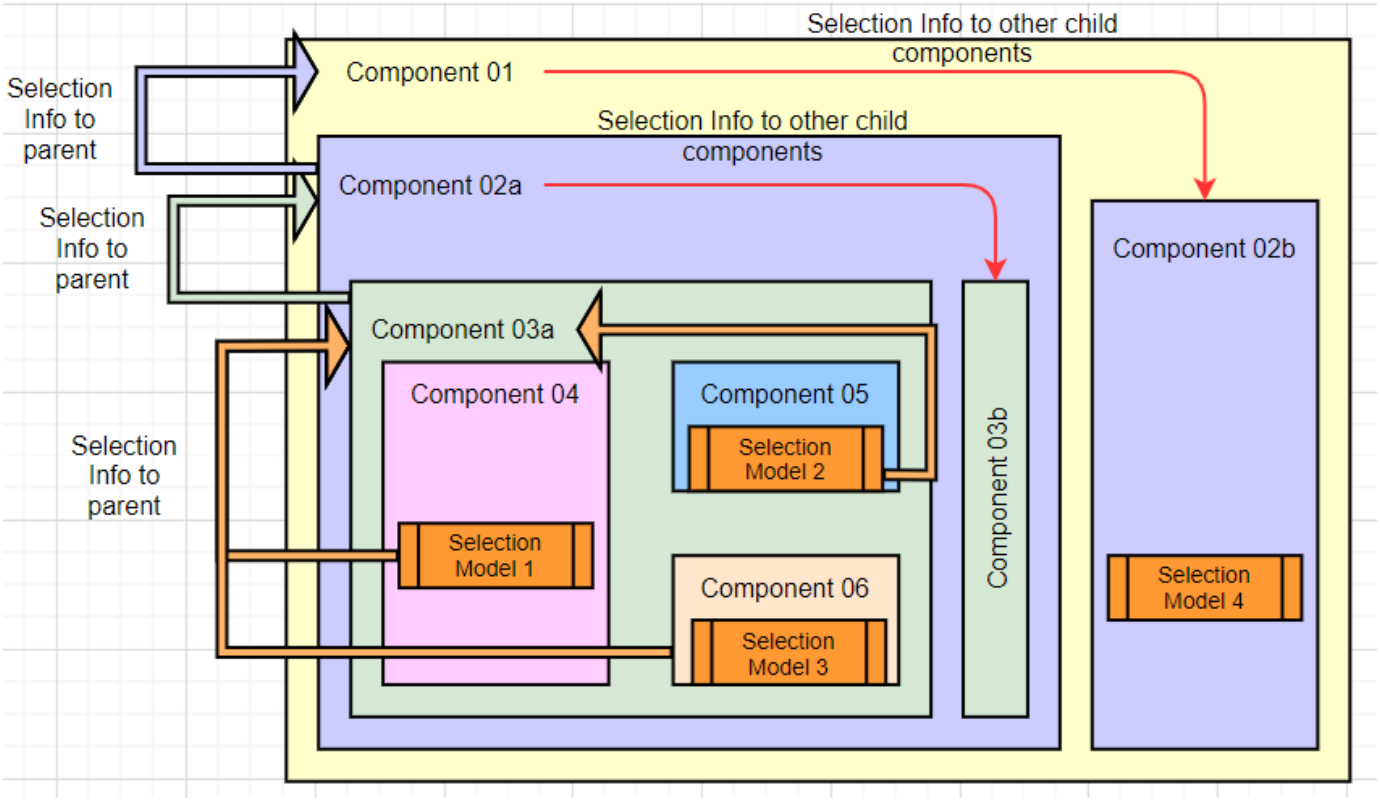
It allows the **inverse flow** of selection information from child components to the parent component while still following the rules of uni-directional data flow.

Consider the following component hierarchy:


Siemens Web Framework
Unpublished work. © 2025 Siemens

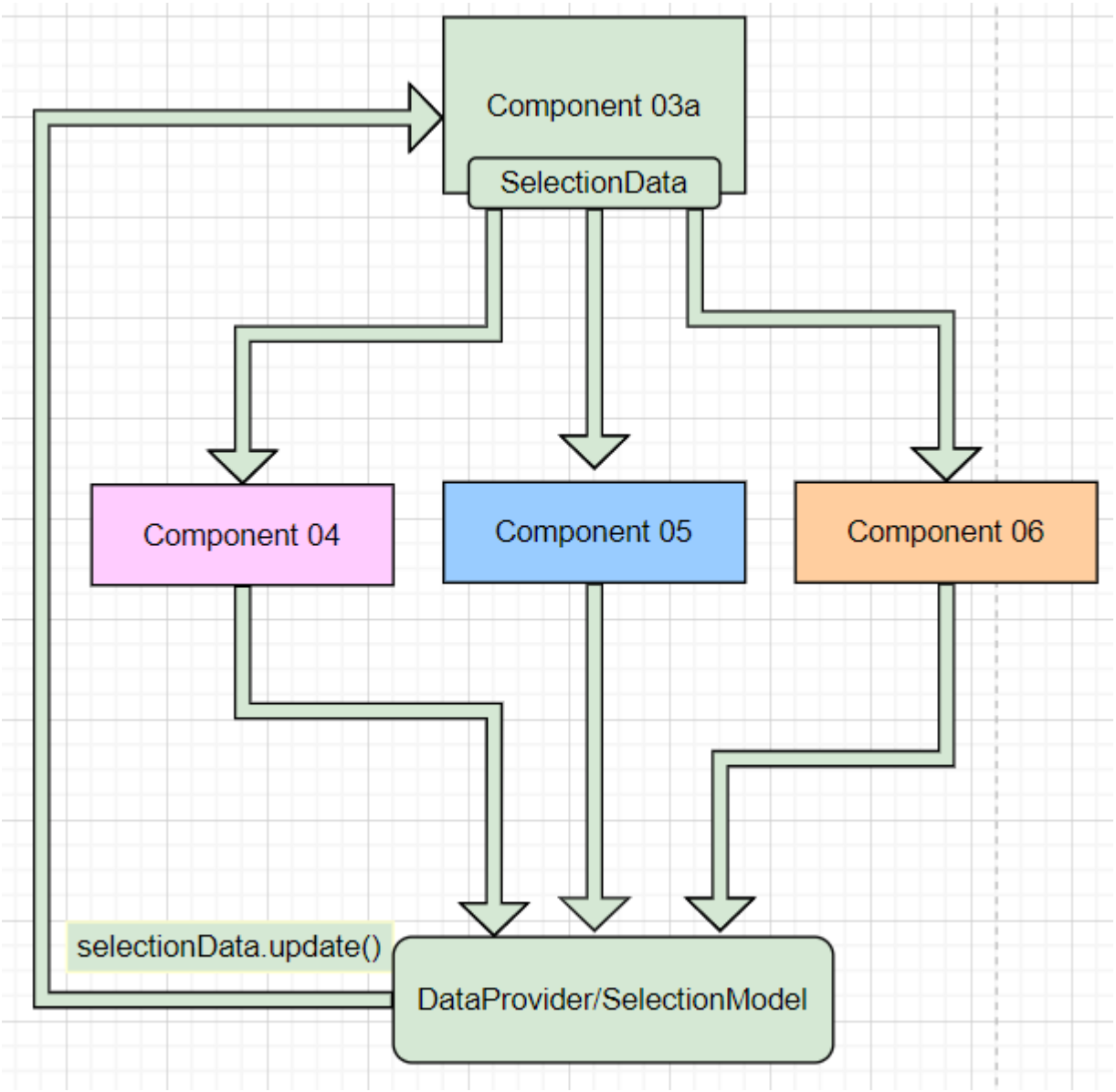



The selections (selection models) may be with a component at any level in the overall page hierarchy. However, the parent components would still need to know about them so that the information can be shared with the other sibling components in the page.



To facilitate this flow of information from child components to parent components, we need `selectionData`.

For Component03a in the above example, the `selectionData` will be created at that component, sent down, and whenever there is an actual update in selections, the `update` method on `selectionData` will get invoked, which is how the selections will be made available at the Component03a level.




It is implemented as an **atomic data**, which is why when the parent component sends it down, it is sent with an `update` method on which the child component can execute with the data available. For the `selectionData` use case, the child components would be invoking the parent component's `update` method with the selection information they have. This is how the information flows from selection models embedded in the nested component hierarchy to their immediate parents, who can then share it with their other child components, their parent components and so on.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Who defines the `selectionData`?

By default, many components in Framework like **primary work area (PWA)**, **secondary workarea (SWA)**, and **workarea (WA)** define `selectionData` so that they can manage the inverse flow of selections and inform the other page-level components about the selections being made in one of their child components. This is how selection propagation is being managed in the Framework layer.

### When to define `selectionData`?

If there is a custom component that needs to intercept/override the default selection flow available in the framework; then and only then the custom component would need to define its `selectionData` and make sure of the following:

1. Define your own `selectionData`
2. Send this created `selectionData` down in your component hierarchy
3. Listen to the updates in `selectionData`
4. Handle the updates - do custom logic here
5. Update the `selectionData` received from the parent

### How do I define `selectionData`?

`selectionData` needs to be defined in the `data` section of the view model, like this:

```
{
  "data": {
    "selectionData": {
      "initialValues": {},
      "meta": {}
    }
  }
}
```

The declarative engine will process this, and convert it into a corresponding **atomic data** object which will be made available on the component in which it is defined.

### How do I send `selectionData` down in the component hierarchy?

Similar to any other prop; `selectionData` can be passed down to the child components either as a direct prop or via subPanelContext, for example:

```
<aw-custom-component name="props.name" sub-panel-
context="props.subPanelContext" selection-data="data.selectionData"></aw-
custom-component>
```

We also need to make sure that `selectionData` is defined in the `props` section of the child component so that it can receive the information:

```
{
  "props": {
    "name": {
      "type": "object"
    },
    "sub-panel-context": {
      "type": "object"
    },
    "selection-data": {
      "type": "object"
    }
  }
}
```

The `selectionData` can be sent any level down to the required child component, which has access to selectionModel. The intermediate components need not play any other role than just ensuring that they are passing the `selectionData` received from the parent further down to the required child components.

### How do I associate `selectionData` with the child component having the selectionModel?

Once the `selectionData` is received in the child component which actually has the dataProvider or the selectionModel; it needs to be associated with them. In order to associate it with dataProvider, the `inputData` section of the dataProvider can be used:

```
{
  "dataProviders": {
    "customDataProvider": {
      "action": "customData",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}",
      "inputData": {
        "selectionData": "{{props.selectionData}}"
      }
    }
  }
}
```

Or suppose the component has the access to selectionModel. In that case, the `selectionData` can also be added explicitly on it which can be shared down to the dataProvider, again, by using the `inputData` section, for example:

```
export const AwCustomRenderFunction = ( props, data ) => {
    if( data.selectionModel && props.selectionData ) {
        data.selectionModel.selectionData = props.selectionData;
    }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    return includeComponent( viewBase + _.upperFirst( view ),
{ selectionModel: data.selectionModel} );
};
```

```
{
  "dataProviders": {
    "customDataProvider": {
      "action": "customData",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}",
      "inputData": {
        "selectionModel": "{{data.selectionModel}}"
      }
    }
  }
}
```

...or you could send both, for example:

```
{
  "dataProviders": {
    "customDataProvider": {
      "action": "customData",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}",
      "inputData": {
        "selectionData": "{{props.selectionData}}",
        "selectionModel": "{{props.selectionModel}}"
      }
    }
  }
}
```

The declarative engine will ensure processing this `inputData` and associating the sent information to the selectionModel or the corresponding dataProvider.

### How does `selectionData` gets updated with the actual selections?

Once the `selectionData` is associated with the dataProvider or the selectionModel, the declarative framework internally manages it and ensures that whenever a selection change happens inside the selection model; the `update` method available on the sent `selectionData` is invoked. It invokes the `update` method with the information related to selections which can be accessed on `selectionData.selected`.

### How to listen to updates in the `selectionData`?

To listen to the `selectionData` updates, we need to use the `onUpdate` **lifecycle hook**. The declarative framework will ensure that the corresponding `action` is invoked whenever there is a change in the specified observer, which is `selectionData` for us. This standard declarative action provides the

complete control to the component defining this observer to receive all the selection information and manage it as per the business requirements.

See the following example:

```
{
  "actions": {
    "customSelectionChangeAction": {
      "actionType": "JSFunction",
      "method": "customSelectionChangeHandler",
      "inputData": {
        "localSelectionData": "{{data.selectionData}}",
        "parentSelectionData": "{{props.selectionData}}"
      },
      "deps": "js/CustomService"
    }
  },
  "lifecycleHooks": {
    "onUpdate": [
      {
        "action": "customSelectionChangeAction",
        "observers": [
          "viewModel.atomicData.selectionData"
        ]
      }
    ]
  }
}
```

Please note that the updated selection information is available under `data.selectionData`. `customSelectionChangeHandler` here is the custom action which has the information about the selections and is free to implement any business logic as required by the component.

### How to propagate the updates in `selectionData` up in the component hierarchy?

This is required only if a component is intercepting the flow of `selectionData`. In this case, the intercepting component will define its `selectionData`, send it down, listen to it and do the required custom processing. Once done, it should resume the information flow by invoking the available `update` method on the `selectionData` received from the parent component.

See the following example:

```
export const customSelectionChangeHandler= ( localSelectionData,
parentSelectionData ) => {
    if( !_.isEmpty( localSelectionData ) ) {
        // Custom handling for selections goes here - as required
        // Invoke the update on the parentSelectionData with either same
selection data OR modified
        // selection data OR additional selection data - basically whatever
is required by the component
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        parentSelectionData.update( localSelectionData );
};
```

### How do I listen to selection change in the same view model using selectionData?

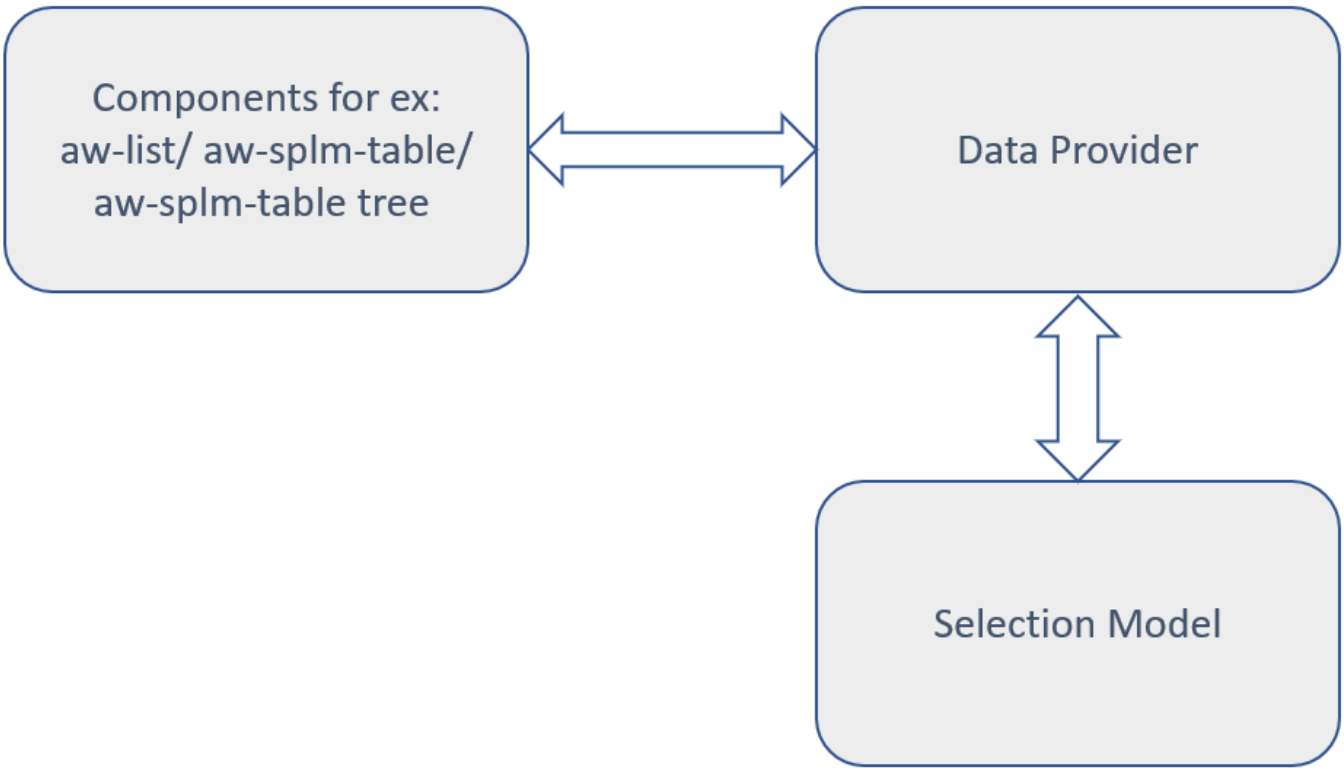
1. You need to first define selection data
2. Associate that selectionData with a data provider

```
{
   "dataProviders": {
     "customDataProvider": {
       "action": "customData",
       "response": "{{data.searchResults}}",
       "totalFound": "{{data.totalFound}}",
       "inputData": {
         "selectionData": "{{fields.selectionData}}"
       }
     }
   }
 }
```

3. Listen to selection change in the data provider using selectionData
4. Write your business logic in an observer action that you want to perform on selection change.

## selectionModel

- A selection model contains a list of selected UIDs and other internal state information such as multi-select state, selection mode, and selection status.
- The selection model is used in the data provider mainly responsible for maintaining all the selection objects' information and provides APIs to access information.




## Creating selectionModel

The following example demonstrates how to create the selection model declaratively.

```
{
  "selectionModels": {
    "yourSelectionModel": {
      "selectionMode": "multiple",
      "selectionTracker": [
        "alternateID",
        "uid",
        "props.targetObject.uid"
      ]
    }
  }
}
```

Here:

- `selectionModelMode` is the mode of the selections you want to support, either `single` or `multiple`.
- `selectionTracker` is the array of attributes that can be used to identify objects uniquely while performing selection. In example above, `selectionModel` will use these attributes in sequence to identify the selection ID to set selection.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Associating custom selectionModel with dataProvider

Following example demonstrates how to associate the created selectionModel with the dataProvider.

```
{
  "dataProviders": {
    "performSearch": {
      "action": "getMovies",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}",
      "selectionModelMode": "single",
      "inputData": {
        "selectionModel": "{{subPanelContext.selectionModel}}"
      }
    }
  }
}
```

### Sharing of selectionModel

The following example demonstrates how to share the selection model when using the **AwPrimaryWorkarea**.

### selectionModel APIs

Please refer **Selection Model APIs**

## Form Validation

Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This is called client-side form validation and helps ensure data submitted matches the requirements specified in the form controls. This wiki leads you through basic concepts and examples of client-side form validation.

Client-side validation is an initial check and an important feature of a good user experience; by catching invalid data on the client-side, the user can fix it straight away. If it gets to the server and is then rejected, a noticeable delay is caused by a round trip to the server and then back to the client-side to tell the user to fix their data.

In Siemens Web Framework, form validation has two main parts

- Widget-level validation, which can also be divided into the following parts:
  - Widget type This is a built-in validation in which the ViewModelProperty type determines the user input value.


Unpublished work. © 2025 Siemens

- `isRequired` `isRequired` is another built-in property supported on ViewModelProperty objects. Declaring `isRequired` as `true` enforces the user to enter a value before form validation.
- Validation criteria `validationCriteria` is again another optional property on every **ViewModelProperty** object. You can use validation criteria to attach their custom validation with a widget, and you can attach single/multiple validation criteria on a widget. For the form validation successful, all the validation criteria attached with the widget must be successful.

- Form-level - A form is considered *valid* only when all the defined validations (`isRequired` or `ValidationCriteria`) on all ViewModelProperties defined on the component (view model) are successful.

## Simple Form Validation

Consider the following panel defining a simple login form:

```
<aw-command-panel>
  <aw-panel-body>
    <aw-textbox prop="data.email"></aw-textbox>
    <aw-password prop="data.password"></aw-password>
  </aw-panel-body>
  <aw-panel-footer>
    <aw-button action="login" enable-when="formProp.attributes.valid"
>Login</aw-button>
  </aw-panel-footer>
</aw-command-panel>
```

...and the corresponding `data` section in the view model:

```
{
  "data": {
    "email": {
      "type": "String",
      "isRequired": "true",
      "isEditable": "true",
      "dbValue": "",
      "displayName": "Email"
    },
    "password": {
      "type": "String",
      "isRequired": "true",
      "isEditable": "true",
      "dbValue": "",
      "displayName": "Password"
    }
  }
}
```

The `aw-panel-body` custom element always creates a form, so it is mandatory to use it to perform form validation. In this case, the `Login` button must be enabled only when a value is entered both in


Siemens Web Framework
Unpublished work. © 2025 Siemens

the email and password field. Hence, you need to specify `"isRequired": "true"` for both of the ViewModelProperty objects in the view model.

Now the value of `formProp.attributes.valid` will come as `true`, only when you have entered a value for both fields.

**Note**

If you enter *any* character, including spaces, in a field marked as required, the field will be considered valid. If you want to consider the field blank if it only contains spaces, you have to specify your own user-defined validation criteria as explained in the following section.

## User-defined validations

There are situations, however, that require more complex validation rules and you may need to attach some very specific validations to the widget.

In such cases, define some validation criteria for the property. To attach the validation criteria, `validationCriteria` property should be added to the object as shown below.

```
{
  "data": {
    "email": {
      "type": "STRING",
      "validationCriteria": [
        {
          "condition": "conditions.invalidEmail",
          "value": "The value provided is not a valid email address"
        }
      ],
      "displayName": "Email"
    }
  }
}
```

In this case, one validation criterion is provided, and it is based on the condition of a regular expression.

The condition `conditions.invalidEmail` can be defined as follows:

```
{
  "conditions": {
    "invalidEmail": {
      "expression": {
        "$source": "data.email.dbValue",
        "$query": {
          "$notinregexp": "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9_]+?\\.[a-zA-Z]
{2,3}$"
        }
      }
```

```
        }
    }
}
```

**Tip**

See **Conditions** and **Validation criteria** for more information.

## Additional Examples

The following example shows how to implement form validation in JSX code:

```
<AwPanel
    className='w-6'
    caption='This is Panel Caption'
    {...formProp} >

    <AwPanelHeader>
        <div>{data.subHeader}</div>
    </AwPanelHeader>
    <AwPanelBody>
        <AwPanelSection caption='Form Fields'>
            <AwTextbox {...fields.email}/>
            <AwTextbox {...fields.firstName}/>
            <AwTextbox {...fields.lastName}/>
            <AwWidget {...fields.age}/>
            <AwWidget {...fields.height}/>
            <AwTextbox {...fields.department}/>
            <AwWidget {...fields.feedback}/>
            <AwDate {...fields.hireDate}/>
        </AwPanelSection>
    </AwPanelBody>
    <AwPanelFooter className='align-right'>
        <AwButton type='button' isDisabled={!formProp.attributes.dirty} Reset </AwButton>
        <AwButton type='submit' isDisabled={!formProp.attributes.valid}>Submit</AwButton>
    </AwPanelFooter>
</AwPanel>
```

The following example shows how to implement form validation in a declarative view template:

```
aw_60_a > src > thinclient > validationcontractjs > src > html > Crt1CreateValidationContractView.html > aw-command-panel > aw-form > aw-panel-body > aw-add
1   <aw-command-panel caption="i18n.Create_AR_Panel_Title" anchor="data.addObjectPanelPinCommands" context="data">
2       <aw-form name="commandPanelForm" visible-when="data.activeView!='assignProjectSub'&&data.activeView!='addReferenceSub'">
3           <aw-panel-body>
4               <aw-add max-recent-count='3' visible-tabs="new" type-override-id="CreateAR"
5                   include-types="Crt0VldnContract" sub-panel-context="{ callback: data.callbackObj }">
6               </aw-add>
7           </aw-panel-body>
8       </aw-form>
9       <aw-checkbox prop="data.openOnCreate" visible-when="conditions.isNewTabAddButtonVisible" enable-when="data.pinnedToForm.dbValue
10      <aw-panel-footer visible-when="data.activeView!='assignProjectSub'&&data.activeView!='addReferenceSub'">
11          <!-- panel-footer button for New tab -->
12          <aw-button action="initCreateInputs" visible-when="formProp.attributes.valid && conditions.isNewTabAddButtonVisible">
13              <aw-i18n>i18n.Create_Button_Title</aw-i18n>
14          </aw-button>
15      </aw-panel-footer>
16  </aw-command-panel>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Form-level API Usage examples

The following example shows how to reset a form:

```
 <aw-button action="resetForm">Reset</aw-button>
```

```
{
    "resetForm": {
            "actionType": "JSFunction",
            "method": "resetAllForm",
            "inputData": {
                "data":"{{data}}",
                "formProp":"{{formProp}}"
            },
            "deps": "js/showcaseFormUtils"
        }
}
```

```
export let resetForm = function( data, formProp ) {
    let listProp = { ...data.listBoxArrayReset };
    listProp.dbValue = [ 'State 5' ];
    listProp.uiValue = [ 'State 5' ];
    let val  =  { ...data.listBoxReset };
    val.dbValue = 'Iowa';
    val.uiValue = 'Iowa';
    let dateValue = { ...data.dateDetails };
    uwPropService.setValue( dateValue, new Date( '10/10/2022' ),
'10/10/2022' );
    let listArray = { ...data.xyzList };
    listArray.uiValue = '123, 465';
    listArray.dbValue = [ '123', '465' ];

    // Below is method to be invoked for reset with set of values that
needs to be updated

    formProp.reset( { listBoxReset:val, showcaseAtomic :
atomicValue, dateDetails :dateValue,  listBoxArrayReset : listProp,
xyzList:listArray } );

};
```


Unpublished work. © 2025 Siemens

The following example shows how to clear a form :

```
 <aw-button action="clearForm">Clear</aw-button>
```

```
{
    "clearForm": {
            "actionType": "JSFunction",
            "method": "clearForm",
            "inputData": {
                "formProp":"{{formProp}}"
            },
            "deps": "js/showcaseFormUtils"
        }
}
```

```
export let clearForm = function( formProp ) {
    formProp.clear();
};
```

## Advanced use cases

The following section include some more examples on how to implement validation in more advanced use cases.

## Programmatic update of widgets

In many use cases, you may need to update the widget value programmatically, and you may receive the widget value through some REST API call after the component is already mounted. In these cases you need to update the widget value and also trigger the validation programmatically.

Consider the following example view and view model. In this case a sample value is retrieved from a REST endpoint and stored to `data.sample`:

```
<aw-panel-body>
    <aw-panel-section>
        <aw-widget prop="data.sample"/>
    </aw-panel-section>
</aw-panel-body>
```

```
{
  "actions": {
    "fetchValue": {
      "actionType": "RESTService",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "method": "GET",
      "inputData": {
        "request": {
          "method": "GET",
          "withCredentials": false,
          "url": "/my-api/get-sample-value/"
        }
      },
      "outputData": {
        "sample": "{{functions:updateWidget}}"
      }
    }
  },
  "functions": {
    "updateWidget": {
      "functionName": "updateWidget",
      "parameters": [
        "{{fields.sample}}"
      ]
    }
  },
  "data": {
    "sample": {
      "type": "String",
      "isRequired": "true",
      "dbValue": "",
      "displayName": "sample"
    }
  },
  "lifecycleHooks": {
    "onMount": "fetchValue"
  }
}
```

In this case, you need to manually update the `sample` field to make sure validation is triggered:

```
export updateWidget = (restOutput, sampleField)=> {
    let value = restOutput.value;
    sampleField.update(value, { isEditable: true });
}
```

## Dynamic Widgets

In some cases, you may need to update the properties of a ViewModelProperty object dynamically. For example, you may need to mark a ViewModelProperty object as required only when it becomes visible.

Consider the following view:

```
<aw-panel-body>
    <aw-panel-section caption="i18n.properties">
        <aw-widget exist-when="data.selected.caw0Context!=='Capa'
```

```
            prop='data.capImage'"></aw-widget>
    </aw-panel-section>
</aw-panel-body>
```

In this case, if a widget is not visible based on an `exist-when` condition, it should not be marked as required because it cannot be filled in.

Instead, you should configure an `onUpdate` hook so that the the ViewModelProperty object (`data.capImage`) is set as required only when it is displayed. Consider the following view model associated to the previous view:

```
{
  "actions": {
    "updateIsRequiredFlag": {
      "actionType": "JSFunction",
      "method": "updateIsRequiredFlag",
      "inputData": {
        "capImage": "{{fields.capImage}}",
        "caw0Context": "{{data.selected.caw0Context}}",
        "creationType": "{{data.creationType}}"
      }
    }
  },
  "data": {
    "capImage": {
      "type": "String",
      "isRequired": "false",
      "isEditable": "true",
      "dbValue": "",
      "displayName": "sample"
    }
  },
  "lifecycleHooks": {
    "onUpdate": [
      {
        "action": "updateIsRequiredFlag",
        "observers": [
          "data.selected.caw0Context",
          "data.creationType"
        ]
      }
    ]
  }
}
```

The `updateIsRequiredFlag` action is called when the value of `data.selected.caw0Context` or `data.creationType` changes, and it executes the following function:

```
export const updateIsRequiredFlag = ( capImage, caw0Context, creationType ){
    if ( creationType && caw0Context === 'Capa' ) {
        capImage.update( capImage.value, { isRequired: true } );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    } else {
        capImage.update( capImage.value, { isRequired: false } );
    }
}
```

Note how, in this case, you can use the `update` method of a field also to update its properties like `isRequired`, besides its value.

**Note**

The `updateIsRequiredFlag` operates on the field object associated to `data.capImage`. Therefore, you need to pass `fields.capImage` instead of `data.capImage`.

**Tip**

See also the Form Validation example in **Siemens Web Framework Storybook**.

## Validation Criteria

Validation criteria is a declarative way to validate property values on the client side. It helps to ensure user fills out each form field with correct format and value. The validation criteria can be attached to a property by defining 'validationCriteria' key on the corresponding viewmodel property using the following syntax:

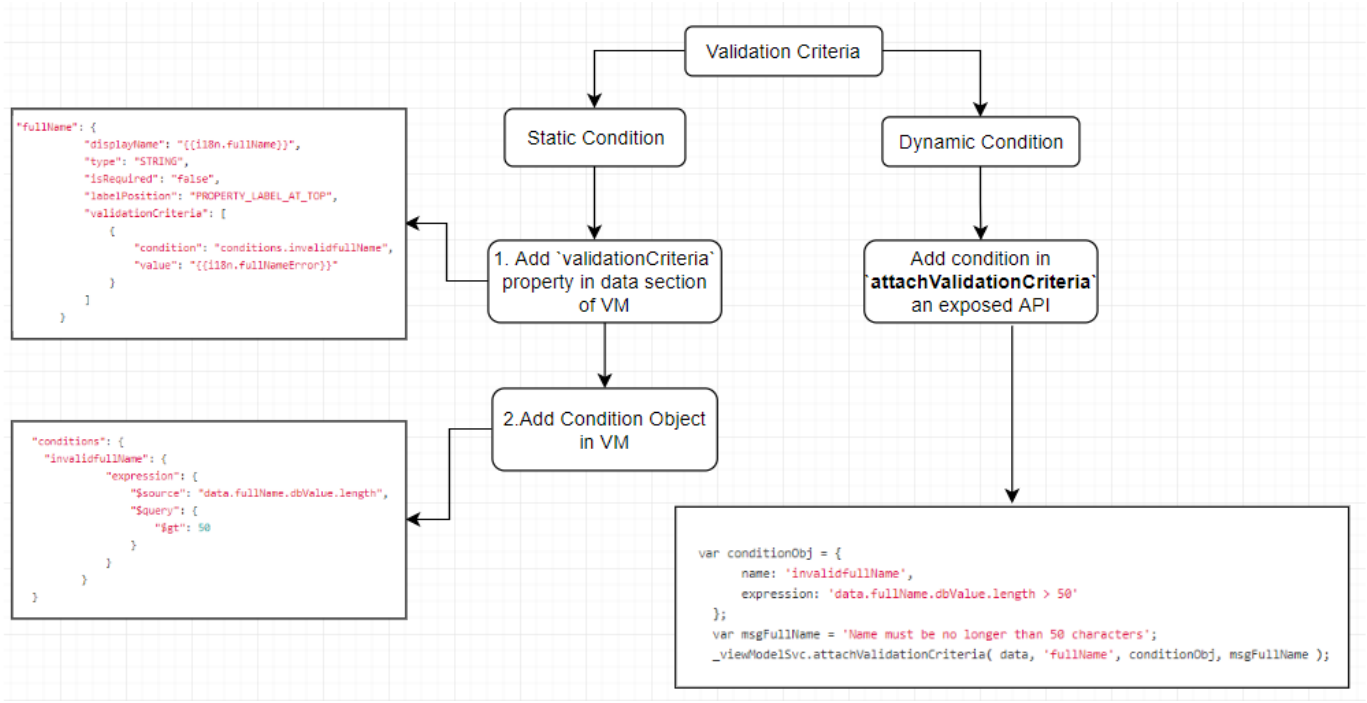
```
{
  "viewmodel_property_name": {
    "displayName": "Name",
    "type": "STRING",
    "isRequired": true,
    "validationCriteria": {
      "condition": "<condition_name>",
      "value": "<msg>"
    }
  }
}
```

**condition** - condition name which will be part of conditions object in view model json. **value** - An error message which will render on screen when condition is valid.

### Attaching Validation Criteria

There are two ways to define validation criteria:

1. Static
2. Dynamic




Let us take an use case to understand the how to define and attach validation criteria.

For the form given below, I would like to validate the 'Full Name' property. An error message should be shown if the 'Full Name' value goes beyond 50 characters.

```
<aw-panel>
      <aw-panel-body>
         <aw-form name="mainPanelForm">
         <!-- Full Name -->
            <aw-textbox prop="data.fullName"></aw-textbox>
            <aw-numeric prop="data.mobileNumber"></aw-numeric>
            <aw-button> {{i18n.submit}}</aw-button>
         </aw-form>
      </aw-panel-body>
</aw-panel>
```

Now lets understand how to define the validation criteria for "data.fullName" property object in the view model json :

## Static Validation Criteria

*Data Object*


Siemens Web Framework
Unpublished work. © 2025 Siemens

In the "data" section of view model, "fullName" is a property object which is bound with aw-textbox in the view. In order to attach the validation criteria, **validationCriteria** property should be added to the object as shown below.

```
{
  "fullName": {
    "displayName": "{{i18n.fullName}}",
    "type": "STRING",
    "isRequired": "false",
    "labelPosition": "PROPERTY_LABEL_AT_TOP",
    "validationCriteria": [
      {
        "condition": "conditions.invalidfullName",
        "value": "{{i18n.fullNameError}}"
      }
    ]
  }
}
```

Lets have a look on condition object of view model json

*Condition Object*

The condition "invalidfullName" can be defined as follows. Please refer **conditions** for more information.

```
{
  "conditions": {
    "invalidfullName": {
      "expression": {
        "$source": "data.fullName.dbValue.length",
        "$query": {
          "$gt": 50
        }
      }
    }
  }
}
```

## Dynamic Validation Criteria

In the "data" section of view model, "fullName" is a property object to be bound with a aw-textbox in view.

```
{
  "fullName": {
    "displayName": "{{i18n.fullName}}",
    "type": "STRING",
    "isRequired": "false",
    "labelPosition": "PROPERTY_LABEL_AT_TOP"
```

```
    }
}
```

In order to add restriction dynamically on the length of the value entered, use `attachValidationCriteria` method which is in `viewModelService` file. This method initializes the condition service, register listeners for conditions using the given scope and also creates a listener for condition for a dynamically created viewmodel property. The parameters for this method are:

1. declViewModel

   - The object whose properties will be updated when the associated conditions are attached.

2. propertyName

   - The dynamically created propertyName who properties will be updated when the associated conditions are updated.

3. conditionObj

   - the conditions object can be defined in one of the following formats :

```
{
    "name": "invalidfullName",
    "expression": "data.fullName.dbValue.length > 50"
}
```

**OR**

There are a finite set of operators that can be used inside a conditions expression. See **Conditions** for more detailed information.

```
{
    "name": "invalidfullName",
    "expression": {
        "$source": "data.fullName.dbValue.length",
        "$query": {
            "$gt": 50
        }
    }
}
```

1. msgString

   - The message which will be shown if the validation criteria doesn't match

2. dataCtxNode

   - The 'dataCtxNode' (aka '$scope') to register the expression watch against.


Siemens Web Framework
Unpublished work. © 2025 Siemens

Dynamic condition should be added in the user defined JS function. The user defined JS function can be called by any event/action. The user can refer to the following example of view model

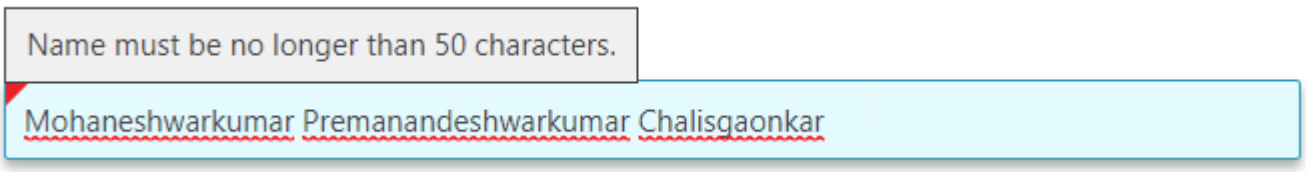
```
{
  "actions": {
    "xxxMyCustomAction": {
      "actionType": "JSFunction",
      "method": "xxxUserDefinedJSFunction",
      "inputData": {
        "data": "{{data}}"
      },
      "deps": "js/xxxCustomJSFile"
    }
  },
  "lifecycleHooks": {
    "onMount": "xxxMyCustomAction"
  }
}
```

In the above example, a validation is needed when "xxxViewName" is loaded. Thus when "xxxViewName" loads, an action "xxxMyCustomAction" is triggered which is responsible for the dynamic validation.

Refer to the below code snippet to add restriction on the length of the text dynamically in "xxxCustomJSFile":

```
export let xxxUserDefinedJSFunction = function(data){
  var conditionObj = {
        name: 'invalidfullName',
        expression: 'data.fullName.dbValue.length > 50'
    };
    var msgFullName = 'Name must be no longer than 50 characters';
    _viewModelSvc.attachValidationCriteria( data, 'fullName', conditionObj,
msgFullName );
}
```

Error message:




## Validation criteria usage illustrations

**1 ) \$gt /\$gte /\$lt /\$lte /\$eq /\$ne**

*Data Object:*

```
"data": {
    "iterationEndDate": {
        "displayName": "{{i18n.endDate}}",
        "type": "DATE",
        "labelPosition": "PROPERTY_LABEL_AT_SIDE",
        "validationCriteria": [
            {
                "condition": "conditions.invalidEndDate",
                "value": "Iteration End date has to be greater than
Iteration start Date."
            }
        ]
    },

}
```

*Condition Object:*

```
"conditions": {
"invalidEndDate": {
            "expression": {
                "$source": "data.invalidEndDate.dbValue",
                "$query": {
                    "$lt": "Date({{data.iterationStartDate.dbValue}})"
                }
            }
        }}
```

Iteration end date should throw error if iteration end date is less than iteration start date.

**2) \$notinrange**

2.1) Date widget

**Note**

: For date widgets the data context node to \$query should be specified in **Date({})** format. Here, you need to wrap data context object of date inside 'Date' keyword along with parenthesis. 'Date' keyword is only applicable for date widgets.

*Data Object:*

```
"data": {
"dueDate": {
            "displayName": "{{i18n.dueDate}}",
            "type": "DATE",
            "isRequired": "false",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "dbValue": "",
            "labelPosition": "PROPERTY_LABEL_AT_SIDE",
            "validationCriteria": [
                {
                    "condition": "conditions.invalidDate",
                    "value": "{{i18n.dueDateError}}"
                }
            ]
        },
        "creationDate": {
            "displayName": "{{i18n.creationDate}}",
            "type": "DATE",
            "isRequired": "false",
            "dbValue": "",
            "labelPosition": "PROPERTY_LABEL_AT_SIDE"
        },
        "expirationDate": {
            "displayName": "{{i18n.expirationDate}}",
            "type": "DATE",
            "isRequired": "false",
            "dbValue": "",
            "labelPosition": "PROPERTY_LABEL_AT_SIDE"
        }}
```

*Condition Object:*

```
    "invalidDate": {
            "expression": {
                "$source": "data.dueDate.dbValue",
                "$query": {
                    "$notinrange": [
                        "Date({{data.creationDate.dbValue}})",
                        "Date({{data.expirationDate.dbValue}})"
                    ]
                }
            }
```

Here, Due date should throw error if due date is not between Creation Date and Expiry Date.

2.2) aw-widget for String

*Data Object:*

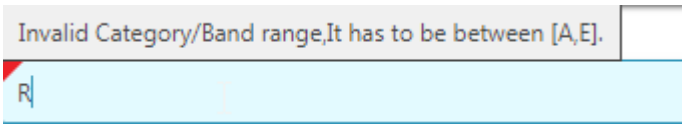
```
"category": {
            "displayName": "{{i18n.category}}",
            "type": "STRING",
            "maxLength": 1,
            "isRequired": "false",
            "dispValue": "SamplePlantName",
```

```
            "dbValue": "",
            "validationCriteria": [
                {
                    "condition": "conditions.invalidCategory",
                    "value": "{{i18n.categoryError}}"
                }
            ]
        }
```

*Condition Object :*

```
        "invalidCategory": {
            "expression": {
                "$source": "data.category.dbValue",
                "$query": {
                    "$notinrange": [
                        "A",
                        "E"
                    ]
                }
            }
        }
```

*Error Message*




2.3) aw-widget for Number /Double

*Data Object :*

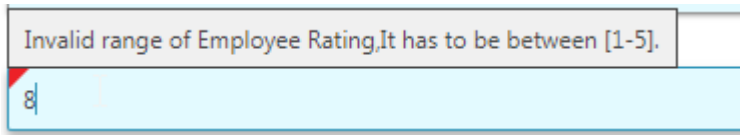
```
    "employeeRating": {
            "displayName": "{{i18n.employeeRating}}",
            "type": "INTEGER",
            "labelPosition": "PROPERTY_LABEL_AT_TOP",
            "validationCriteria": [
                {
                    "condition": "conditions.invalidRating",
                    "value": "{{i18n.employeeRatingError}}"
                }
            ]
        }
```


Unpublished work. © 2025 Siemens

*Condition Object :*

```
    "invalidRating": {
            "expression": {
                "$source": "data.employeeRating.dbValue",
                "$query": {
                    "$notinrange": [
                        1,
                        5
                    ]
                }
            }
        }
```

*Error Message*




**3) \$and /\$$or:**

Also, you can combine 2 conditions inside \$and /\$or like below:

*Data Object:*

```
"investmentDate": {
            "displayName": "{{i18n.investmentDate}}",
            "type": "DATE",
            "isRequired": "false",
            "dbValue": "",
            "labelPosition": "PROPERTY_LABEL_AT_SIDE",
            "validationCriteria": [
                {
                    "condition": "conditions.invalidDate",
                    "value": "{{i18n.dueDateError}}"
                }
            ]
        }
```

*Condition Object :*

```
      "conditions": {
        "invalidDate": {
            "expression": {
                "$or": [ {
                        "$source": "data.investmentDate.dbValue",
                        "$query": {
```

```
                        "$lt":
"Date({{data.financialYearStartDate.dbValue}})",
                        }
                    },
                    {
                        "$source": "data.investmentDate.dbValue",
                        "$query": {
                        "$gt":
"Date({{data.financialYearEndDate.dbValue}})",
                        }
                    }
                ]
            }
        }
```

Investment date will throw error if it is less than financial year start date or greater than financial year end date.

**4) \$notinregexp** *Data Object :*

```
     "emailId": {
            "displayName": "{{i18n.emailId}}",
            "type": "STRING",
            "isRequired": "true",
            "isEditable": "true",
            "isRequired": "false",
            "labelPosition": "PROPERTY_LABEL_AT_TOP",
            "validationCriteria": [ {
                "condition": "conditions.invalidEmailId",
                "value": "{{i18n.emailIdeError}}"
            } ]
        }
```

*Condition Object :*

```
     "invalidEmailId": {
            "expression": {
                    "$source": "data.emailId.dbValue",
                    "$query": {
                        "$notinregexp": "^[a-z0-9_.+-]+@[a-z0-9-]+\\.[a-
z0-9-.]+$"
                    }
            }
        }
```

Email Id will throw error if it doesn't satisfy regExp.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Accessing the DOM

As of version 5, Siemens Web Framework leverages the React framework internally to render user interfaces. Like other modern JavaScript frameworks, React uses a **virtual DOM** implementation to perform fast comparisons and updates of the real **DOM** rendered on the page.

It is important to understand therefore that:

- the virtual DOM is just a lightweight *representation* of the DOM
- changes to the virtual DOM will trigger a re-render of components and updates to the DOM
- direct changes to the DOM will *not* be automatically reflected on the virtual DOM

As a general rule, you should always operate at virtual DOM level (by leveraging declarative views and JSX render functions in **components**) for best performance and more control over the application state. However, in certain situations, such as when integrating third-party components, you may need to access the DOM directly.

To obtain a reference to a DOM element and perform DOM manipulations:

1. Define a named reference object in the render function's options (`splitter`, in this example).

```
{
     "schemaVersion": "1.0.0",
     "lifecycleHooks": {
         "render": "render",
         "onMount": "initialize"
     },
     "actions": {
         "render": {
             "actionType": "JSFunction",
             "method": "awSplitterRenderFunction",
             "deps": "js/AwSplitterService",
             "options": {
                 "elementRefList": [ "splitter" ]
             }
         },
         "initialize": {
             "actionType": "JSFunction",
             "method": "initSplitter",
             "deps": "js/AwSplitterService",
             "inputData": {
                 "reflist": "{{parameters.elementRefList}}"
             }
         }
     }
}
```

2. Bind the reference object named `splitter` inside the render function with a given virtual DOM element:


Unpublished work. © 2025 Siemens

- Use the `ref` keyword for plain HTML elements:

```
export const awSplitterRenderFunction = ( { elementRefList } ) => {
    return <div ref={elementRefList.get('splitter')}></div>;
}
```

- Use the `domRef` keyword for components:

```
export const awSplitterRenderFunction = ( { elementRefList } ) => {
    return <AwButton domRef={elementRefList.get('splitter')}></
AwButton>;
}
```

3. Consume the reference object in some JavaScript code as needed. Make sure to call the `.current` property as shown in the following example to get the DOM element currently associated with the virtual DOM element in the previous step. Once you have a real DOM element, you can use all the normal imperative DOM manipulation APIs on that element.

```
export const initSplitter = ( elementRefList ) => {
  const splitter = elementRefList.get('splitter').current;
  let area1 = splitter.previousElementSibling;
  let area2 = splitter.nextElementSibling;
}
```

# Responsiveness

## Overview

Responsiveness is a key characteristic of the Siemens Web Framework that suggests that application should respond to the user's behavior and environment based on screen size, platform and orientation.

This practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from large screen devices to smaller screens, the Framework automatically switches the application layout to support the screen size.

## Setting The Viewport

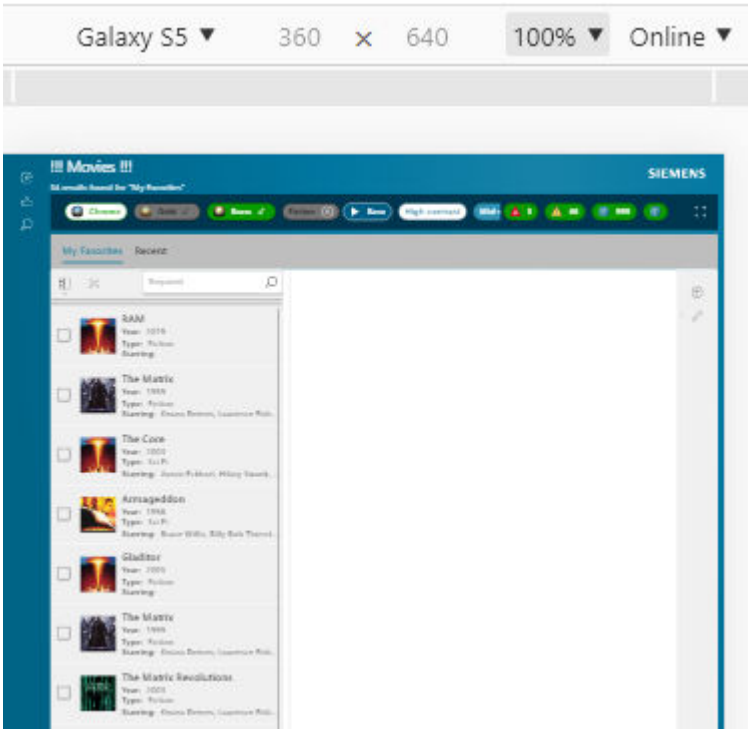
To get responsive behavior out of the box, we add the viewport meta tag.

The viewport is the user's visible area of a page. It varies with the device - it will be smaller on a mobile phone than on a computer screen.

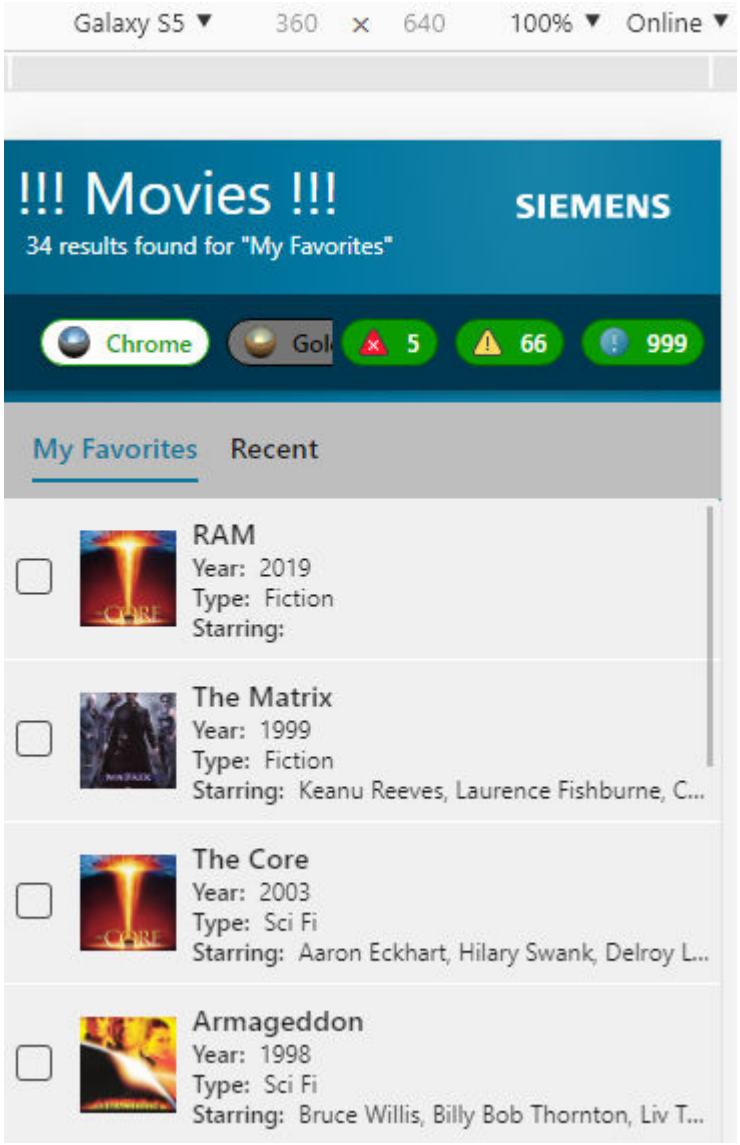
```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Without viewport meta tag



## With viewport meta tag

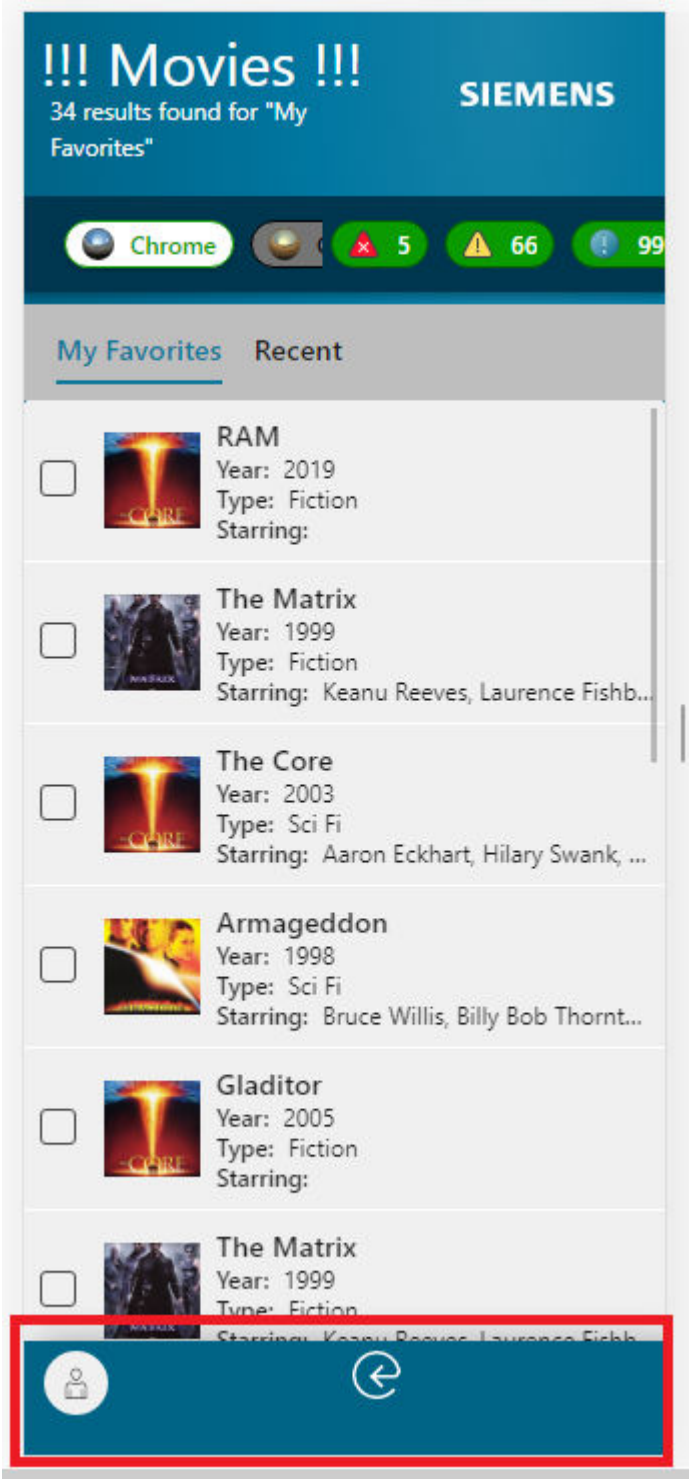



## Navigation bar support

Siemens Web Framework supports mobile devices out of the box. The left hand navigation bar is shown in the footer where commands need to be contributed explicitly for them to show up.This works only when location and sublocation pattern is used.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Footer commands




## Adding Commands in Footer

The commands added to the footer are like any other commands in Siemens Web Framework. The commands to be shown in narrow mode footer bar need to be placed against the anchor `aw_footer`,


Siemens Web Framework
Unpublished work. © 2025 Siemens

for e.g. to place the back command in the footer, we need to add it to `commandPlacements` within `commandsViewModel.json` as shown below:

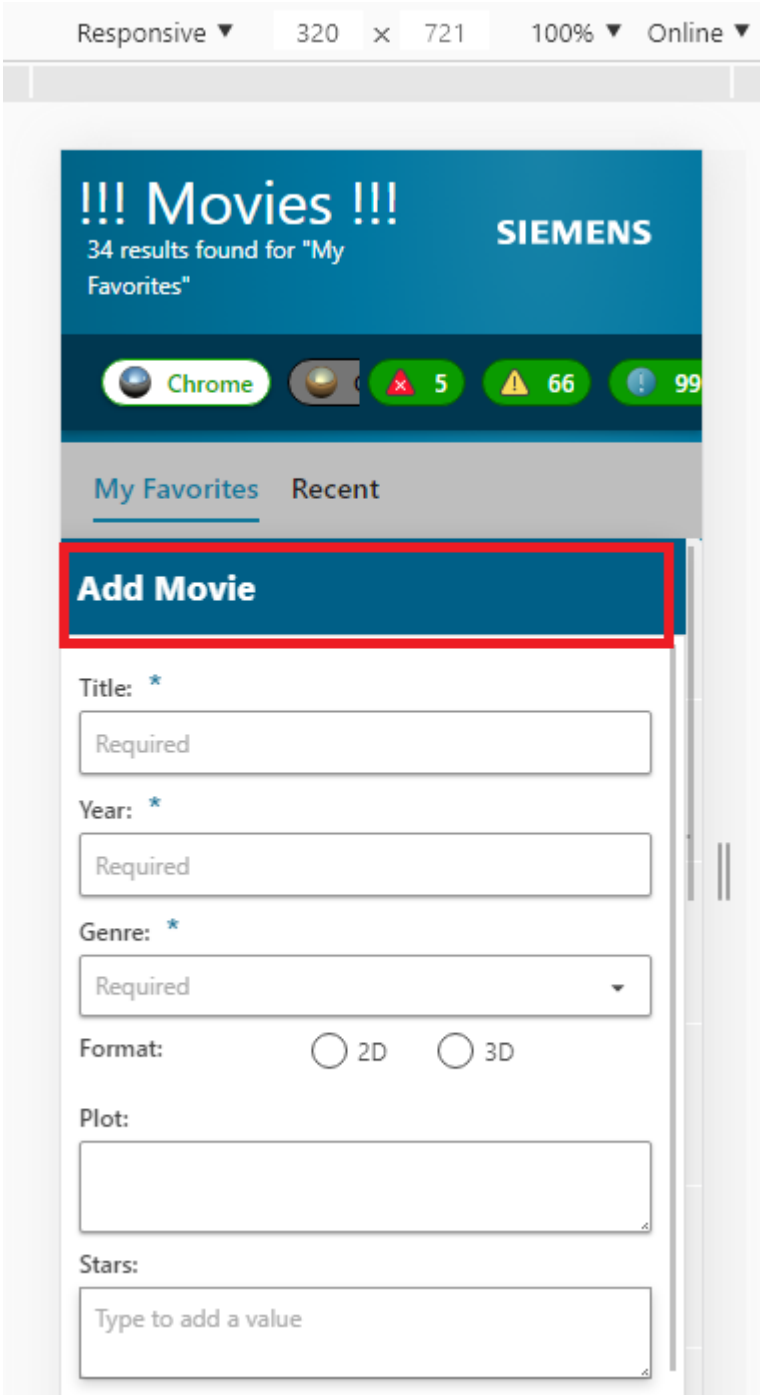
```
{
  "commandPlacements": {
    "Awp0GoBackFooterToolbar": {
      "id": "Awp0GoBack",
      "uiAnchor": "aw_footer",
      "relativeTo": "Awp0GoHome",
      "priority": -1
    }
  }
}
```

Rest of command definition remains same as any other command in `commandsViewModel.json`

## Command panel in mobile layout

All the commands that can open command panels should be contributed to the footer in the narrow mode layout.

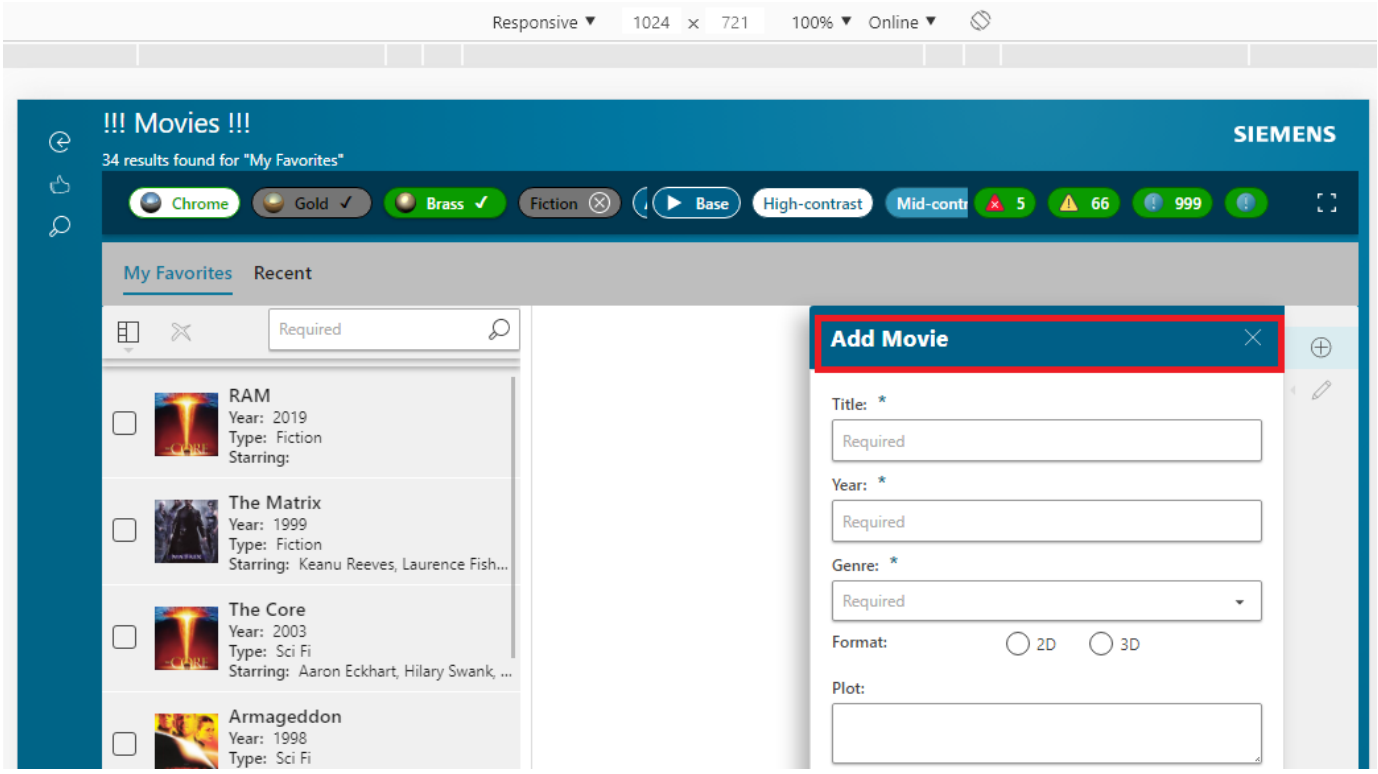
The behavior of command panel changes in mobile layout. It doesn't have a close button. UX team doesn't recommend showing close button in mobile layout.





Siemens Web Framework
Unpublished work. © 2025 Siemens

## Command Panel in desktop layout




## Responsive Designs - Grid View

Siemens Web Framework uses 12 column grid system to render layout, that means the total available width is divided into 12 columns. The grid would always retain 12 columns irrespective of device sizes, that means same column on the extra large devices would have more available width compared to that on extra small device.

A responsive website would typically show less columns on smaller devices in order to better utilize the available width. The showing less columns can be achieved in two ways, either drop columns (wrap) and support page scroll, so user can still entire page contents, or hide certain columns altogether. Siemens Web Framework supports dropping column (wrap).

In Siemens Web Framework, responsive Layouts constructed with the combination of `aw-row` and `aw-column`.

To know more about layout elements please refer **Layout Wiki**, and to learn more about the responsive designs, please refer `Responsive Designs` section from above wiki.

## Example in Storybook

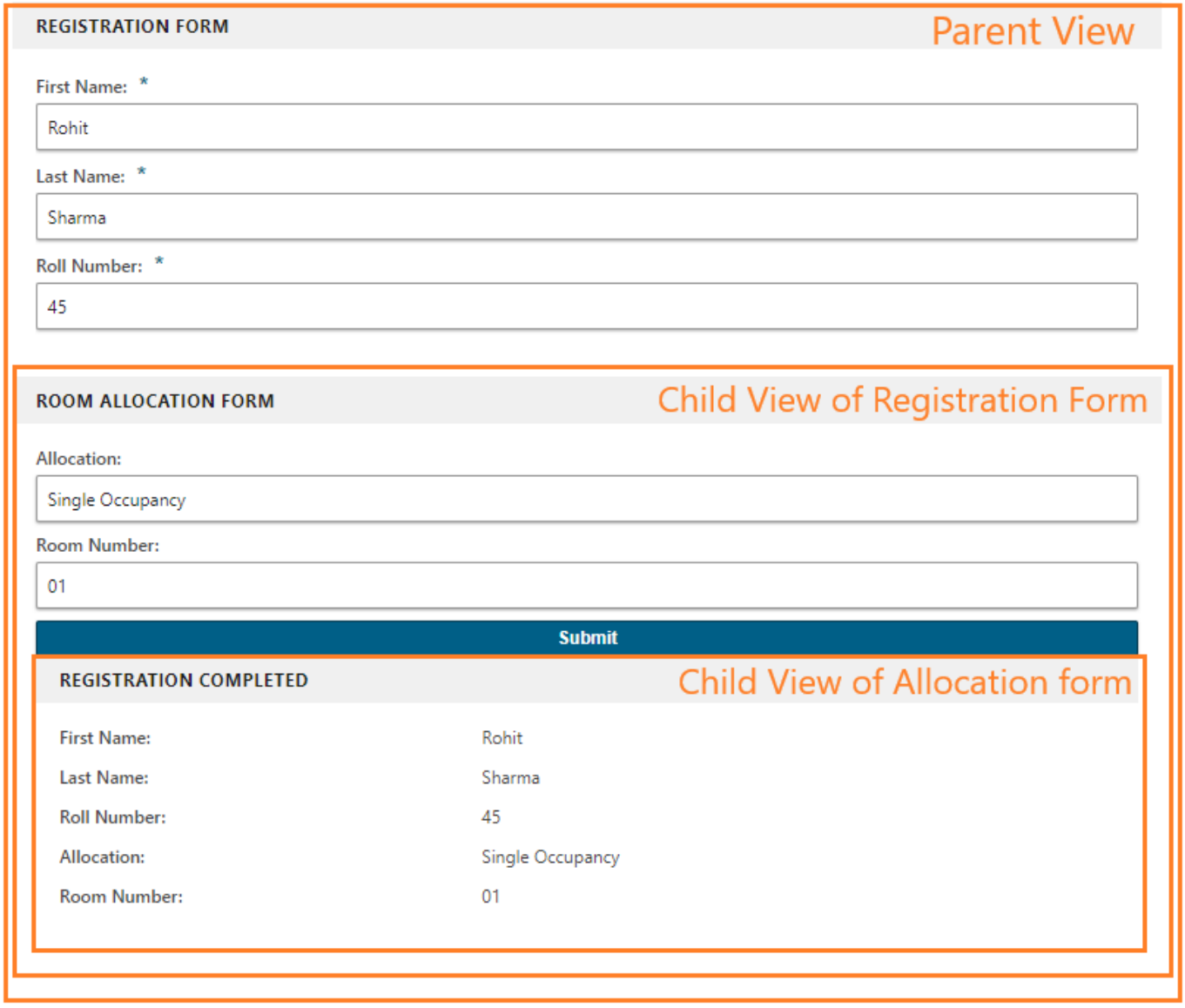
Please refer to `aw-column` example in **Siemens Web Framework Storybook** and see `Column with responsive grid layout`.

# Reusability

Reusability is a key characteristic of any programming framework, and is no different for Siemens Web Framework. The declarative artifacts like view and view model can be reused to a certain extent.

## Reusability through aw-include

`aw-include` is an element which does not have any UI representation of itself. However, it can be extensively used to embed one view into another view.



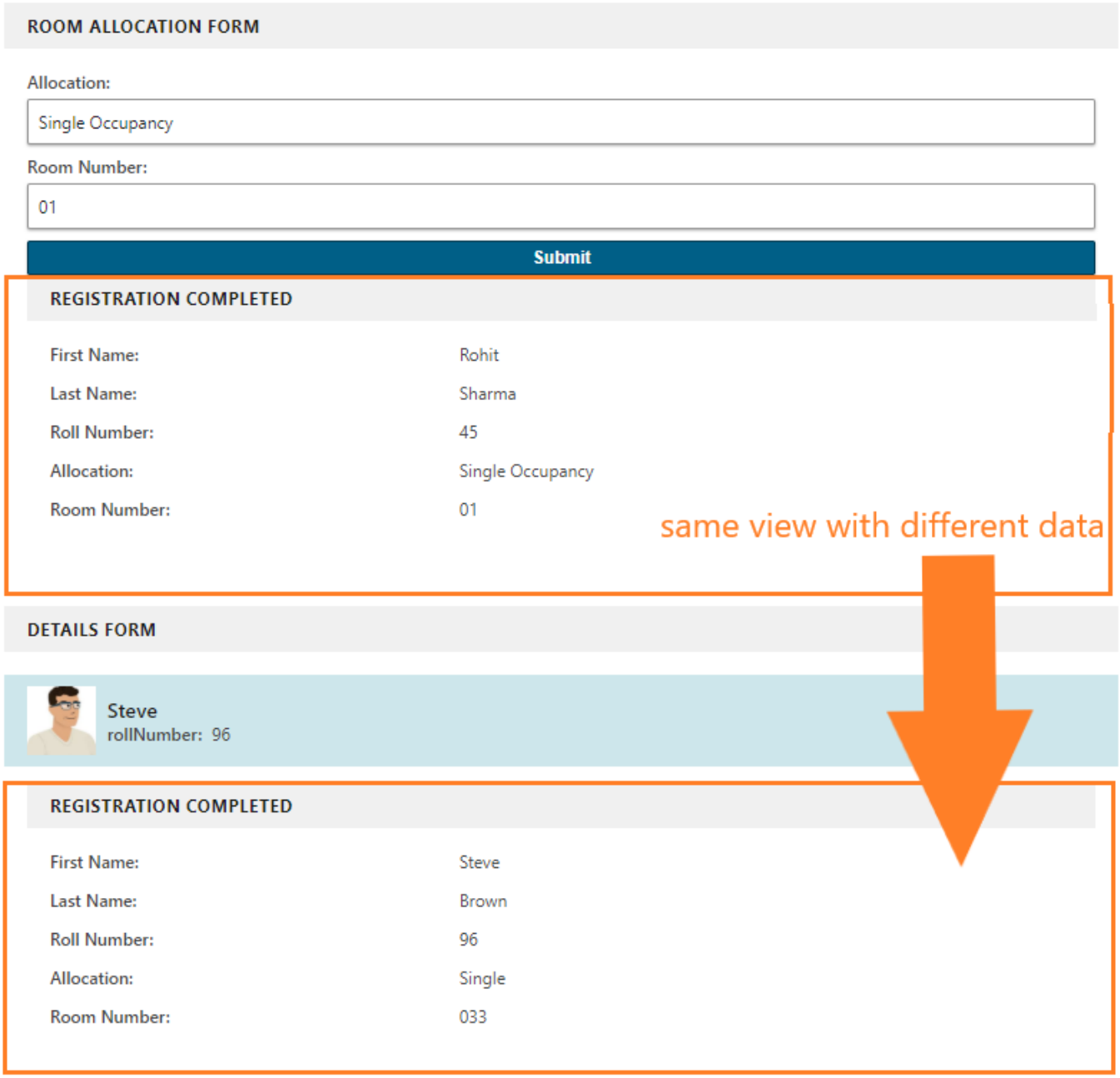


Siemens Web Framework
Unpublished work. © 2025 Siemens

For example:

```
<aw-include name="RoomAllocationForm"></aw-include>
```

To explain programmatically, when aw-include is triggered, it loads the child view and view model onto memory. It has its own declarative view model. The parent view model can communicate with the child view model through **sub-panel-context**.





Unpublished work. © 2025 Siemens

Here in this example we have passed different subPanelContext objects from parentView to childView through aw-include and referred it in viewmodel json.When we click on button "Submit" data gets shared with the form through event and subPanelContext as a input data. A View appears which displays the shared data. Similarly, In details form, on the selection of list item, a same view appears with list item's data.

The `aw-include` component helps in inclusion of a declarative view fragment from another file. The included contents are rendered as child nodes of the specified element. It helps in better code separation.

Please refer to the Example of `aw-include` in **Siemens Web Framework Storybook**


Siemens Web Framework
Unpublished work. © 2025 Siemens

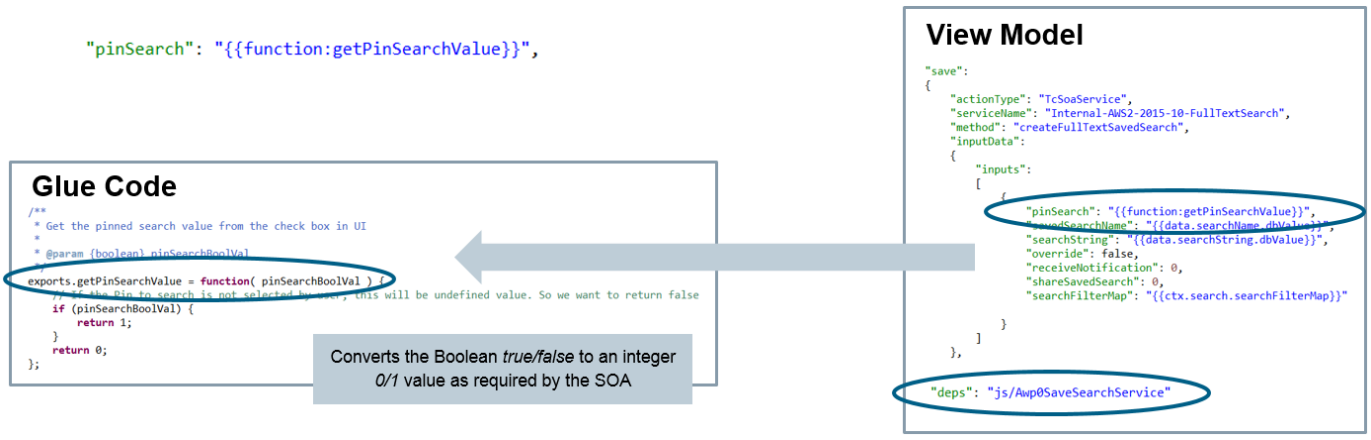
# 7. Configuration and Extensibility


- Bootstrap
- Configuration
  - Authentication
  - Clipboard
  - Solution
  - Header
  - Adapters
  - Extended Tooltip
- Codeful extensions
- Internationalization (i18n)/Localization (i10n)


## Codeful extensions

The individual UI elements' internal implementation is codefull. Best practice in using Siemens Web Framework is for codeless usage of elements in the declarative **View** and **View Models**. In this way the amount of code you have to maintain, test, download to the browser, and manage as a touchpoint to given web frameworks is signficantly reduced. Codeless configuration also allows for zero down time automated Cloud deployment workflows.

However, when required codefull extensions (aka "glue code") can be used in your declarative **View Models**. You can call a JavaScript function using the following `{{function:nameOfJsFunction}}` where *nameOfJsFunction* is a function in the service that will be demand loaded that is specified in the *"deps"* (e.g. `"deps": "js/MyServiceName"` ).



### JavaScript actions

**Actions** can have an *actionType* of *"JSFunction"*. As with any action, the *inputData* provides the JSON inputs and *outputData* can be used to map the response. Again, the *"deps"* field is used for providing the service to demand load which has the JavaScript *"method"*.

```
{
  "convertToList": {
    "actionType": "JSFunction",
    "method": "createListModelObjects",
    "inputData": {
      "input": "{{data.templatesInput}}",
      "propName": "props.template_name",
      "addEmpty1stItem": true
    },
    "outputData": {
      "templates": ""
    },
    "deps": "js/listBoxService"
  }
}
```

## Context configuration

### How to define context

To define a new contribution to the header , you need to do the following:

1. Add `contextConfiguration.json` as peer to your `module.json`.

```
{
  "contexts": {
    "projectContext": {
      "view": "projectContext"
    },
    "locationContext": {
      "view": "locationContext"
    }
  },
  "placements": {
    "projectPlacement": {
      "anchor": "headerContextPosition",
      "contextId": "projectContext",
      "priority": 20
    },
    "locationPlacement": {
      "anchor": "headerContextPosition",
      "contextId": "locationContext",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "priority": 50
        }
      }
    }
```

Where:

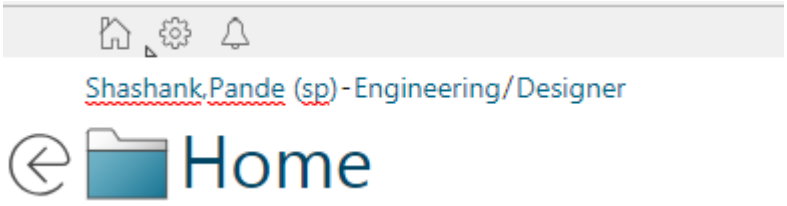
- `View` : View Id for the contribution
- `anchor` : the placement where contributed links need to show
- `priority` : the sequence in which the links should be displayed.
- To override some contribution for some pages, use the *activeWhen* and *visibleWhen* in placements.

2. Override json placement defination

```
    {
      "placements": {
        "projectPlacement": {
          "anchor": "headerContextPosition",
          "contextId": "projectContext",
          "priority": 20,
          "activeWhen": {
            "condition":
"conditions.cmdShowObjectDevConfigCommandActive"
          },
          "visibleWhen": {
            "condition":
"conditions.cmdShowObjectDevConfigCommandActive"
          }
        }
      }
    }
```

## How to configure context control at Workspace

It's possible to control what contexts are shown in the header area. This configuration is driven via **workspace** definition.




Workspace definition

```
{
    "schemaVersion": "1.0.0",
```

```
    "workspaceId": "TcAdminWorkspace",
    "workspaceName": {
        "source": "/i18n/WorkspaceMessages",
        "key": "tcAdminWorkspaceName"
    },
    "workspaceType": "Exclusive",
    "defaultPage": "showGateway",
    "availableContextConfigurations": [
        "userContext",
        "groupRoleContext"
    ],
}
```

### availableContextConfigurations

It's an `Optional` param and it's an array of strings where each value is the contextId for each of link which would show-up in context control while working in a specific workspace. Each value is the state defined in `contextConfiguration.json`. If availableContextConfigurations is not defined, then all of the contextIds/contextConfiguration available in the kit would show up.

### Here are all the contextIds available OOTB

- userContext
- groupRoleContext
- workSpaceContext
- projectContext
- locationContext
- revisionContext
- changeContext

## Clipboard service

To define a clipboard, you need to do the following:

```
/**
 * Add clipboard implementation.<code JavaScript>/**
 * @module js/myClipboardImpl
 */
var exports = {};

/**
 * Return an array of Objects currently on the clipboard.
 *
 * @return {Array} Current contents of the clipboard.
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
 */
export let getContents = function() {
    return [];
};

/**
 * Sets the current contents of the clipboard.
 *
 * @param {Object[]} contentsToSet - Array of Objects to set as the current
clipboard
 *            contents.
 */
export let setContents = function( contentsToSet ) {};

/**
 * Return the content of the clipboard that is cached.
 *
 * @return {Object[]} Array of current Objects that is cached.
 */
export let getCachableObjects = function() {
    return [];
};

/**
 * Copies the URL for the given object to OS clipboard
 *
 * @param {Object} selObject - selected object
 * @return {Boolean} verdict whether the content was successfully copied to
the clipboard or not
 */
export let copyUrlToClipboard = function( selObject ) {
    return false;
};

/**
 * Copies hyperlink to OS clipboard
 *
 * @param {Object[]} content - array of selected object whose hyperlink is
created and copied to os
 *            clipboard
 * @return {Boolean} successful whether the content was successfully copied
to the clipboard or not
 */
export let copyHyperlinkToClipboard = function( content ) {
    return false;
};

export default exports = {
    getContents,
    setContents,
    getCachableObjects,
    copyUrlToClipboard,
    copyHyperlinkToClipboard
}
```

Then, add `clipboard.json` as peer to your `module.json`.

```
{
    "myClipboardName": {
        "dep": "js/myClipboardImpl"
    }
}
```

- Indicate in your solution definition `kit.json` to use this clipboard.

```
{
    "solutionDef": {
        "clipboard": "myClipboardName"
    }
}
```

## Bootstrap

If your software (third-party or not) needs to be added to the app bootstrap of the client, you should not need to edit the `bootstrap.js`.

To support this, you can create a `bootstrap.json` file (adjacent to your `module.json`). Here's the schema of the file:

```
{
    "config": {
        "paths": {
            "moduleName": "path relative to assets directory within war"
        },
        "shim": {
            "moduleName": {
                "deps": [
                    "moduleName of another module which your module
requires to be loaded first"
                ],
                "exports": "exports"
            }
        }
    },
    "import": [
        "module to import"
    ],
    "initModule": [
        "myAngularModuleName"
    ],
    "postImport": [
        "self executing JavaScript"
    ],
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "postInit": [
        "import( 'js/myService' ).then( function()
{ app.getInjector().get( 'myServiceNGID' ).myInitFunction(); } );",
    ],
    "minOnly": [
        "base filename for JavaScript files without non-minified versions"
    ]
}
```

- The `config` field is merged and passed to the build config.
- The `import` field is merged into the top-level module imports to load other modules before the bootstrap.
- The `postImport` field should be used sparingly to handle third-party corner cases.
- The `initModule` field is merged and passed to `ngModule.module` 2nd parameter of dependent modules.
- The `postInit` field should be used sparingly to handle third-party corner cases. These are code snippets that are executed after Angular is initialized.
- The `minOnly` field is used during the build with environment variable `DRAFT` non set or set to "true". This will cause the bootstrap to always load the minified version of the JavaScript. If not provided (which is preferred), the build will reference the non-minified version of your module to support developers debugging.

## Notes

```
From v3.3.0 onwards, Siemens Web Framework uses webpack as a default
bundler.
```

- With webpack, `config.shim` is not supported. Instead, the user provides the configuration.

```
{
    "config": {
        "paths": {
            "jquery": "lib/jquery/jquery.min"
        },
        "provides": [
            {"window.jQuery": "jquery"},
            {"jQuery": "jquery"}
        ]
    }
}
```

```
From v4.0.0 onwards, requirejs and postRequire properties have been
replaced with import and postImport.
```

## Learn more...

- **Webpack**

## Examples

## CKEditor

- src\repo\com.ckeditor\bootstrap.json

```
{
    "config": {
        "paths": {
            "ckeditor": "lib/ckeditor/ckeditor"
        }
    }
}
```

## ACE

- src\repo\ace\bootstrap.json

```
{
    "config": {
        "paths": {
            "ace": "lib/ace-1.2.0"
        }
    },
    "initModule": [
        "ui.ace"
    ]
}
```

## DHTMLXGantt

- src\repo\dhtmlxgantt\bootstrap.json

```
{
    "config": {
        "paths": {
            "dhtmlxgantt": "lib/dhtmlxgantt/dhtmlxgantt.min",
            "dhtmlxgantt_smart_rendering": "lib/dhtmlxgantt/ext/
dhtmlxgantt_smart_rendering.min",
            "dhtmlxgantt_multiselect": "lib/dhtmlxgantt/ext/
dhtmlxgantt_multiselect.min",
            "dhtmlxgantt_marker": "lib/dhtmlxgantt/ext/
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
dhtmlxgantt_marker.min",
            "dhtmlxgantt_tooltip": "lib/dhtmlxgantt/ext/
dhtmlxgantt_tooltip.min"
        },
        "shim": {
            "dhtmlxgantt": {
                "deps": [
                    "angular"
                ],
                "exports": "gantt"
            },
            "dhtmlxgantt_smart_rendering": {
                "deps": [
                    "dhtmlxgantt"
                ],
                "exports": "gantt"
            },
            "dhtmlxgantt_multiselect": {
                "deps": [
                    "dhtmlxgantt"
                ],
                "exports": "gantt"
            },
            "dhtmlxgantt_marker": {
                "deps": [
                    "dhtmlxgantt"
                ],
                "exports": "gantt"
            },
            "dhtmlxgantt_tooltip": {
                "deps": [
                    "dhtmlxgantt"
                ],
                "exports": "gantt"
            }
        }
    }
}
```

# Authentication

This contains information about authenticator and post authentication pipeline.

## Authenticator

To define a client authenticator, you need to do the following:

```
/**

* Add authenticator JavaScript file which exposes one method, `getAuthentica
```

```
tor` which returns a function that provides the authenticator.<code JavaScri
pt>/**
 * @module js/myAuthenticatorService
 */
import AwPromiseService from 'js/awPromiseService';

let exportsObj = {};
export let getAuthenticator = function() {
    var exports = {};

    /**

* This function determines if authentication is required. If already authent
icated, none is required.
     *

* @return {Promise} if promise is resolved, then no authentication is requir
ed. if promise fails, then the authenticate function will be called.
     */
    exports.checkIfSessionAuthenticated = function() {
        return AwPromiseService.instance.resolve();
    };

    /**

* authenticator specific function to carry out authentication. In the intera
ctive case, we just resolve
     * directly to continue the pipeline.
     */
    exports.authenticate = function() {
        return AwPromiseService.instance.resolve();
    };

    /**

* this is called during the authentication process. It gets invoked after th
e authentication is
     * completed/ready. It is a spot to do any session level initialization.
     *

* @return {Promise} promise to be resolved after the authenticator does self
 initialization
     */
    exports.postAuthInitialization = function() {
        return AwPromiseService.instance.resolve();
    };

    /**

* triggers the authenticator sign out logic. Returns a promise invoked upon
completion
     *

* @return {Promise} promise to be invoked upon completion of the signout tas
ks
     */
```


Unpublished work. © 2025 Siemens

```
    exports.signOut = function() {
        return AwPromiseService.instance.resolve();
    };

    return exports;
};
export let moduleServiceNameToInject = 'myAuthenticatorService';
export default exportsObj = {
    getAuthenticator,
    moduleServiceNameToInject
};
```

- Add `authenticator.json` as peer to your `module.json`.

```
{
    "myAuthenticatorName": {
        "dep": "js/myAuthenticatorService"
    }
}
```

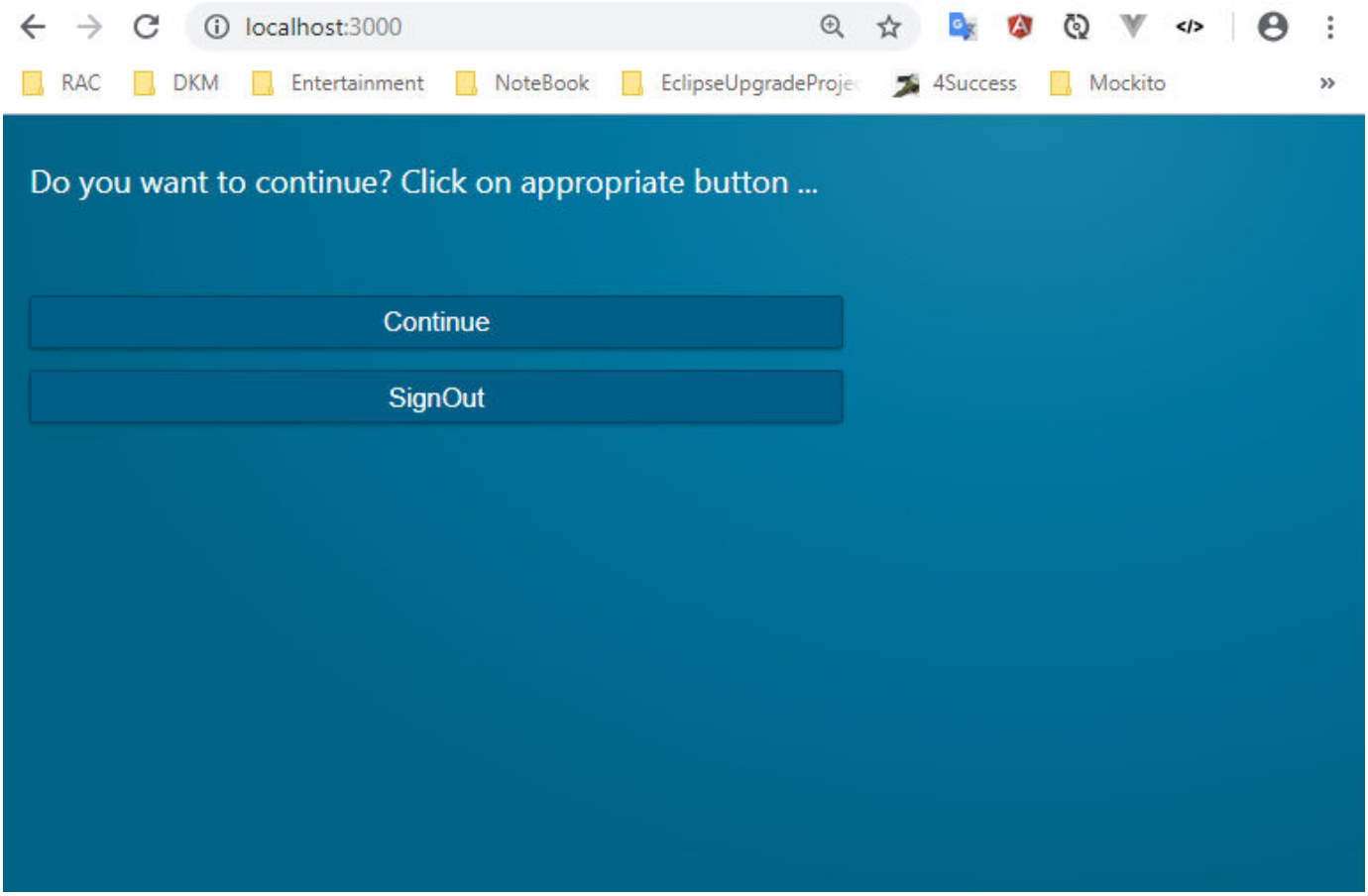
- Indicate in your solution definition `kit.json` to use this authenticator.

```
{
    "solutionDef": {
        "authenticator": "myAuthenticatorName"
    }
}
```

## Post Authentication Pipeline

This is the post login pipeline execution stage. The session manager blocks on any post logic processes. This is a configuration point to execute any contributed post authentication logic. The "postLoginPipeline" named contributions are used to obtain pipeline "step" definitions which identify the logic or route to be run.

**Example-** After login/authentication successful route to the page where we can prompt to the user about want to continue or signout.




To define the post login pipeline for above requirement do the following steps.

- Create `contributions.json` file in your module parallel to `module.json` with following contents.

```
{
    "postLoginPipeline": [ "js/sampleLoginPipelineContribution"],
    "states": [ "js/confirmLoginjs.states"]
}
```

Here we are contributing "postLoginPipeline" through `sampleLoginPipelineContribution.js` file and "state" through `confirmLoginjs.states.js` file

- Create `sampleLoginPipelineContribution.js` file.

Contributed step definition should return the promise and resolve with following object. either routeName or workFunction should be defined not both.

```
 {
       name: 'ConfirmLogin',        //Name of the pipeline step definition
       active: true,                //Status of the pipeline step definition
       routeName: 'confirmLogin',   // route name where you want to navigate
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
       workFunction: null          // promise where you can define resolve/
reject logic based on some logic.
 }
```

```
/**
 * This is a contribution for a login process blocking step definition
 *
 * @module js/sampleLoginPipelineContribution
 */
const contribution = {
    getPipelineStepDefinition: function() {
        var confirmLoginStepDefn = {
            name: 'ConfirmLogin',
            active: true,
            routeName: 'confirmLogin', // either routeName or workFunction,
not both
            workFunction: null
        };
        return confirmLoginStepDefn;
    }
};

export default function( key, deferred ) {
    if( key === "postLoginPipeline" ) {
        deferred.resolve( contribution );
    } else {
        deferred.resolve();
    }
};
```

- In authenticator after authentication complete create the stage object (either get it from server or define manually) and store it into local storage. import the localstorage service and modify the `postAuthInitialization` method from `authenticator.js` (see above section for how to define the authenticator). stage object should contains following properties.

```
{
    stage.name = "confirmLogin";     //Name of the stage, same as defined
in sampleLoginPipelineContribution.js file
    stage.status = false;            //status of the stage
    stage.priority = 0;              //priority of the stage, if there are
multiple stage then sets the priority according to that
}
```

```
import AwPromiseService from 'js/awPromiseService';
import localStrg from 'js/localStorage';

let exportObj = {};

export let getAuthenticator = function() {
    var exports = {};
```

```
    /**
     * this is called during the authentication process. It gets invoked
after the authentication is
     * completed/ready. It is a spot to do any session level initialization
and post login pipeline stages initialization.
     *
     * @return {Promise} promise to be resolved after the authenticator
does self initialization
     */
    exports.postAuthInitialization = function() {

        var postLoginStageKey = "postLoginStagesKey";
        var postLoginStages = localStrg.get( postLoginStageKey );

        if( !postLoginStages ) {
            postLoginStages = [];
            var stage = {};
            stage.name = "confirmLogin";
            stage.status = false;
            stage.priority = 0;
            postLoginStages.push( stage );
            localStrg.publish( postLoginStageKey,
JSON.stringify( postLoginStages ) );
        }

        return AwPromiseService.instance.resolve();
    };

    return exports;
};
export let moduleServiceNameToInject = 'myAuthenticatorService';
export default exportObj = {
    getAuthenticator,
    moduleServiceNameToInject
};
```

- Now it's time to add the states. Create `confirmLoginjs.states.js` file with following code.

```
/**
 * This is a contribution for a login process blocking step definition
 *
 * @module js/confirmLoginjs.states
 */
const contribution = {
    "confirmLogin": {
        templateUrl: '/html/confirmlogin.html',
        controller: 'ConfirmLogin',
        resolve: {
            loadController: [ '$q', '$injector', function( $q ) {
                var deferred = $q.defer();
                // Note: Starting with Siemens Web Framework 4.0.0, dynamic
requirejs statements should be replaced with import.
                import( 'js/
confirm.login.controller' ).then( deferred.resolve );
```


Unpublished work. © 2025 Siemens

```
                return deferred.promise;
            } ]
        }
    }
};
export default function( key, deferred ) {
    if( key === 'states' ) {
        if( deferred ) {
            deferred.resolve( contribution );
        } else {
            return contribution;
        }
    }
};
```

- Create `confirmlogin.html` for confirmLogin state.

```
<form class="aw-layout-mainView afx-dark-background" name="myForm">
    <div>
        <div class="aw-layout-panelBody">
            <div class="aw-layout-panelSection">
                <div class="aw-widgets-propertyLabel aw-base-scrollPanel">
                    Do you want to continue? Click on appropriate button ...
                </div>
            </div>
        </div>
        <div class="aw-layout-panelFooter" style="width: 400px;">
            <button type="button" class="aw-base-blk-button" aw-
click="continueOn()">Continue</button>
            <button type="button" class="aw-base-blk-button" aw-
click="signOut()">SignOut</button>
        </div>
    </div>
</form>
```

- Create `confirm.login.controller.js` for confirmLogin state.

```
// NEEDS TO BE UPDATED FOR REACT
/**
 * @module js/confirm.login.controller
 * @class module
 * @memberOf module
 */
import logger from 'js/logger';

app.controller( 'ConfirmLogin', [
    '$scope',
    '$state',
    'sessionManagerService',
    function( $scope, $state, sessionMgrSvc ) {

        /**
```

```
         * member to hold the continuation promise in the execution flow.
         */
        var nextContinuation;

        // assertion is that the session manager populated this data member
with
        // the continuation.
        if( $state.current.data && $state.current.data.nextContinuation ) {
            nextContinuation = $state.current.data.nextContinuation;
        } else {
            logger.info( "ERROR - no continuation available for the
pipleline controller" );
        }

        /**
         * function binding for the continue on action
         */
        $scope.continueOn = function() {
            nextContinuation.resolve();
        };

        /**
         * function binding for the sign out action
         */
        $scope.signOut = function() {
            // create a failure object with reason code?  What to pass
along for exit reason?
            nextContinuation.reject( {
                'cancel': true
            } );
            sessionMgrSvc.terminateSession();
        };
    }
] );
```

## FAQ

**1. What is the difference between post authentication pipeline and postAuthInitialization hook?**

**Answer:** postAuthInitialization method of `authenticator.js` gets invoked after the authentication is completed or ready. in this method you can write the code to fetch the data from server and makes it available to application.

post authentication pipeline contribution provides you to navigate to route and invoke promise workFunction functionality. This will also gives you the guarantee to execute it after the authentication complete but the major difference is pipeline stage data get's stored in local storage so it will not query for the data every login (if you getting the stage data from server.) until clear the cache.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Solution configuration

## Solution configuration for paste handling

The paste handling can be configured at two levels.

- **Default paste handler** – This can be configured at Solution level i.e. in kit.json
- **Handlers that are Specific to target + source type combination** - This can be configured at module level i.e. in /paste.json

This configuration is optional. However, the Paste won't work till this is configured.

## How to Configure Handlers that are specific to target + source type combination

`paste.json` defined for the module inside the solution holds this configuration.

This is the paste.json configuration (until Siemens Web Framework 3.2)

```
{
    "targetTypes":
    {
        "ChangeImpactedItem":
        {
            "sourceTypes":
            {
                "WorkspaceObject":
                {
                    "handler": "defaultPasteHandlerForImpactAnalysis",
                    "deps": "js/Cm1ImpactedWidgetService"
                }
            }
        }
    }
}
```

This structure depicts that against one target object type there can be multiple source object types and each source object type can be associated with one handler. For example in above schema target object type is `ChangeImpactedItem` and source object type is `WorkspceObject` which has been defined with the respective handler configuration.

So when drag and drop operation is performed between this specified target and source object, this handler will get invoked.

Now, starting SWF 3.3, this configuration point is being enhanced to meet two requirements:

1. To allow the applications to define more than one handlers (depending upon some context) for the same source and target combination
2. To allow the applications to manage their custom success and failure messages

This is the new schema for paste.json:

```
{
    "pasteHandlers": {
        "defaultPasteHandlerForImpactAnalysis": {
            "action": "defaultPasteHandlerForImpactAnalysis",
            "activeWhen": {
                "condition":
"conditions.isPasteHandlerActiveForImpactAnalysis"
            }
        }
    },
    "actions": {
        "defaultPasteHandlerForImpactAnalysis": {
            "actionType": "JSFunction",
            "method": "defaultPasteHandlerForImpactAnalysis",
            "deps": "js/Cm1ImpactedWidgetService",
            "inputData": {
                "targetObject": "{{pasteContext.targetObject}}",
                "sourceObject": "{{pasteContext.sourceObject}}"
            },
            "events": {
                "success": [ {
                    "name": "dragDrop.success",
                    "condition": "pasteContext.isDragDropIntent",
                    "eventData": {
                        "sourceObjects": "{{pasteContext.sourceObject}}",
                        "targetObject": "{{pasteContext.targetObject}}"
                    }
                } ]
            }
        }
    },
    "conditions": {
        "isPasteHandlerActiveForImpactAnalysis": {
            "expression": {
                "$and": [ {
                        "$source": "pasteContext.targetObject",
                        "$query": {
                            "$typeOf": "ChangeImpactedItem"
                        }
                    },
                    {
                        "$source": "pasteContext.sourceObject",
                        "$query": {
                            "$typeOf": "WorkspaceObject"
                        }
                    }
                ]
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            }
        }
    }
}
```

Above schema depicts a `pasteHandler` which has its action defined declarative. It also has an optional active-when clause to associate a condition with it. Similar to commands, there can be more than one pasteHandlers defined now even for same source and target object types and when the condition corresponding to any pasteHanlder is evaluated to true, the action associated with that handler is executed.

For the given example, if the target object is `ChangeImpactedItem` and source object is `WorkspaceObject` the `defaultPasteHandlerForImpactAnalysis` will get activated and the handler `defaultPasteHandlerForImpactAnalysis`will get invoked.

**Example with multiple paste Handlers defined for same source and target combination:**

```
{
    "pasteHandlers": {
        "defaultPasteHandlerForImpactAnalysis": {
            "action": "defaultPasteHandlerForImpactAnalysis",
            "activeWhen": {
                "condition":
"conditions.isPasteHandlerActiveForImpactAnalysis"
            }
        },
    "defaultPasteHandlerForImpactAnalysisNew": {
            "action": "defaultPasteHandlerForImpactAnalysis",
            "activeWhen": {
                "condition":
"conditions.isPasteHandlerActiveForImpactAnalysisOther"
            }
        }
    },
    "actions": {
        "defaultPasteHandlerForImpactAnalysis": {
            "actionType": "JSFunction",
            "method": "defaultPasteHandlerForImpactAnalysis",
            "deps": "js/Cm1ImpactedWidgetService",
            "inputData": {
                "targetObject": "{{pasteContext.targetObject}}",
                "sourceObject": "{{pasteContext.sourceObject}}"
            },
            "events": {
                "success": [ {
                    "name": "dragDrop.success",
                    "condition": "pasteContext.isDragDropIntent",
                    "eventData": {
                        "sourceObjects": "{{pasteContext.sourceObject}}",
                        "targetObject": "{{pasteContext.targetObject}}"
                    }
                } ]
```

```
            }
        }
    },
    "conditions": {
        "isPasteHandlerActiveForImpactAnalysis": {
            "expression": {
                "$and": [ {
                        "$source": "pasteContext.targetObject",
                        "$query": {
                            "$typeOf": "ChangeImpactedItem"
                        }
                    },
                    {
                        "$source": "pasteContext.sourceObject",
                        "$query": {
                            "$typeOf": "WorkspaceObject"
                        }
                    },
                    {
                      ctx.selected.length > 1
                    }

                ]
            }
        },
        "isPasteHandlerActiveForImpactAnalysisOther": {
            "expression": {
                "$and": [ {
                        "$source": "pasteContext.targetObject",
                        "$query": {
                            "$typeOf": "ChangeImpactedItem"
                        }
                    },
                    {
                        "$source": "pasteContext.sourceObject",
                        "$query": {
                            "$typeOf": "WorkspaceObject"
                        }
                    },
                    {
                      ctx.selected.length === 1
                    }

                ]
            }
        }

    }
}
```

Now, since these pasteHandlers and their actions are declaratively defined in paste.json; applications are free to specify their custom messages for success and failure conditions. By default, two events 'dragDrop.success' and 'dragDrop.failure' have been provided which can be leveraged if the application does not want any custom handling of success and failure message. In such a case, the default messages


Unpublished work. © 2025 Siemens

provided by SWF will be shown - same behavior as what is available in SWF 3.2. Like any other events, these events can be fired conditionally and 'pasteContext.isDragDropIntent' is used to support the use cases where action messages for paste command (invoking pasteService.execute) were handled separately and generic messages were required by application teams only for drag and drop operations.

Best Practices :

1. While conditions can be nested, please note that there is a cost associated with the evaluation of these conditions. So recommendation is to keep them simple.
2. The supported format for conditions expressions is object based conditions i.e. conditions using operators like $and$, or etc.
3. In active when expression, validation of `pasteContext.targetObject` should be first then other validation should be carried out.

## How To Configure Default Paste Handler for Solution

- `kit.json` defined for the solution holds this configuration in `defaultPasteHandlerConfiguration`.
- All the fields mentioned in the configuration format are mandatory.

```
{
  "defaultPasteHandlerConfiguration": {
    "bestTargetFitFinder": "bestTargetFit",
    "bestSourceFitFinder": "bestSourceFit",
    "pasteHandler": "tcDefaultPasteHandler",
    "deps": "js/tcDefaultPasteHandler"
  }
}
```

- You need to configure below in the `defaultPasteHandlerConfiguration` -
  - `bestTargetFitFinder:` This function finds the best possible Target Fit for the given target object by reading the paste configuration.
  - `bestSourceFitFinder:` This function finds the best possible Source Fit for the given source object by reading the paste configuration.
  - `pasteHandler:` If no suitable target + source type combination for paste handler is found then this default paste handler function is invoked.
  - `deps:` The dependent file which holds above functions.

## Solution configuration for obtaining images

As a consumer of Afx ('Apollo'), application engineer can configure how and where images are obtain from. We can configure 'imageRepositoryConfiguration' in solution definition (kit.json).

## Structure

```
{
  "imageRepositoryConfiguration": {
    "actionType": "GET",
    "url": "<url>"
  }
}
```

## Example Configurations

- Default configuration, if images are in AW war the configuration will be like:

```
{
    "imageRepositoryConfiguration": {
        "actionType": "GET",
        "url": "{{baseUrl}}"
    }
}
```

- If images are hosted on a web server say **http://imageserver:8887** the configuration will look like:

```
{
    "imageRepositoryConfiguration": {
        "actionType": "GET",
        "url": "http://imageserver:8887”
    }
}
```

## Solution configuration for configuration/preference

As a consumer of Afx (‘Apollo’), application engineer can configure how to provide configuration/ preference service if required. We can configure ‘configurationPreferencesConfiguration’ in solution definition (kit.json).

## Structure

```
{
    "imageRepositoryConfiguration": {
        "actionType": "GET",
        "url": "<url>"
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Example Configurations:

In Active Workspace, the configuration is:

```
{
  "configurationPreferencesConfiguration": {
    "setPreferences": {
      "actionType": "TcSoaService",
      "serviceName": "Administration-2012-09-PreferenceManagement",
      "method": "setPreferences2",
      "inputData": {
        "preferenceInput": "{{data.actionInput.preferenceInput}}"
      },
      "ignoreError": true
    },
    "getPreferences": {
      "actionType": "TcSoaService",
      "serviceName": "Administration-2012-09-PreferenceManagement",
      "method": "getPreferences",
      "inputData": {
        "preferenceNames": "{{data.actionInput.keywords}}",
        "includePreferenceDescriptions":
"{{data.actionInput.includeDescription}}"
      },
      "ignoreError": true
    }
  }
}
```

## Solution configuration for making cell titles clickable in list and table

As a consumer of Afx (‘Apollo’), application engineer can configure cell titles in list and table to be clickable and define commands for different types of clicks. The ‘clickableCellTitleActions’ key in solution definition (kit.json) provides support for four types of clicks on the cell title.

## Structure

```
{
  "clickableCellTitleActions": {
    "click": "<command id of command to execute on single click of cell
title>",
    "ctrlClick": "<command id of command to execute on ctrl + click of cell
title>",
    "shiftClick": "<command id of command to execute on shift + click of
cell title>",
    "doubleClick": "<command id of command to execute on double click of
cell title>"
  }
}
```

## Example Configuration:

In AW, the configuration for clickable cell title actions under solutionDef is:

```
{
  "clickableCellTitleActions": {
    "click": "Awp0ShowObjectCell",
    "ctrlClick": "Awp0OpenInNewTabCellTitle",
    "shiftClick": "Awp0OpenInNewWindowCellTitle",
    "doubleClick": "Awp0ShowObjectCell"
  }
}
```

The commands are defined in commandsViewModel; example for one of the command is given below. The object, for which the cell title was clicked, is available via 'commandContext.vmo'.

```
{
    "commands": {
        "Awp0OpenInNewTabCellTitle": {
            "iconId": "cmdOpenInNewTab",
            "title": ""
        }
    },
    "commandHandlers": {
        "Awp0OpenInNewTabCellTitleCommandHandler": {
            "id": "Awp0OpenInNewTabCellTitle",
            "action": "openInNewTabCellTitle",
            "activeWhen": true,
            "visibleWhen": true
        }
    },
    "commandPlacements": {},
    "actions": {
        "openInNewTabCellTitle": {
            "actionType": "Navigate",
            "navigateTo": "com_siemens_splm_clientfx_tcui_xrt_showObject",
            "navigationParams": {
                "uid": {
                    "$adapt": "{{commandContext.vmo}}",
                    "path": "uid"
                }
            },
            "navigateIn": "newTab"
        }
    },
    "onEvent": [],
    "messages": {},
    "conditions": {},
    "i18n": {}
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Custom locale configuration

By default, Siemens Web Framework provides out-of-the-box support for the following locales:

- cs_CZ
- de
- en_US
- es
- fr
- it
- ja_JP
- ko_KR,
- pl_PL
- pt_BR
- ru_RU
- zh_CN
- zh_TW

Although you currently *cannot* override any of the out-of-the-box supported locales, you *can* define new locale settings, as follows:

1. In your **module** folder, add a `localeDateTime.json` file at the same level as the `module.json` file.
2. Define the settings for additional locales as follows (`en_GB` is used in the following example):

```
{
    "en_GB": {
        "month_2": "MM",
        "month_3": "MMM",
        "month_4": "MMMM",
        "day": "DD",
        "year_2": "YY",
        "year_4": "YYYY",
        "hours": "HH",
        "minutes": "MM",
        "seconds": "SS",
        "monthAfterYear": false,
        "dayOfWeekAfterDay": false,
        "fullDayOfMonth": false,
        "fullMonthAndDayOfMonthWithDayOfWeek": false,
        "fullMonthFormatForMonth": false,
        "dayOfWeekSeparator": " ",
```

```
        "monthDaySeparator": "",
        "dateTimeSeparator": " "
    }
}
```

3. In the `src/i18n` folder of your module, create a `dateTimeServiceMessage_<locale>.json` file (`<locale>` is the locale code specified in the `localeDateTime.json` file, e.g. `en_GB`) and provide localizations for common locale-related text. This is an ordinary **localization file**:

```
{
    "closeText": "Done",
    "prevText": "Prev",
    "nextText": "Next",
    "currentText": "Today",
    "monthName_01" : "January",
    "monthName_02" : "February",
    "monthName_03" : "March",
    "monthName_04" : "April",
    "monthName_05" : "May",
    "monthName_06" : "June",
    "monthName_07" : "July",
    "monthName_08" : "August",
    "monthName_09" : "September",
    "monthName_10" : "October",
    "monthName_11" : "November",
    "monthName_12" : "December",
    "monthNameShort_01" : "Jan",
    "monthNameShort_02" : "Feb",
    "monthNameShort_03" : "Mar",
    "monthNameShort_04" : "Apr",
    "monthNameShort_05" : "May",
    "monthNameShort_06" : "Jun",
    "monthNameShort_07" : "Jul",
    "monthNameShort_08" : "Aug",
    "monthNameShort_09" : "Sep",
    "monthNameShort_10" : "Oct",
    "monthNameShort_11" : "Nov",
    "monthNameShort_12" : "Dec",
    "dayName_01" : "Sunday",
    "dayName_02" : "Monday",
    "dayName_03" : "Tuesday",
    "dayName_04" : "Wednesday",
    "dayName_05" : "Thursday",
    "dayName_06" : "Friday",
    "dayName_07" : "Saturday",
    "dayNameShort_01" : "Sun",
    "dayNameShort_02" : "Mon",
    "dayNameShort_03" : "Tue",
    "dayNameShort_04" : "Wed",
    "dayNameShort_05" : "Thu",
    "dayNameShort_06" : "Fri",
    "dayNameShort_07" : "Sat",
    "dayNameMin_01" : "Su",
    "dayNameMin_02" : "Mo",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "dayNameMin_03" : "Tu",
    "dayNameMin_04" : "We",
    "dayNameMin_05" : "Th",
    "dayNameMin_06" : "Fr",
    "dayNameMin_07" : "Sa",
    "weekHeader" : "Wk",
    "dateFormat": "mm/dd/yy",
    "yearSuffix": "",
    "anteMeridiem": "AM",
    "postMeridiem": "PM"
}
```

4. Add the identifiers of any new custom locale to the `locale` property of your **site configuration file**.

**Note**

Make sure the identifier of the locale is unique. Overriding existing locales is currently not supported.

## Adapters

While working on declarative UI, at times the object being displayed in DUI panel or the object on which command is being executed could be a proxy object. In such cases, the object needs to be adapted to a different object before performing any action on it. Declarative UI provides mechanism to adapt to a different object.

Let's say an application is working on a proxy object which is of type "SchTaskDeliverable". The object type is stored as an attribute on "modelType.typeHierarchyArray". The identifier for the underlying object is stored on a property named "fnd0DeliverableInstance" on the proxy object. So adapter.json needs to be defined like this.

`adapter.json`

```
[
    {
        "conditions": {
            "modelType.typeHierarchyArray": {
                "$in": "SchTaskDeliverable"
            }
        },
        "target": {
            "prop": "fnd0DeliverableInstance"
        }
    }
]
```

It has the capability to evaluate complex and/or conditions.

**Condition with single logical AND**

```
{
  "conditions": {
    "$and": [
      {
        "props.awb0Name.dbValues[0]": {
          "$eq": "2"
        }
      },
      {
        "props.awb0Name.uiValues[0]": {
          "$eq": "testProp"
        }
      }
    ]
  }
}
```

**Condition with single logical OR**

```
{
  "conditions": {
    "$or": [
      {
        "props.awb0Name.dbValues[0]": {
          "$eq": "2"
        }
      },
      {
        "props.awb0Name.uiValues[0]": {
          "$eq": "testProp"
        }
      }
    ]
  }
}
```

**Condition with combination of logical OR and AND**

```
{
  "conditions": {
    "$and": [
      {
        "$and": [
          {
            "props.awb0Desc.dbValues[0]": {
              "$eq": "This is a dummy description"
            }
          },
          {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "props.awb0Desc.uiValues[0]": {
              "$eq": "This is a dummy description"
            }
          }
        ]
      },
      {
        "$or": [
          {
            "props.awb0Name.dbValues[0]": {
              "$eq": "5"
            }
          },
          {
            "props.awb0Name.uiValues[0]": {
              "$eq": "testProp5"
            }
          }
        ]
      }
    ]
  },
  "target": {
    "prop": "awb0UnderlyingObject"
  }
}
```

**Using adapter in declarative actions**

Declarative UI framework provides "$adapt" keyword to adapt an object.

This is how the adapted object can be passed as an input for an action.

```
{
  "actions": {
    "removeFavorites": {
      "actionType": "JSFunctionAsync",
      "method": "removeFavorites",
      "inputData": {
        "selectedObjects": {
          "$adapt": "{{ctx.mselected}}"
        }
      }
    }
  }
}
```


Unpublished work. © 2025 Siemens

Let's say I want to pass a navigation parameter after adapting the proxy object. The following definition will adapt the selected object, retrieve the value from path "props.mdl0model_object.dbValues[0]" and pass it as a navigation parameter "uid".

```
{
  "actions": {
    "cm1ShowDelegatedObject": {
      "actionType": "Navigate",
      "navigateTo": "com_siemens_splm_clientfx_tcui_xrt_showObject",
      "navigationParams": {
        "uid": {
          "$adapt": "{{ctx.selected}}",
          "path": "props.mdl0model_object.dbValues[0]"
        }
      }
    }
  }
}
```

The adapt can work on an array as well. The following definition will adapt all the selected object, retrieve the uid from path and pass to the action as an array.

```
{
  "getDatasetRefProperties": {
    "actionType": "JSFunctionAsync",
    "method": "getProperties",
    "deps": "soa/dataManagementService",
    "inputData": {
      "uids": {
        "$adapt": "{{ctx.mselected}}",
        "path": "uid"
      },
      "propNames": [
        "ref_list",
        "release_status_list"
      ]
    },
    "events": {
      "success": [
        {
          "condition": "conditions.cellDatasetHasRefList",
          "name": "dataset.downloadFile"
        }
      ]
    }
  }
}
```

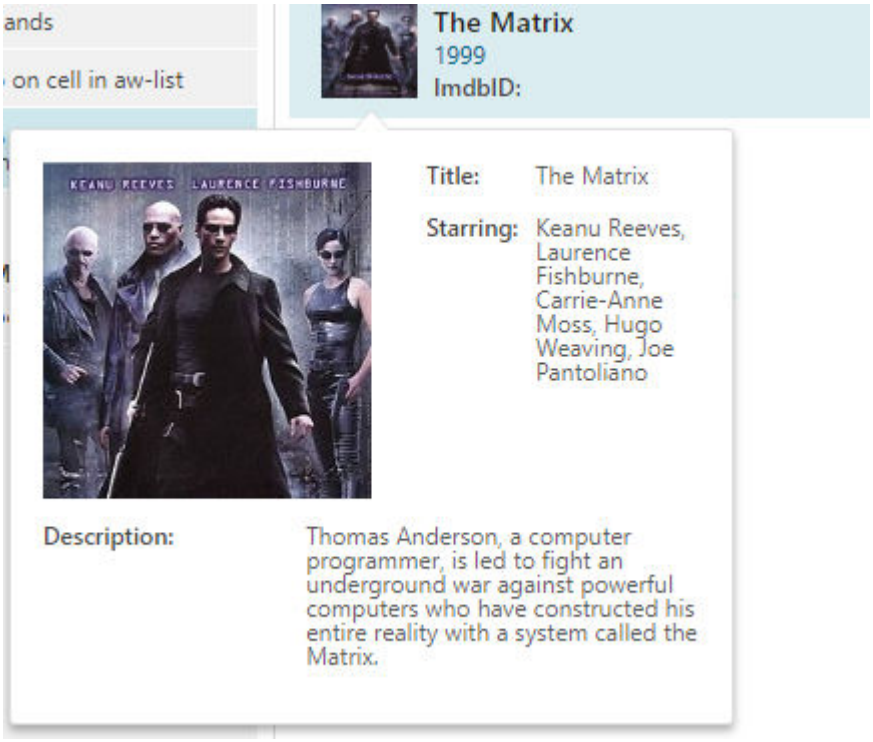

Siemens Web Framework
Unpublished work. © 2025 Siemens

# Extended tooltips

Tooltips are descriptive messages that are displayed near the cursor when it hovers over a toolbar or ribbon panel button or any UI element from the screen. Typically, we use tooltips with commands and to show the title property that is associated with a command and provides a simple description of the task that the command performs. However, simple description is not sufficient and there is a need for extended tooltip that would provide a more rich level of information about a command. The content for an extended tooltip comes from View Model. Extended tooltip content can contain rich-text and images. It can be used in simple UI elements, commands and custom cell.

**Default tooltip**

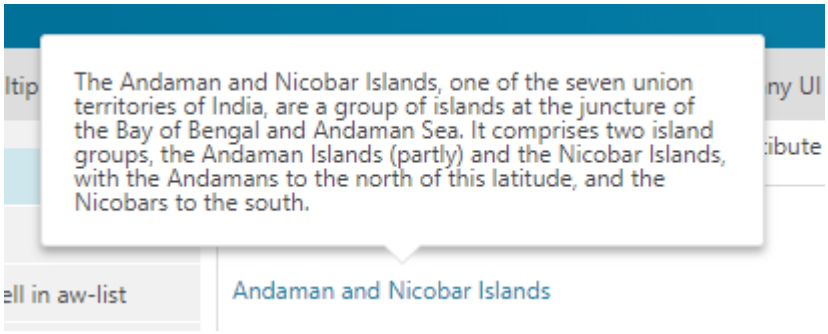
**Extended -tooltip**





Unpublished work. © 2025 Siemens

## Tooltip with only text




If an, extended tooltip with only text is to be shown on the hover of aw-link, then following is the change required in view file. You can configure it by 2 ways.

In view, the user has to provide the value for `extended-tooltip` attribute that takes any object(For eg: tooltipWithOutView) which is present in data section of the view model. The view will look like:

```
<aw-link prop="data.box1" extended-tooltip="data.tooltipWithOutView">
</aw-link>
```

In the view model, the tooltip object, that is, tooltipWithOutView should have `extendedTooltipContent` as a key which contains the text/evaluated text which will be shown in the tooltip.

```
{
  "data": {
    "tooltipWithOutView": {
      "extendedTooltipContent": "{{data.box2.uiValue}}"
    }
  }
}
```

To evaluate the box2 uivalue,add the following code snippet:

```
{
  "data": {
    "box2": {
      "uiValue": "{{i18n.toolTipContext}}"
    }
  },
  "i18n": {
    "toolTipContext": [
      "TCUICommandPanelsMessages"
    ]
  }
}
```

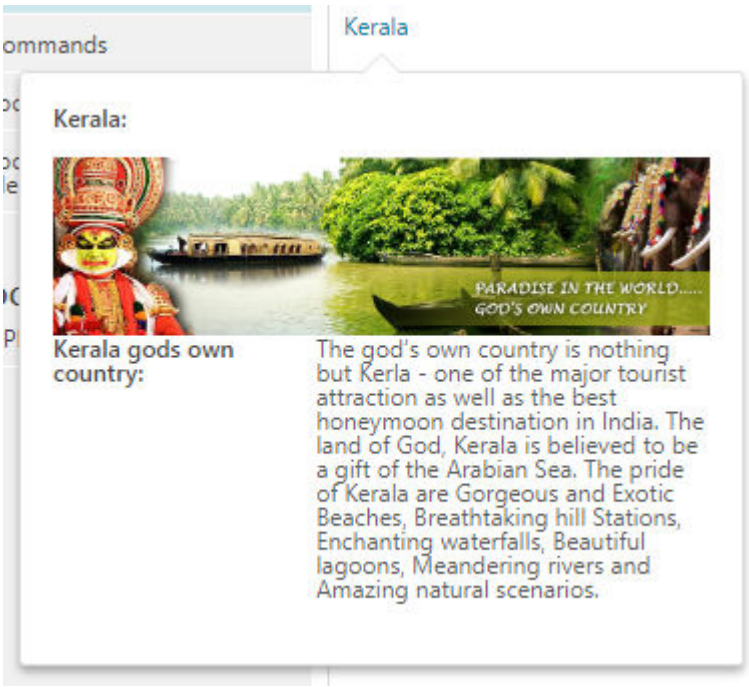

Siemens Web Framework
Unpublished work. © 2025 Siemens

If you do not have viewModel for your view file. You can pass object which includes extended tooltip content, like the following:

```
const tooltipContent = { extendedTooltipContent: 'Tooltip custom content' };
```

```
<aw-link-with-popup prop="prop" use-icon="useIcon" id={{id}} extended-
tooltip="{{tooltipContent}}">
</aw-link-with-popup>
```

## Tooltip with rich content




Extended-tooltip with rich content has to be shown on the hover of aw-link ,then following configuration is needed on the element.You can configure it by 2 ways.

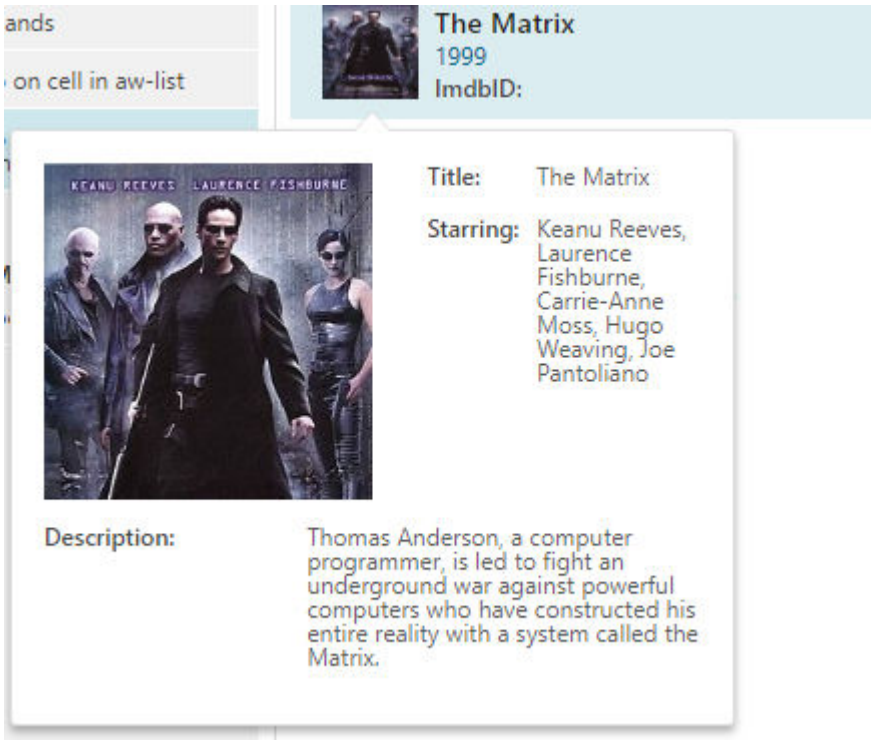
1. The view of the element where tooltip is shown on hover remains same as tool-tip with only text. However, in the view model the user has to configure `view` of the extended- tooltip. The `view` could be any view provided by the user. In this case "extendedTooltip" is the view which contains rich content for extended tooltip. The view model looks like :

```
{
    "data": {
    "tooltipWithView": {
        "view": "extendedTooltip"
        }
    }
}
```

2. If you do not have viewModel for your view file. You can directly pass view name(extendedTooltip) in UI element instead of fetching it from viewmodel like #1.

```
    <aw-link prop="data.box1" extended-tooltip="extendedTooltip"></aw-
link>
```

## Tooltip on cell of aw-list




When user wants to show tooltip on hover of cell in the aw-list, then the user should add `extended-tooltip-context` and `extended-tooltip` attributes on the custom cell element. The value of `extended-tooltip-context` will be the name of the object which will be present in the scope and the value of `extended-tooltip` is the any object(For eg: listTooltip) which is present in data section of the view model. The view will look like :

```
<aw-list dataprovider="data.dataProviders.performSearch">
    <aw-omdb-cell vmo="item" extended-tooltip-context="item"
        extended-tooltip="data.listTooltip">
    </aw-omdb-cell>
</aw-list>
```

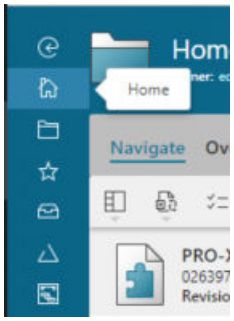
View model should be same as Tool-tip with rich content.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Tooltip on commands

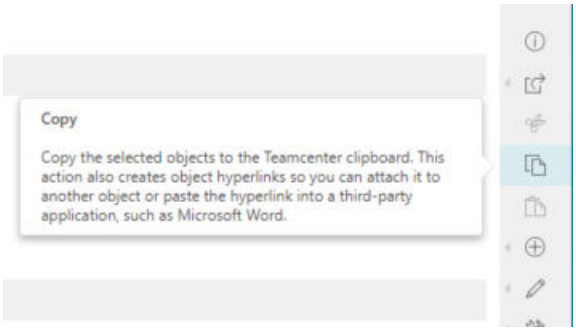
You can configure a rich tooltip to show styled texts, images etc. Here are examples of command configured to show default tooltip, tooltip with description and custom tooltip:

1. Here, in the example below, only the title is configured using the default extended tooltip.



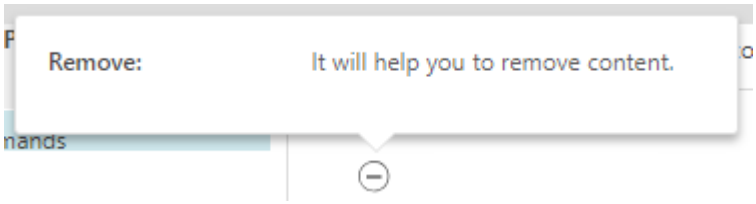

```
{
    "commands": {
        "Awp0home": {
            "iconId": "cmdHome",
            "title": "Home"
        }
    }
}
```

2. Here, in the example below, both the title and description are configured using the default extended tooltip.



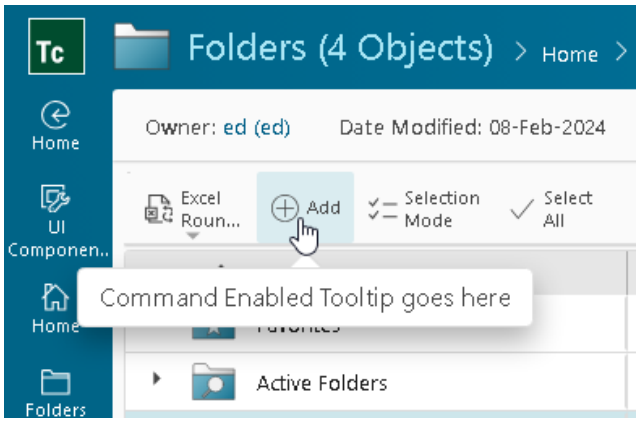
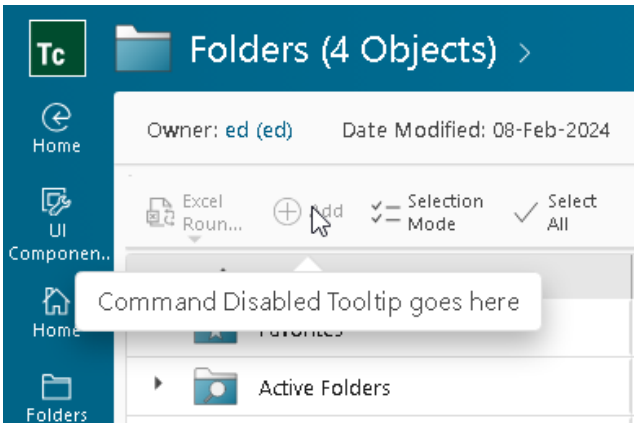

```
{
    "commands": {
        "Awp0CopyCommand": {
        "iconId": "cmdCopy",
        "title": "Copy",
        "description": "Copy the selected objects to the clipboard.
This action also creates object hyperlinks so you can attach it to
another object or paste the hyperlink into a third-party application,
sunch as Microsoft Word."
        }
    }
}
```

3. A custom tooltip is configured for the command to show the tooltip. To define a custom tooltip for your command, you need to pass the component name in the command's extendedTooltip definition.




```
{
    "commands": {
        "removeCmd": {
        "iconId": "cmdRomoveIco",
        "extendedTooltip": {
                "view": "extendedTooltipRemoveCmd"
            }
        }
    }
}
```

4. You can define advanced tooltips for commands using the command state available on subPanelContext as shown below. Command states like enabled and selected can be leveraged to show these advanced tooltips.

| Tooltip when Command is Enabled | Tooltip when Command is Disabled |
|---|---|
|  |  |



```
<div exist-when="props.subPanelContext.enabled">
    <!-- Command Enabled Tooltip goes here -->
</div>
<div exist-when="!props.subPanelContext.enabled">
    <!-- Command Disabled Tooltip goes here -->
</div>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

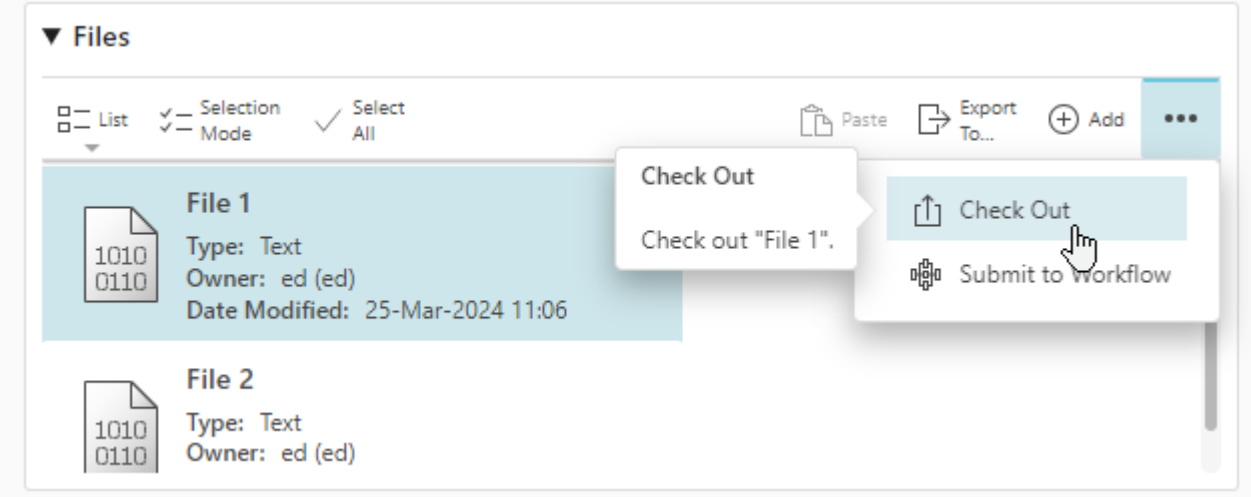
5. A dynamic tooltip is now supported with string replacement based on the current selection

```
{
    "commands": {
        "checkoutCmd": {
            "iconId": "cmdCheckout",
            "extendedTooltip": {
                    "messages": {
                        "single": "{{i18n.singleSelectMessage}}",
                        "multiple": "{{ii18n.multiSelectMessage}}"
                    },
                    "view": "ExtendedTooltipDefault"
                }
            }
    }
}
```

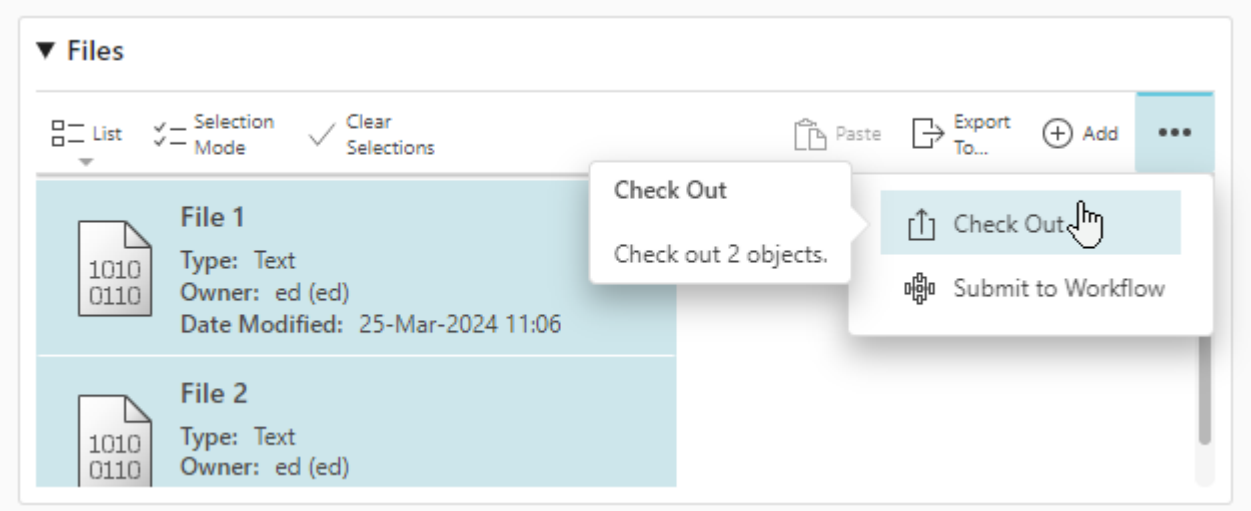
Where the i18n definition is:

```
{
    "singleSelectMessage": "Check out \"{0}\"",
    "multiSelectMessage": "Check out {0} objects"
}
```

singleSelectMessage will be shown for single selection.




And multiSelectMessage will be shown for multiple selection.




The `selectionData` present on the `commandContext` will be used for string replacement will a fallback to `ctx.mselected` where `selectionData` is not present.

**NOTE:**

From **Siemens Web Framework v4.2.0** onwards, any command which uses `extendedTooltipDefault` will be thrown as error in the audit run.

Following will be the message on prompt:

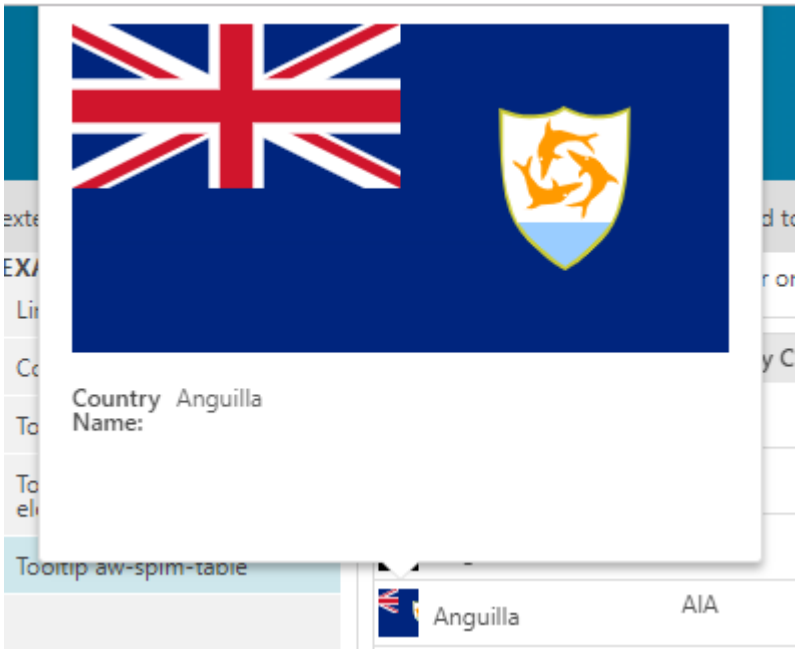
```
Command <commandId> no longer requires extendedTooltip configuration to disp
lay title or description in tooltip.
```

Hence, command configuration specified below is invalid:

```
{
  "commands": {
    "commandId": {
      "title": "commandTitle",
      "extendedTooltip": {
        "view": "extendedTooltipDefault"
      }
    }
  }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Tooltip on cell of aw-splm-table




When user wants to show tooltip on hover of cell in the aw-splm-table, then the user should add `extended-tooltip-context` and `extended-tooltip` attributes on the custom cell element.

```
const cell =`
<div class = "aw-splm-tableCellTop">
    <img
        class ="aw-base-icon aw-type-icon aw-splm-tableIcon" src="$
{vmo.typeIconURL}"
        extended-tooltip-context="tooltipDetails" extended-tooltip= "$
{dataDetails}" />
</div>`;
```

In the extended-tooltip-context, the user should pass string,that is, tooltipDetails, which should be present in the scope. In extended-tooltip user should configure dataDetails, which is defined in viewmodel of aw-splm-table. In dataDetails,the user assign string , that is, "data.tableTooltip". The object of that string is provided in the data section of view model.

```
{
  "actions": {
    "loadColumnsAndAddCellRenderer": {
      "actionType": "JSFunction",
      "method": "loadColumnsAndAddCellRenderer",
      "inputData": {
        "columnProvider": "{{data.columnProviders.exampleColumnProvider}}",
        "dataProvider": "{{data.dataProviders.exampleDataProvider}}",
        "dataDetails": "data.tableTooltip"
      },
```

```
      "deps": "js/splmTableTooltipProvider"
    }
  },
  "data": {
    "tableTooltip": {
      "view": "countryTooltip"
    }
  }
}
```

The user has to inject `ngUtils` dependency in JS file and convert HTML string to angularJS element by calling `element()` function and pass HTML content, that is, cell as a parameter. After that it should compile an angularJS template to HTML DOMElement by calling `compile()` function and pass parent element, angularJS element and scope data(any object to be set on the primary '$scope'), that are, tableElem, currentElement and tooltipObject respectively as parameters.

```
var tooltipObject = {
        tooltipDetails:{
        countryImg: vmo.typeIconURL,
        countryName: vmo.props.name && vmo.props.name.uiValue
        }
    };
var currentElement = ngUtils.element( cell );
var compiledResult = ngUtils.compile( tableElem, currentElement, null,
null, tooltipObject );
```

## Usage of subPanelContext in Tooltip for Widgets

## Usage in Commands

The `subPanelConetxt` can be accessed from the extended tooltip, given the properties to be used in the tooltip view, should be configured as follows in command definition in `commandsViewModel.json`:

```
{
  "checkOutCommandExtWithSubPanelContext": {
    "iconId": "cmdCheckout",
    "isGroup": false,
    "title": "With SubPanelContext",
    "extendedTooltip": {
      "view": "extendedCheckoutCommandWithSubPanelContext",
      "subPanelContext": {
        "ctxLabel": "{{ctx['location.titles'].headerTitle}}",
        "labelFromCommandContext":
"{{commandContext.labelFromCommandContext}}"
      }
    }
  }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

In `extendedCheckoutCommandWithSubPanelContextViewModel.json` file, the data can be defined as below, which uses the properties configured:

```
{
  "actions": {},
  "data": {
    "tooltipProperty1": {
      "displayName": "From ctx",
      "type": "STRING",
      "isRequired": "false",
      "isEditable": "true",
      "dbValue": "{{subPanelContext.ctxLabel}}",
      "dispValue": "{{subPanelContext.ctxLabel}}"
    },
    "tooltipProperty2": {
      "displayName": "From command Context",
      "type": "STRING",
      "isRequired": "false",
      "isEditable": "true",
      "dbValue": "{{subPanelContext.labelFromCommandContext}}",
      "dispValue": "{{subPanelContext.labelFromCommandContext}}"
    }
  }
}
```

In the `extendedCheckoutCommandWithSubPanelContextView.html` file, here is how the data is accessed:

```
<aw-flex-column width="fill">
    <aw-label prop="data.tooltipProperty1"></aw-label>
    <aw-label prop="data.tooltipProperty2"></aw-label>
</aw-flex-column>
```

## Usage in other widgets, like Chips

This can now also be used in other widgets instead of using the `extended-tooltip-context` attribute.

A `aw-chip` widget displaying tooltip accessing `commandContext` and `ctx`.

```
<aw-chip extended-tooltip="data.extendedTootipData">
```

Defining the data in view model,

```
{
  "data": {
    "extendedTootipData": {
      "view": "MyTooltip",
      "subPanelContext": {
```

```
        "dataToPass": "{{ctx.selectedObject}}"
      }
    }
  }
}
```

The view and the view model for `MyTooltip` can now be defined similarly as above.

# Layout slots

**Note**

This functionality is available from **Siemens Web Framework v3.2.0** onwards.

A Layout Slot is a configurable element where user can specify their own view/viewmodel against the slots. In case user is not willing to change the UI provided by the framework, Common Frame already has its own default view/viewmodel wherever required.

## Authoring aw-layout-slot

To use the slot , consumer needs to add aw-layout-slot in their view.html The attributes that aw-layout-slot has

1. `name` : This is a placeholder against which the view file is contributed. To maintain the uniqueness, it's recommended to to use the combination of the viewName+uniqueAnchor.
2. `class` : Specify a class *aw-layout-fill* if the slot contents are expecting to grow. Specify a class *aw-layout-fit* if the slot contents are of fix size.
3. `context` : This attribute is used to pass the data from parent view to contributed view.

```
<aw-layout-slot name="aw_avatar" context="localData" class="aw-layout-
fill"/>
```

This would be sufficient to identify an area of the interface as a layout slot called `aw_avatar`, and optionally passing some context data (localData) to the contributed view/viewModel.

## Contributing the custom view against named slot

aw-layout-slot reads the data from the configuration file layoutSlots.json. This configuration file holds the slots and it's visibility conditions.

- Contribute custom view to any slot that has been created anywhere in the UI


Siemens Web Framework
Unpublished work. © 2025 Siemens

- Specify visibility conditions.

```
{
    "slots": {

        "avatarSlot": {
            "name": "aw_avatar",
            "view": "commonAvatar",
            "activeWhen": {
                "condition": "conditions.avatarEnabled"
            }
        }
    },
    "conditions": {
        "avatarEnabled": {
            "expression": "!ctx.disableAvatar"
        }

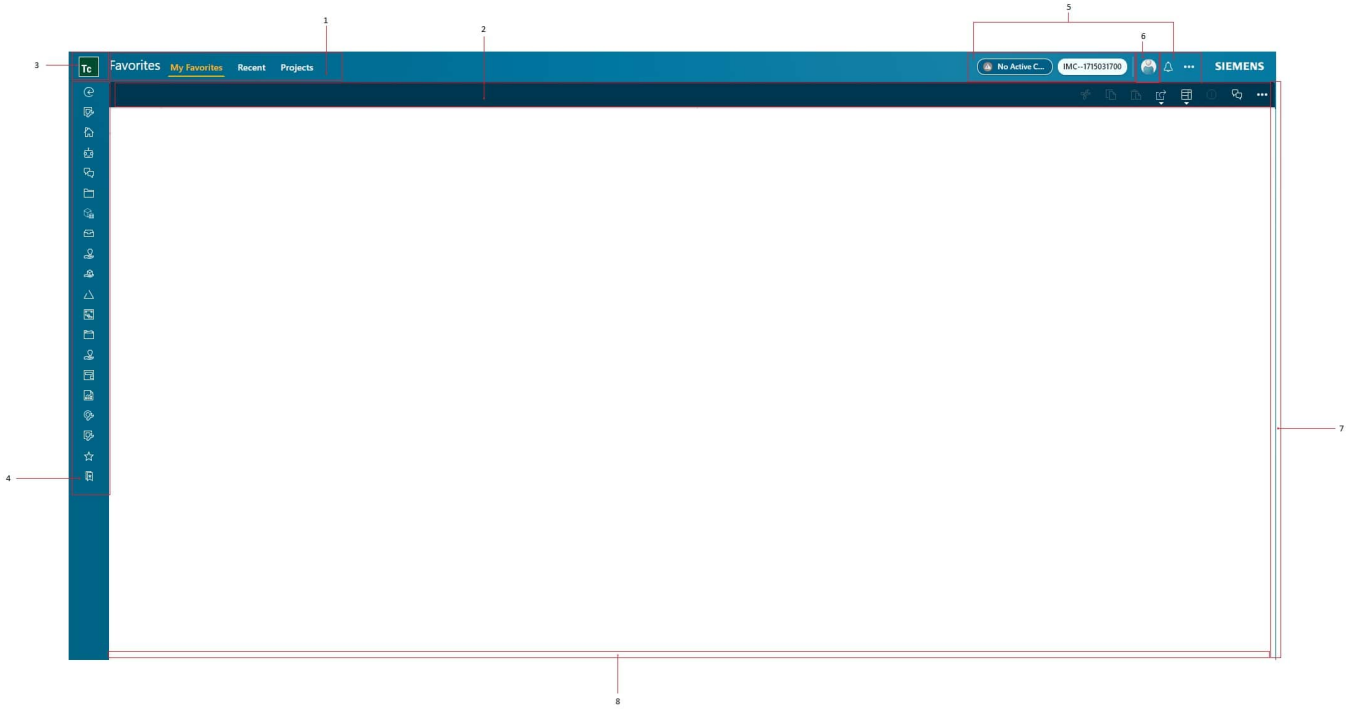
    }
}
```

**Slot can be determined by**

1. If there is only one contribution for a slot and its condition is satisfied, render the specified view.
2. If there are multiple contributions to the same slots but only one condition is satisfied, render the view specified in the slot for which the condition is satisfied.
3. If there are multiple contributions to the same slots and more than one conditions are satisfied, render the view specified in the slot for which the *longest* (ideally more complex) condition is satisfied.
4. Use the context to use the in condition expression.

## Example

Consider the following example




Here, all highlighted areas are the slots provided by framework in the common frame. User can contribute their custom view/viewmodel.

In this way, consumers may decide to:

- Disable (hide) certain slots, by making the corresponding condition fail by providing the condition expression.
- Provide alternative contents for one or more slots.
- Provide class for one or more slots to fill or fit the contents.

# Sync strategy

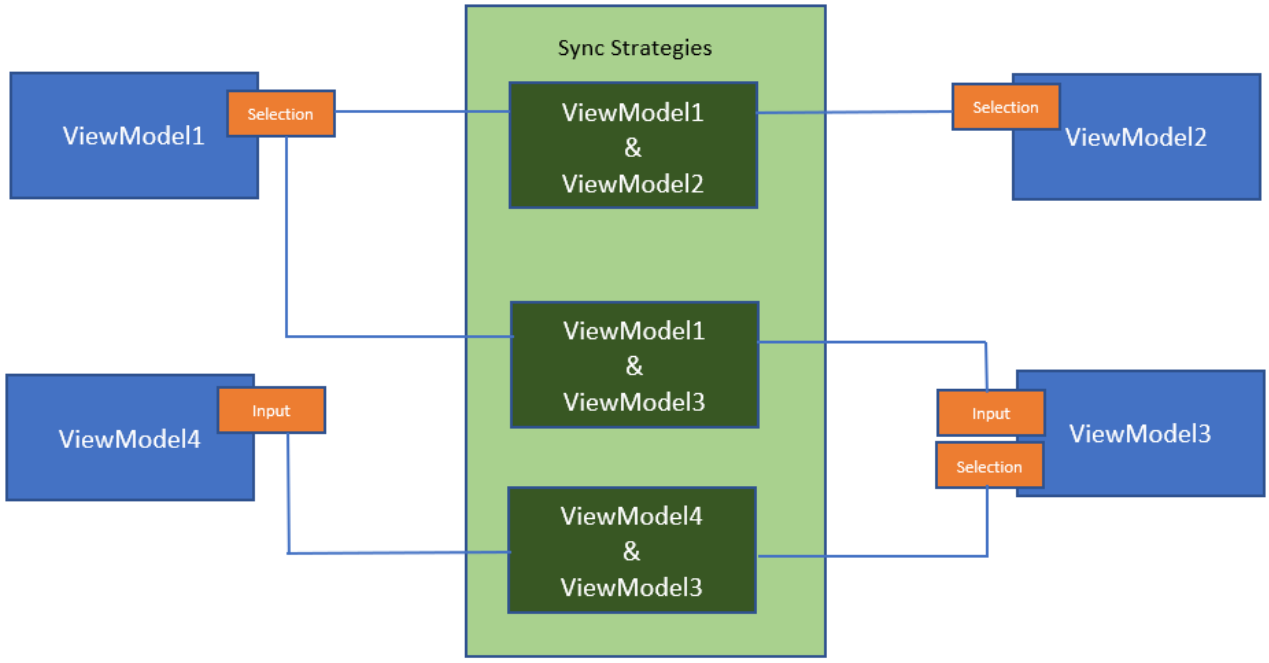
Sync Strategy is a declarative configuration point which enables you to define the interactions between any two viewModels. These interactions could be of any nature – say the selection of one viewModel needs to be fed as an input to another viewModel (Master Details pattern) or the selection in one needs to be synchronized (shown as selected or highlighted) in another viewModel etc.

One way to achieve these interactions would be via events where the identities of the viewModels are known to each other. This poses a problem for the reuse of viewModels.

Sync Strategy is a declarative data binding between viewModels which lets user do the configuration of these interactions/bindings into a separate configuration file; thus letting the viewModels be re-usable and completely unaware about each other.


Unpublished work. © 2025 Siemens

This is achieved by introducing 'ports' on the viewModels. These ports are data bound in the configuration file known as syncStrategy.json. This is a module level configuration.




## Defining syncStrategy.json

Example:

Considering two example views with names 'movies' and 'movieDetails' containing the ports with names 'moviesSelection' and 'movieDetailsInput' respectively, here is how the sync strategy configuration would look like:

```
{
  "schemaVersion": "1.0.0",
  "syncStrategies": {
    "movieStrategyObject": {
      "source": {
        "view": "movies",
        "port": "moviesSelection"
      },
      "target": {
        "view": "movieDetails",
        "port": "movieDetailsInput"
      }
    }
  },
  "syncStrategyHandlers": {
```

```
    "movieSyncStrategyHandler": {
      "id": "movieStrategyObject",
      "activeWhen": true
    }
  }
}
```

| Field | Description |
|---|---|
| `syncStrategies` | It contains a list of objects which define the relationship between two specific views and their ports. Multiple such objects can be defined. |
| `syncStrategyHandlers` | It contains a list of syncStrategyHandlers which using the declarative "activeWhen" clause specify the declarative "conditions" under which the corresponding sync strategy should be applied. It can also take an optional declarative "action" which is a user defined data conversion/pre-step to be performed before the sync strategy is applied. Atleast one handler per Sync Strategy object needs to be defined. 'id' is the mandatory field which maps the syncStrategyHandler to the syncStrategyObject. |

Please refer Sync Strategy Schema for more information.

There can be multiple syncStrategy objects defined for one viewModel to multiple viewModels depending upon the interactions required between various viewModels.

## Defining Ports

Ports are the observables defined on the viewModels which can be influenced either internally by the owning viewModel or by the syncStrategy object defined in the configuration file.

Example:

Here is how the port with name 'movieDetailsInput' would look like:

```
{
  "ports": {
    "movieDetailsInput": {
      "syncObject": {},
      "onChangeAction": "fetchMovieDetails"
    }
  }
}
```

Here 'fetchMovieDetails' is the name of the example action which needs to be invoked whenever the value of the syncObject on port 'movieDetailsInput' is changed.


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Field | Description |
|---|---|
| `syncObject` | It is the object which gets data bound between the source and the target viewModels. Once populated on the source viewModel and Sync is triggered; it is data bound to the target viewModel by the Sync Strategy. Multiple ports can be defined on a viewModel. Same ports and views need to be referred in the Sync Strategy configuration. This object is mandatory to be defined on both Source and Target viewModels. It is available in both view as well as the viewModel and can be referred directly as ports.movieDetailsInput.syncObject. This object is available on the view and can be referred directly in the view as well as viewModel. For current example, it will be accessed as ports.movieDetailsInput.syncObject. |
| `onChangeAction` | It is the declarative "action" that needs to be executed once the syncObject defined on that port is changed. It is mandatory to be defined on the Target viewModel. |

## Triggering Sync

Sync Strategy needs to be explicitly triggered by the Source viewModel by invoking a declarative "action" of type "Sync". The mandatory input parameters are the port on which the Sync Strategy needs to be triggered along with the syncObject containing the data values which need to be synchronized on the target viewModel.

Example: Here 'updateMovieSelectionPort' is the example action defined on the Source viewModel which is responsible for triggering the Sync on example port named 'moviesSelection' with the value populated in syncObject.

```
{
  "actions": {
    "updateMovieSelectionPort": {
      "actionType": "Sync",
      "inputData": {
        "port": "moviesSelection",
        "syncObject": "{{data.eventData.selectedObjects}}"
      }
    }
  }
}
```

| Field | Description |
|---|---|
| `port` | It is a mandatory input parameter for the action of type 'Sync'. It indicates the name of the port on which sync Action needs to be executed. |
| `syncObject` | It is a mandatory input parameter for the action of type 'Sync'. It indicates the data which needs to be pushed on the specified input port. |

# Drag and drop

Drag and Drop (DnD) is powerful User Interface concept which makes it easy to copy, reorder and delete items with the help of mouse clicks. This allows the user to click and hold the mouse button down over an element, drag it to another location, and release the mouse button to drop the element there.

**Note**

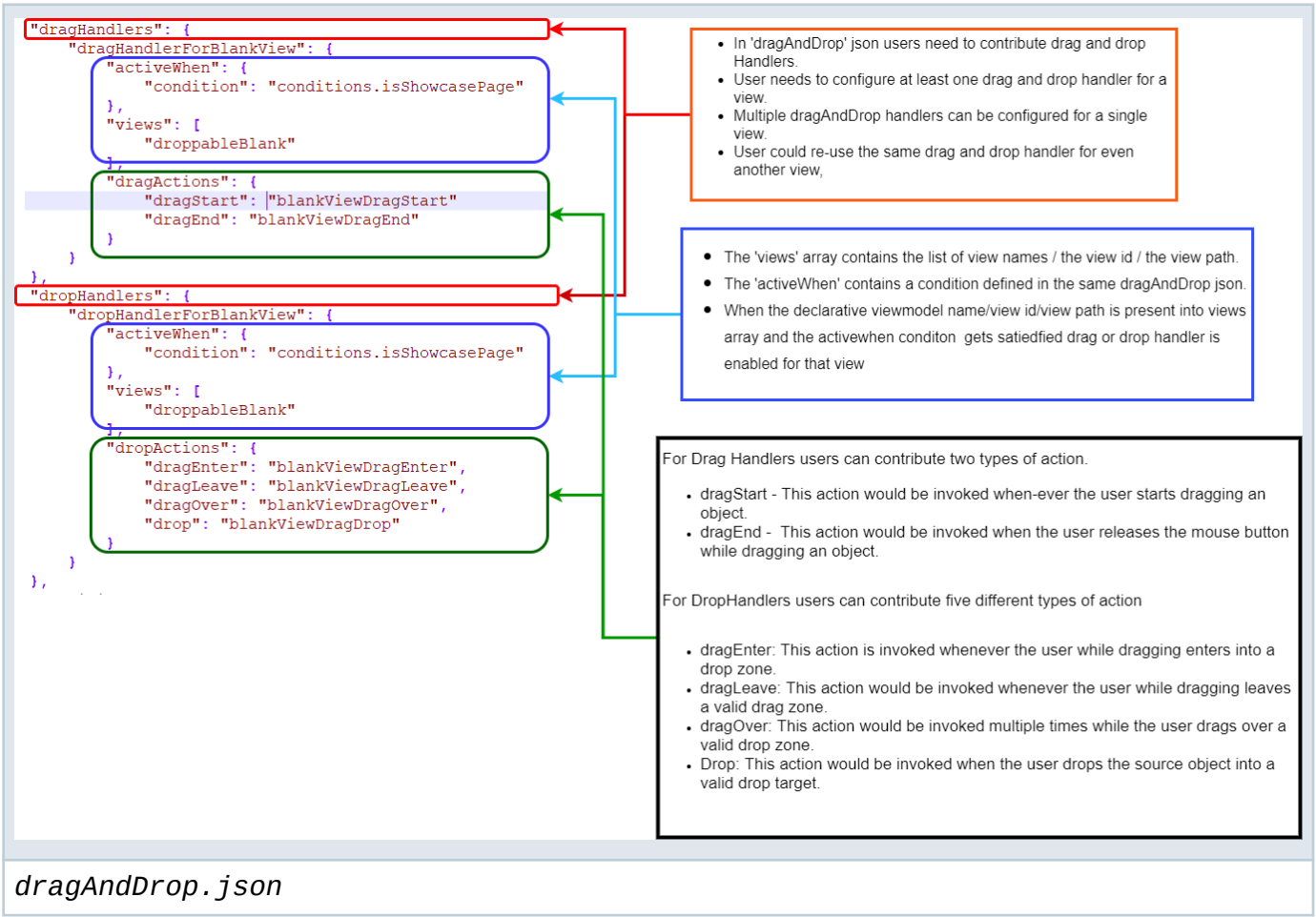
Available from **Siemens Web Framework v4.2.0** onwards.

Siemens Web Framework supports drag and drop capability to create interactive applications. It can be configured per view basis by defining a configuration file 'dragAndDrop.json' at the module level.

## DragAndDrop JSON Format

```
{
  "dragHandlers": {
    "dragHandlerForBlankView": {
      "activeWhen": {},
      "views": [],
      "dragActions": {
        "dragStart": "",
        "dragEnd": ""
      }
    }
  },
  "dropHandlers": {
    "dropHandlerForBlankView": {
      "activeWhen": {},
      "views": [],
      "dropActions": {
        "dragEnter": "",
        "dragLeave": "",
        "dragOver": "",
        "drop": ""
      }
    }
  },
  "actions": {},
  "conditions": {}
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens




*dragAndDrop.json*

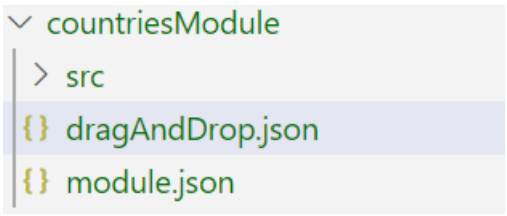
## Configuring Drag and Drop

Consider two views:

- `countriesList` view consists of list of countries.
- `countryDetails` view shows details about a country when a cell from the country list is dropped on it.

Let us see how we can configure the drag and drop for the above use case:

1. Add a `dragAndDrop.json` configuration file at the module level.



2. **Enable drag for countriesList view**. To enable drag on the list view, define a dragHandler for the *countriesList* view with the corresponding drag actions in the *dragAndDrop.json*.

```
{
  "dragHandlers": {
    "dragHandlerForListView": {
      "activeWhen": true,
      "views": [
        "countriesList"
      ],
      "dragActions": {
        "dragStart": "dragStartFn",
        "dragEnd": "dragEndFn"
      }
    }
  },
  "actions": {
    "dragStartFn": {
      "actionType": "syncFunction",
      "method": "listViewDragStart",
      "inputData": {},
      "deps": "js/showcaseDragAndDrop"
    },
    "dragEndFn": {
      "actionType": "syncFunction",
      "method": "listViewDragEnd",
      "inputData": {},
      "deps": "js/showcaseDragAndDrop"
    }
  }
}
```

3. **Enable drop on countryDetails view**. To enable drop on the countryDetails view, define a dropHandler for the *countryDetails* with corresponding drop actions in the *dragAndDrop.json*.

```
{
  "dropHandlers": {
    "dropHandlerForDetailsView": {
      "activeWhen": true,
      "views": [
        "countryDetails"
      ],
      "dropActions": {
        "dragEnter": "dragEnterFn",
        "dragLeave": "dragLeaveFn",
        "dragOver": "dragOverFn",
        "dragEnd": "dropFn"
      }
    }
  },
  "actions": {
    "dragEnterFn": {
      "actionType": "syncFunction",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "method": "blankViewDragEnter",
          "inputData": {
            "eventType": "blankViewDragEnter"
          },
          "deps": "js/showcaseDragAndDrop"
        },
        "dragLeaveFn": {
          "actionType": "syncFunction",
          "method": "blankViewDragLeave",
          "inputData": {
            "eventType": "dndDrop"
          },
          "deps": "js/showcaseDragAndDrop"
        },
        "dragOverFn": {
          "actionType": "syncFunction",
          "method": "blankViewDragOver",
          "inputData": {
            "eventType": "blankViewDragOver"
          },
          "deps": "js/showcaseDragAndDrop"
        },
        "dropFn": {
          "actionType": "syncFunction",
          "method": "blankViewDragDrop",
          "inputData": {
            "eventType": "blankViewDragDrop"
          },
          "deps": "js/showcaseDragAndDrop"
        }
      }
    }
```

## Drag Handlers

`dragHandlers` are used to enable and customize the drag operation for a view and the components inside it. Not all the components used inside a view are 'draggable', when a dragHandler is active for a view. 'Siemens Web Framework' as of now supports drag operation for the below components:

- Rows for aw-splm-table.
- cell for aw-list.

This means if a dragHandler is enabled for a view and the view contains any of the above mentioned widget, then user can customize the drag operations for them.

## When the dragHandler becomes active for a view?

dragHandler become active for a view when **both** the below conditions are satisfied for a view.

- Views: This is an array of view names/view id or view Path. If the current declarative view model name is present inside this views array dragHandler become active.
- activeWhen: The condition against it should be evaluated to true.

## What are the different types of action user can customize for dragHandler ?

User can customize two types of action for drag handler:

| DragActions | Description |
|---|---|
| `dragStart` | This action would be invoked when user starts a drag operation on a drag enabled component in a view. The caching of the dragged data can be done during this phase. |
| `dragEnd` | This action would be invoked when a drag operation is ended either by releasing the mouse button or by hitting the escape key. Clean up of the cached dragged data can be done during this phase, if needed. |

## Drag Action definition

The action associated bind-ed with drag actions is expected to be a **synchronous javascript action**. Consumer can only associate declarative action type `syncFunction` with drag actions. At runtime the js function (bind-ed with drag action) receives a system generated object as the last parameter of the function.This system generated parameter contains multiple objects to help user to customize the drag operations.Below is detailed structure of the system generated object.

```
const dragStartFn = ([, arg1[, arg2[, ...argN-1], [....dragAndDropParams]]])
```

**Parameters**:

- `[, arg1[, arg2[, ...argN-1]]`: User defined parameters in the 'inputdata' section of action definition.
- `dragAndDropParams`: System generated parameter.
  - `event` : The browser generated 'DragEvent'.
  - `targetElement`: The DOM element on which the drag start operation is performed.
  - `targetObjects` : This would contain the objects selected when drag start operation started.For table and list component, the targetObject field will contain the list of objects selected for drag.For other components this field would be null.
  - `declViewModel` : This would the contain the declViewModel of the component on which drag start operation is performed.
  - `callbackAPIs`: NA.

`Return Value`:


Siemens Web Framework
Unpublished work. © 2025 Siemens

- During the *dragStart* phase, a custom drag image can be set for the dragged element. To set the custom drag image, return an object with the following information:

```
return {
    setDragImage: {
        dragImage: `<Element>: customImageElement`,
        xOffest: `<Integer>: xOffsetValue`,
        yOffest: `<Integer>: yOffsetValue`
    }
}
```

## Drop Handlers

```
{
  "dropHandlers": {
    "dropHandlerForBlankView": {
      "activeWhen": {
        "condition": "conditions.isShowcasePage"
      },
      "views": [
        "droppableBlank"
      ],
      "dropActions": {
        "dragEnter": "blankViewDragEnter",
        "dragLeave": "blankViewDragLeave",
        "dragOver": "blankViewDragOver",
        "drop": "blankViewDragDrop"
      }
    }
  }
}
```

`dropHandlers` are used to enable and customize the drop operation for a view and the components inside it. If a dropHandler is activated for a certain view, then the same dropHandler becomes applicable to all the components inside the view. This means, consumer can handle any drop/dragEnter/dragOver operation for any component inside a view at the view level. Not all the components used inside a view have drop configured, when a dropHandler is active for a view. 'Siemens Web Framework' as of now supports drop operation for the below components:

- aw-splm-table: table container, individual rows of table.
- aw-list: list container, individual cells of a list.
- aw-file-upload
- view element

For few components i.e aw-list and aw-splm-table framework does some special operation to provide some parameters to consumer to ease the drag and drop operation.


Unpublished work. © 2025 Siemens

## When the dropHandler becomes active for a view ?

dropHandler become active for a view when **both** the below condition is satisfied for the view.

- Views: This is an array of view names/view id or view Path. If the current declarative view model name is present inside this views array dropHandler become active.
- activeWhen: The condition against it should be evaluated to true.

## `Drop Actions`

User can customize four different types of action for drop Handler.

| DropActions | Description |
|---|---|
| `dragEnter` | This action would be invoked when mouse enters(while dragging) a drop area. |
| `dragLeave` | This action would be invoked when mouse leaves(while dragging) a drop area. |
| `dragOver` | This action would be invoked multiple times when user hover the mouse(while dragging) into a valid drop area. |
| `drop` | This action is invoked when user releases the mouse drag on a valid drop area. |

## Drop Action definition

The action bind-ed with the drop actions is expected to be a `synchronous javascript action`. User can only use of action type 'syncFunction'. At runtime the synchronous js function (bind-ed with drag end) receives a system generated object as the last parameter of the function. This system generated parameter contains multiple objects to help user to customize the drop operations.

```
    let dropActionFn = ([, arg1[, arg2[, ...argN-1],
[....dragAndDropParams]]])
```

**Parameters**:

- `[, arg1[, arg2[, ...argN-1]]` : User defined parameters in the 'inputdata' section of action definition.
- `dragAndDropParams` : System generated parameter.
  - `event` : The browser generated event.
  - `targetElement`: The drop enabled component on which event is triggered.
  - `targetObjects` : For splm-table and list, this field would contain the target vmo beneath the mouse pointer.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- `declViewModel` : This field would contain the declViewModel of the component on which event is triggered.
- `callbackAPIs`: Object containing the framework level Api's to assist drag and drop.

**Return Value**:

During the `dragEnter` and `dragOver` operation if the targetElement is a valid drop container, return an object with the following properties.

```
if( validDropArea) {
     return {
         preventDefault: true,
         dropEffect: 'copy/move/link'
      }
 } else {
     return {
         dropEffect: 'none'
      }
 }
```

The drop effect gives a visual feedback to the user regarding the type of operation that will be performed if the dragged data in dropped on the particular element.

```
To invoke drop event on a valid drop container the default browser listners
need to be cancelled( preventDefault: true ) in the dragEnter and dragOver
phase.
```

## Drag And Drop API in details.

1. `highlightTarget`: This api is provides a visual indication of a valid drop target. Element can be highlihgted/unhighlighted using the *isHighlightFlag*. This api expects an object with the following keys:

| Key | Description |
|---|---|
| `isHighlightFlag` | Boolean value indicating whether the element is to be highlighted or not. |
| `targetElement` | The element that needs to be highlighted/unhighlighted. |

```
Element(s) can also be highlighted by publishing
'dragDropEvent.highlight' event. This comes handy when the ask is to
highlight multiple elements on the page. The eventData structure is
same as defined above fot he api.
```

2. `updateFileData`: This callbackAPI is specific to aw-file-upload widget. Updating the file data on the aw-file-upload widget is a unlike all the other widgets in Siemens Web Framework. This api

basically updates the dropped file data on the widget. This api expects the following arguments: | Key | Description | |:--------|:----------------------| | `event` | The drop event on the aw-file-upload widget.

## Navigation URL Token Configuration *(Teamcenter-specific)*

You can use a **navigationURLToken.json configuration file** to configure a hyperlink for a reference **View Model Property** on **View Model Object**.

1. To see a reference property in a specific location when it is opened using an **aw-widget** or custom form, create a `navigationURLToken.json` configuration file in your module folder, at the same level as the `src` folder.
2. In this configuration file, define **navigationHandlers** as shown in the following example. Handlers will be activated when the `activeWhen` condition is evaluated to `true`.
3. Then, define **conditions**. This follows the standard Siemens Web Framework **condition** format supported in either string or object-based form.
4. Finally, define **actions**. Actions must be of type **Navigate** and support all the attributes supported by this action type.

Here is a sample configuration for a custom Business Object.

```
{
  "$schema": "declarativeNavigationURLTokenSchema-1.0.0.json",
  "navigationHandlers": {
    "cusotmHandler": {
      "action": "showCustomLocation",
      "activeWhen": {
        "condition": "conditions.isObjectApplicableToOpenInCustomLocation"
      }
    }
  },
  "conditions": {
    "isObjectApplicableToOpenInCustomLocation ": {
      "expression": {
        "$source": "localContext.vmo",
        "$query": {
          "$typeOf": "CustomBoType"
        }
      }
    }
  },
  "actions": {
    "showCustomLocation ": {
      "actionType": "Navigate",
      "navigateTo": "customLocation",
      "navigationParams": {
        "uid": {
          "$adapt": "{{localContext.vmo}}",
          "path": "uid"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        }
      },
      "navigateIn": "newTab"
    }
  }
}
```

**Note**

For reference properties that support showing their content in the Show Object Location, there's no need for this override, as Active Workspace has default configuration to open those properties in the Show Object Location.

### Migrating navigationURLToken.json files created before version 7.1 of Siemens Web Framework

If you created a `navigationURLToken.json` file befor version 7.1 of Siemens Web Framework (Teamcenter 2406), you must migrate them to the new format that was introduced from this version onwards.

To do so, you can do one of the following:

- From the Active Workspace staging directory, execute the the command `npm run migrateConfigs`. This command will migrate all applicable configurations to the new format.
- Modify the file manually, and:
  - Convert the existing `navigations` object to `actions` within the `actions` section.
  - Convert existing `conditions` object to the standard condition based on which which each handler should be true.
  - Define a `navigationHandler` section with an `action` and an `activeWhen` condition, ensuring it is associated with both actions and conditions defined above.

## Tab

Tabs in Siemens Web Framework are defined and configured declaratively that provide a graphical means for the user to work with their product, process or other data.

Declarative Tabs leverage the same View Model concepts such as **localization**, **conditions** and **actions**.

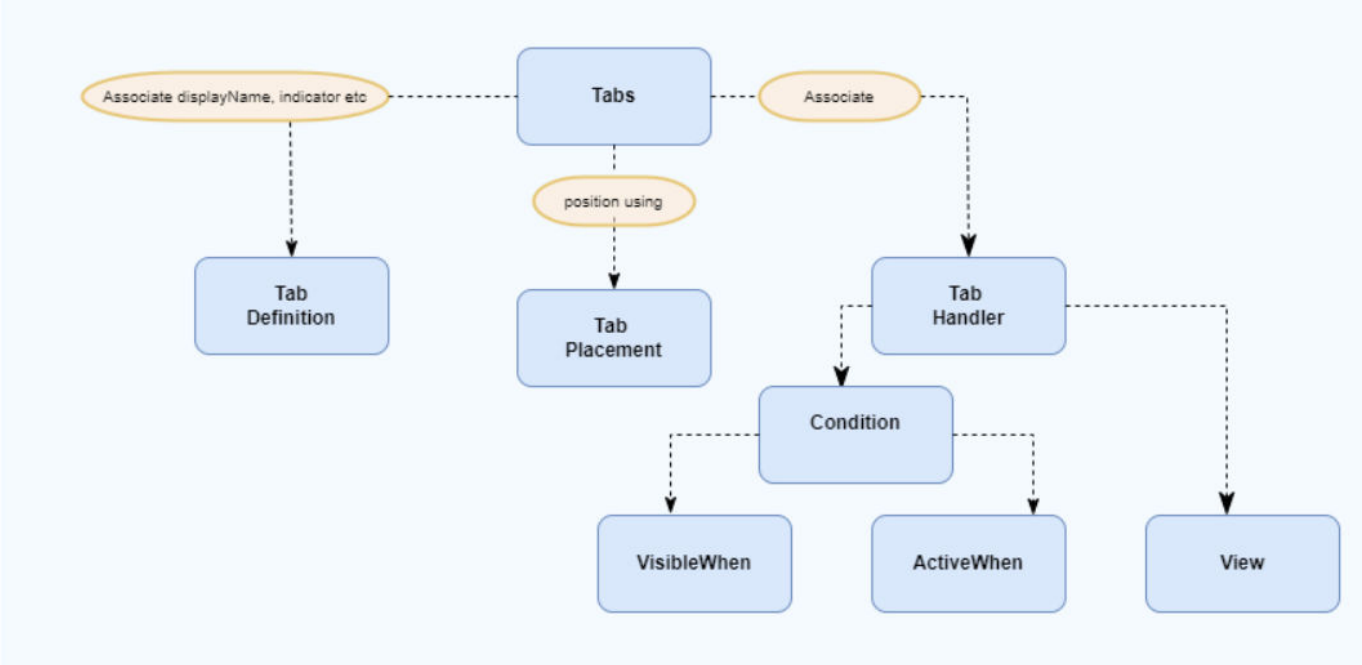
In general, any tab needs to have the following while being defined:

- **Appearance** - like an icon, label and a tooltip.
- **Behavior** - such as when it should be presented to the user, when it should be active, visible etc.


Unpublished work. © 2025 Siemens

- **Position** - where it should be placed in the UI relative to other Tabs.

In Siemens Web Framework, the View has **tab bars** in which **Tabs** can be placed (the elements). The View Model is responsible for view state such as data, actions, conditions and localizations in which we describe the tab and its behaviours.




In summary, with declarative Tabs you can:

- Define a Tab including ID, Indicator and displayName.
- Define where a tab is presented in the UI through 'uiAnchor' and set the order in which the tabs are displayed using priority.
- Define conditional expressions as to when a tab is visible.
- Leverage existing Declarative UI View Model concepts such as i18n, Condition, and Action.
- Use tab context to provide additional data that may be necessary to evaluate tab visibility or tab execution

## How are Tabs declared

Define your Tabs view model in a `tasbsViewModel.json` file right next to your module's module.json file.

When building the *site*, the Siemens Web Framework tooling concatenates all the modules' tabsViewModel.json files into a single `tabsViewModel.json` file.

The `tabsViewModel.json` has the following properties:


Unpublished work. © 2025 Siemens

## Tab Definition

Tab definition helps defining a tab uniquely. The name of the tab (`iconTab1` in example below) is referred as id. Tab is defined inside tabsViewModel.json file.

```
{
  "tabs": {
    "sampleTab1": {
      "displayName": "Green"
    },
    "iconTab1": {
      "displayName": "Gray",
      "indicators": [
        {
          "condition": "conditions.tcCondition",
          "position": "left",
          "iconId": "indicatorFlagSolidGray16"
        }
      ],
      "title": "Fifa World cup",
      "extendedTooltip": {
            "view": "awTabBarActiveWhenEuro"
      }
    },
    ...
  }
}
```

The tab definition contains the following properties:

| Attribute Name | Value Type | Is Required? | Description |
|---|---|---|---|
| **displayName** | String | Optional | An **i18n** key or ctx key or tab context key or just a non-18n string which is resolved to a value which appears as the name of the tab. |
| **title** | String | Optional | An **i18n** key or ctx key or tab context key or just a non-18n string which is resolved to a value which appears as a tooltip when user hovers over the tab. |
| **description** | String | Optional | An **i18n** key assigned to this is resolved to a value which appears in a rich tooltip along with tab *title* when user hovers over the tab. Refer **Extended Tooltip** |
| **indicators** | Object | Optional | iconId which specifies the name of key referring to svg file from image directory. position which specifies the position |

| | | | |
|---|---|---|---|
| | | | of the indicator. condition which specifies the visibility of the indicator |
| **extendedTooltip** | String | Optional | It provides a way to show custom rich tooltip (styled texts, images etc.) in a balloon popup. If not configured, it shows title and 'descsription` attribute value in a rich tooltip. If configured with a custom view, it would show the contents in the custom view. Refer to **Extended Tooltip** |

## Tab Placement

A tab placement defines actual placement (location) of a tab inside application layout. A tab can have multiple placements, meaning to say, same tab can be shown/placed in more than one location using a unique placement definition within application. A tab is placed inside a **aw-tab-bar**

Let's have a look at sample **tabPlacement**:

```
{
    "tabPlacements": {
        "iconTab1Placement": {
            "id": "iconTab1",
            "uiAnchor": "iconsTabAnchor",
            "priority": 5
        }
        ...
    }
}
```

A tab placement has the following properties:

| Attribute Name | Value Type | Is Required? | Description |
|---|---|---|---|
| **id** | String | Required | The tab Id for which the placement is associated |
| **uiAnchor** | String | Required | A well known name for a given aw-tab-bar element. Anchors could be created where needed by Solutions |
| **priority** | String | Required | The relative priority order to render Tabs in ascending order |

## Tab Handlers

A tab can have many **Tab Handlers**. However at any given time, there may at most be only one *active* tabHandler for a tab. A declarative **"activeWhen"** **condition** which evaluates a boolean expression controls whether a handler is currently active. If more than one handler for a given tab evaluates to true, then the more specific condition (`the condition with longer expression`) is chosen to be active. By being active, the tab handler will determine:


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. When a tab is visible.
2. The view associated with the tab.
3. What a tab will do when executed is specified by the declarative **"action"** property for the handler.

A tab handler has the following properties:

| Attribute Name | Value Type | Is Required? | Description |
|---|---|---|---|
| **id** | String | | The tab Id for which the handler is associated |
| **view** | String | Required | The component to be rendered upon selecting the tab |
| **actions** | Object | | OnLeave key will have value which specifies the name of the declarative **action** (REST,SOA,Event or JSFunction) to be executed while switching the tab |
| **activeWhen** | Object | | The name of the declarative **condition** which will determine when this tab handler is active for the associated tab |
| **visibleWhen** | Object | | The name of the declarative condition which will determine when the associated tab is visible. This condition is only evaluated if this tab handler is active. When not defined default is true |
| **subPanelContext** | Object | | The `subPanelContext` can be accessed from the view. It can contain ctx, tab context, i18n or string values |

Here is an example of tabHandler:

```
    "tabHandlers": {
        "iconTab1Handler": {
            "id": "iconTab1",
            "activeWhen": {
                "condition": "conditions.checkSelectedTab"
            },
            "visibleWhen": {
                "condition": "conditions.checkVisibility"
            },
            "actions": {
                "onLeave": "leaveAction"
            },
            "subPanelContext": {
                "country": "{{ctx.india}}",
                "capital": "{{i18n.indiaCapital}}",
                "population": "1.45 billion",
                "currency": "{{tabContext.currency}}"
            }
        }
```

```
        ...
    }
```

## Tab Context

```
<aw-tab-bar anchor="iconsTabAnchor" tab-context="{title: 'icon1'}"></aw-tab-
bar>
```

This **tab-context** is any additional data that may be necessary to evaluate tab visibility, indicator visibility, tab displayName or fields in subPanelContext.

## Tab Selection Data

```
<aw-tab-bar anchor="iconsTabAnchor" tab-selection-
data="data.selectedTab"></aw-tab-bar>
```

This **tab-selection-data** is **Atomic data** which can be accessed from the parent or the view of the tab-bar. It contains information about the current view, tabId. It can also be used to update the tab programatically.

## Selecting the tab programmatically

This example demonstrates how a radio button can programmatically update the tabs through the use of `tab-selection-data`.

```
<aw-radiobutton prop="data.radioTab" list="data.radioTabValues.dbValue"
action="tabChange"></aw-radiobutton>
<aw-tab-bar anchor="iconsTabAnchor" tab-selection-
data="data.selectedTab"></aw-tab-bar>
```

we define an **Atomic Data** named `selectedTab` which is passed to `tab-selection-data` prop.

```
{
    "data": {
        "selectedTab": {
            "initialValues": {
                "tabId": null
            },
            "meta": {}
        },
        "radioTab": {
            "displayName": "Your favourite sport?",
            "type": "STRING",
            "isEditable": "true",
            "dbValue": "selectWhenFootball"
        },
        "radioTabValues": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "type": "STRING",
            "dbValue": [
                {
                    "propDisplayValue": "Football",
                    "propInternalValue": "selectWhenFootball"
                },
                {
                    "propDisplayValue": "Chess",
                    "propInternalValue": "selectWhenChess"
                },
                {
                    "propDisplayValue": "Cricket",
                    "propInternalValue": "selectWhenCricket"
                },
                {
                    "propDisplayValue": "Bowling",
                    "propInternalValue": "selectWhenBowling"
                }
            ]
        }
    },
    "actions": {
        "tabChange": {
            "actionType": "JSFunction",
            "method": "tabChange",
            "inputData": {
                "selectedTab": "{{fields.selectedTab}}",
                "newValue": "{{data.radioTab.dbValue}}"
            },
            "outputData": {
                "label4": ""
            },
            "deps": "js/tabBarService"
        }
    }
}
```

We define an action `tabChange` which takes in selectedTab as input. This can be called to update the tab programmatically as shown below

```
export const tabChange = ( selectedTab, newValue ) => {
    selectedTab.update( { tabId: newValue } );
};
```

## Conditions

Declarative **conditions** can be defined with the required expressions utilizing **application context**. Additional states can be added using tabContext and the **application context** service.

```
    "conditions": {
        "checkSelectedTab": {
```

```
            "expression": "ctx.search.totalFound > 0 &&
ctx.someState.inEdit"
        },
        "checkVisibility": {
            "expression": "ctx.selected"
        },
        "tabHandlerTrue": {
            "expression": "tabContext.tcIndicatorCondition === true"
        }
    }
```



Unpublished work. © 2025 Siemens

# 8. Tools

Siemens Web Framework includes a rich set of tools to support and simplify the creation of web application and debugging, as described in the following section.

## Building and testing

Every Siemens Web Framework project is a NodeJS project and comes with a set of **default NPM scripts** that can be used to perform common tasks, like building your project in production/development mode, or **running unit tests**.

## Runtime logging

**Runtime debug logging** can be enabled via an environment variable beforen building the project or at run time by setting a key on localStorage.

## Generator scripts

A **generator** script can be run to simplify the creation of common declarative constructs like modules, kits, commands and so on. Each generator script asks for a few basic questions and generates the appropriate boilerplate code automatically.

## Auditing

A set of rules have been created to be executed via an **audit script** to check declarative artifacts for common mistakes and content that does not follow best practices.

If you are using Visual Studio Code, most of these checks are performed automatically while you are editing a file and you will get instant visual feedback in case of warnings and errors.

## Google Chrome extensions

A dedicated Google Chrome extension (**Siemens Web Framework DevTools**) is available on Google Chrome store to help you debugging Siemens Web Framework applications.

Additionally, standard React and Redux extensions like **React Development Tools** and **Redux DevTools** can be used to debug Siemens Web Framework applications.

# Default NPM scripts

Siemens Web Framework provides tools for building your **Solution** to create a deployment package (known as a **site**).


Unpublished work. © 2025 Siemens

A set of default NPM script should already be included in the `scripts` section of the `package.json` file of your Siemens Web Framework project. If not, please add them as needed.

```
{
    "scripts": {
        "audit": "node node_modules/@swf/tooling/js/audit.js",
        "build": "react-scripts build",
        "clean": "rimraf out *.log *.tgz node_modules/.cache",
        "eslint": "eslint src test --ext .js",
        "generate": "node node_modules/@swf/tooling/js/generator",
        "pregulp": "node node_modules/@swf/tooling/js/createCache.js",
        "start": "react-scripts start",
        "test": "react-scripts test --watchAll=false"
    }
}
```

## audit

Run `npm run audit` to check your code for common errors related to declarative artifacts like Views and ViewModels. Although a lot of audit checks are triggered automatically when you open a file with Visual Studio Code, you may want to run this task:

- manually, to have an overall view on auditing errors and warnings affecting the entirety of your project.
- automatically, as part of a development pipeline, as a possible additional check before merging source code into your main branch.

## build

Run `npm run build` followed by the name of your site to build your project in production mode. This will trigger a WebPack-powered build process that will produce shippable artifacts in the `out/` directory of your project. Note that:

- the output of your build will be placed in **out/*site_name1***.
- the output of the last site that was built will also be symlynked to the `out/site_last` directory.

> **Tip**
>
> You can configure some **environment variables** to tweak the build process.

## clean

Run `npm run clean` to clean the cache directory created at build-time by the `pregulp` task. Remeber to do this:

- every time you add a new component to your project


Siemens Web Framework
Unpublished work. © 2025 Siemens

- every time you modify the contract (i.e. the props section) of any of your components
- as a possible **troubleshooting** step, if for some reason your changes are not being reflected in the UI at run time.

## eslint

Run `npm run eslint` to lint your JavaScript code using **ESLint**.

A dedicated set of pre-configured ESLint rules for Siemens Web Framework (which is also used by Siemens Web Framework itself) is also available in the `@swf/eslint-config-swf`. Make sure this package is available within your project and reference it in your project's `.eslintrc.js` configuration file, like this:

```
/* eslint-env node */
module.exports = {
    extends: ["@swf/eslint-config-swf"],
};
```

## generate

Run `npm run generate` to start the **generator** script, which lets you choose from a set of declarative artifacts and is able to generate all the necessary boilerplate code.

## pregulp

You should not run this script manually because it is run automatically whenever needed.

## start

Run `npm run start` to run your project in development mode. The project will be built and a development server will be started automatically (on port 3000 by default) to serve your application. Every time you modify any source file, a fast partial build will be triggered and the changes will immediately be reflected in the UI.

> **Tip**
> You can configure some **environment variables** to tweak the build process.

## test

Run `npm run test` to execute your **unit tests**. You can add the `--coverage` flag to generate test coverage reports.


Unpublished work. © 2025 Siemens

**Tip**

You can configure some **environment variables** to modify how unit tests are run.

## Environment variables

The following table lists all environment variables used by the Siemens Web Framework build toolchain (mostly when running the **NPM scripts** for building a project or starting a site).

| Variable | Development | Production | Usage | Tip |
|---|---|---|---|---|
| GENERATE_SOURCEMAP | Ignored | Used | This environment variable is used to turn on or off the source-map generation. See the dedicated section below for more information on this environment variable. | |
| SOURCE_MAP_QUALITY | Used | Ignored | Siemens Web Framework supports three kinds of Source map quality levels: HIGH, MEDIUM, LOW. See the dedicated section below for more information on this environment variable. | For this environment vairbale to work after changing its value, you need to run `aw clean` and then build the application. |
| LIVE_PREVIEW_BUILD | Used | Ignored | Default value: `false`, this env variable is very specific to UI Builder. This is only required in UI Builder when live preview is enabled. | |
| BUNDLE_VISUALIZE | Used | Used | No Default value. This setting is for advanced developers who would like to analyze the generated bundles. Set this variable to `true` and rebuild the application. End of the | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | |
|---|---|---|---|---|
| | | | build process it would automatically open the browser displaying all bundles and their configurations. | |
| MEASURE_PERFORMANCE | Used | Used | No Default Value. If set to `true`, user can see the times taken by each plugin and loaders to compile. | |
| PERFORMANCE_OUTPUT_FILENAME | Used | Used | You can set the value to `true` or `false`, based on that output file will be generated with the performance readings. | |
| WEBPACK_REBUILD_DELAY | Used | Ignored | During development build, user can set this property value so that during rebuild the process will start after that assigned time. | |
| AUTO_REFRESH | Used | Ignored | Default value is set to `true`. If set to `false`, HMR will be disabled and user needs to explicitly refresh the browser to see the changes. | |
| HIDE_CONSOLE_WARNING | Ignored | Ignored | It is used in unit tests. If set to `true`, warnings will not be displayed. | |
| FORCE_IMAGE_OPTIMIZATION | Used | Used | Defaults to `true`, it will use pre-compiled images from `@swf/images`. If set to `false`, it will compile images every time. | |
| GET_DETAIL_LOGS | Used | Used | By default the variable doesn't exists. If set to `true`, detailed logs from webpack get's generated. | |


Unpublished work. © 2025 Siemens

| | | | | |
|---|---|---|---|---|
| | | | This is helpful while debugging or analyzing build scenarios. | |
| LOG_FILE | Used | Used | By default the variable doesn't exists. If set to `true`, build logs get's written in a file at current working directory, which otherwise is displayed on terminal console. | |

### GENERATE_SOURCEMAP usage

This variable can be set to one of the following values:

- `undefined` If creating a production build, source map will be not generated. In the development build, the source map will be generated with the specified SOURCE_MAP_QUALITY value.
- `true` In both the production and development environment source map will be generated with the default SOURCE_MAP_QUALITY value.
- `false` In both the production and development environment source map will not be generated.

### SOURCE_MAP_QUALITY

- LOW Default value. This gives the best performance during the build and rebuild process. Though the source code would be different than the actual source code but easily comprehensible and can be debugged in the browser.
- MEDIUM This gives optimal performance during the build and rebuild. The source code quality is very close to the actual source code.
- HIGH This setting generates the highest quality source map The source code is like the actual source code. It increases the development build and rebuild timings.

## Configure custom production bundles

A means of achieving fewer resource chats is to leverage bundles. Bundles is a technique to amortize several resources to be loaded on demand at the same point in time. Bundles are configured in your Solution's kit.json in the `bundles` section.

Each logical bundle is a named JSON object. The Siemens Web Framework tooling will build your site so that the JS files you list will be combined into the named bundle file which will be loaded on demand. The


Siemens Web Framework
Unpublished work. © 2025 Siemens

goal is to find the balance point between the two undesirable extremes of either downloading everything at start up (bad for start-up performance and browser memory consumption) and creating too many small bundles resulting in many network chats (bad for introducing latency costs).

It is possible to exclude modules from a bundle. Specifying an empty string indicates which Javascript modules not to bundle.

Bundles are used for amortizing Javascript. CSS and JSON configuration artifacts are concatenated by the Siemens Web Framework site building.

A bundle can have the following properties:

- `name`
  - The name of the pre-existing file that the bundle will be written to.
  - Don't provide this field if creating a new output bundle file.
- `deps`
  - This is a list of module names. The format should make how you'd reference these modules from within the code.
- `include`
  - This is a list of modules to include in this bundle.

## Example `bundles` definition in a Solution kit.json

```
    "bundles": {
        "lib/noty/jquery.noty.js": {
        },
        "js/bootstrap.js": {
            "include": [
                "assert",
                "js/aw-gwt-presenter.directive",
                "js/aw.checkAuthentication.controller",
                "js/eventBus",
                "js/iconService.js",
                "js/leavePlace.service",
                "js/localStorage",
                "js/locationNavigation.service",
                "js/ngUtils",
                "js/NotyModule",
                "js/sessionManager.service",
                "js/TcAWServerVersion",
                "js/TcSessionData",
                "js/ui.states",
                "js/viewModelObjectService",
                "soa/module",
                "js/soa/kernel/schema"
            ]
        },
```

```
        "js/tcUserPwAuthenticator.js": {
            "deps": [
                "js/bootstrap"
            ],
            "include": [
                "js/TcSessionData"
            ]
        },
        "js/new.bundle.js": {
            "deps": [
                "js/bootstrap"
            ],
            "include": [
                "js/someExisting.js"
            ]
        },
        "js/new2.bundle.js": {
            "deps": [
                "js/new.bundle"
            ],
            "include": [
                "js/someExisting2.js"
            ]
        }
    }
```

## How to test

- `set DRAFT=false` (environment variable), you'll notice that assets will be placed in a directory which ends with a timestamp
- Build your site
- Run your product
- Run Browser Dev Tool & monitor the network tab

**Note**

From `v4.0.0` onwards, the `exclude`, `optimize`, `allowSourceOverwrites`, `preserveLicenseComments`, and `generateSourceMaps` keywords are no longer supported.

## Bundle definitions are merged from different kit.json

There's no need to repeat the exclusion bundle or bootstrap which are defined in Siemens Web Framework kit.json. However, if you want to add to these, you can add entries from your kit.json.

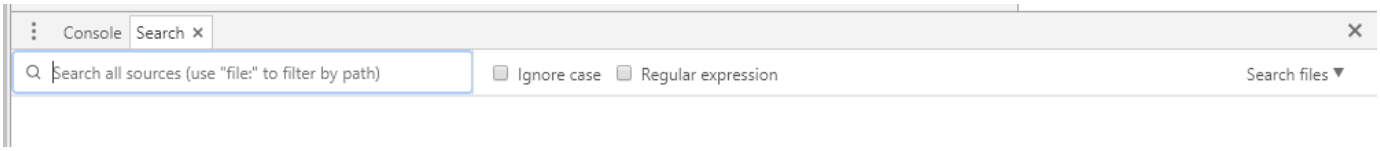
```
{
    "bundles": {
        "js/bootstrap.js": {
            "include": [
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "js/myJavaScript"
            ]
        }
    }
}
```

## How to determine what JavaScript files are loaded at run time?

Chrome DevTool (F12) / Search all files (Ctrl+Shift+F) / "js/myJavaScriptFile"




This will allow you to limit the file search to only the files loaded by the DevTool. This is useful when you're trying to understand what's pulling in a particular JavaScript during the page load.

## Default bundling strategy in SWF

By default following types of bundles are created.

- Bundles for deps defined in all ViewModels of individual modules.
- Bundles for deps defined in Configuration files like Decorators.json/States.json
- Bundles for remaining ViewModel/JS files from individual modules.

## How to enable readable bundle names for build

Set below enviorment variable

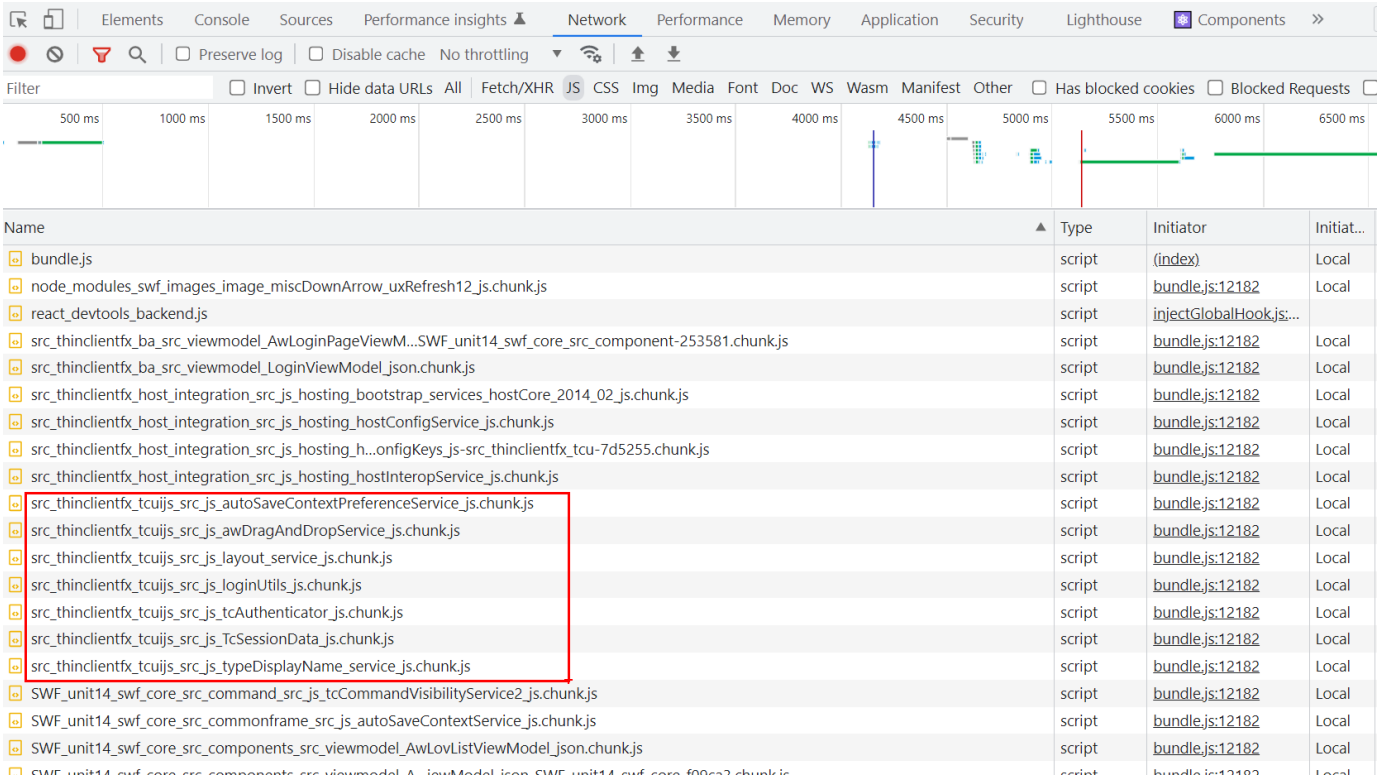
```
set BUNDLING_CHUNK_IDS = named
```

and then execute below commands.

- for Production build : `aw build && express`
- for Development build : `aw start` [Note : chunking is disabled in dev mode, so you will see individual files getting requested]

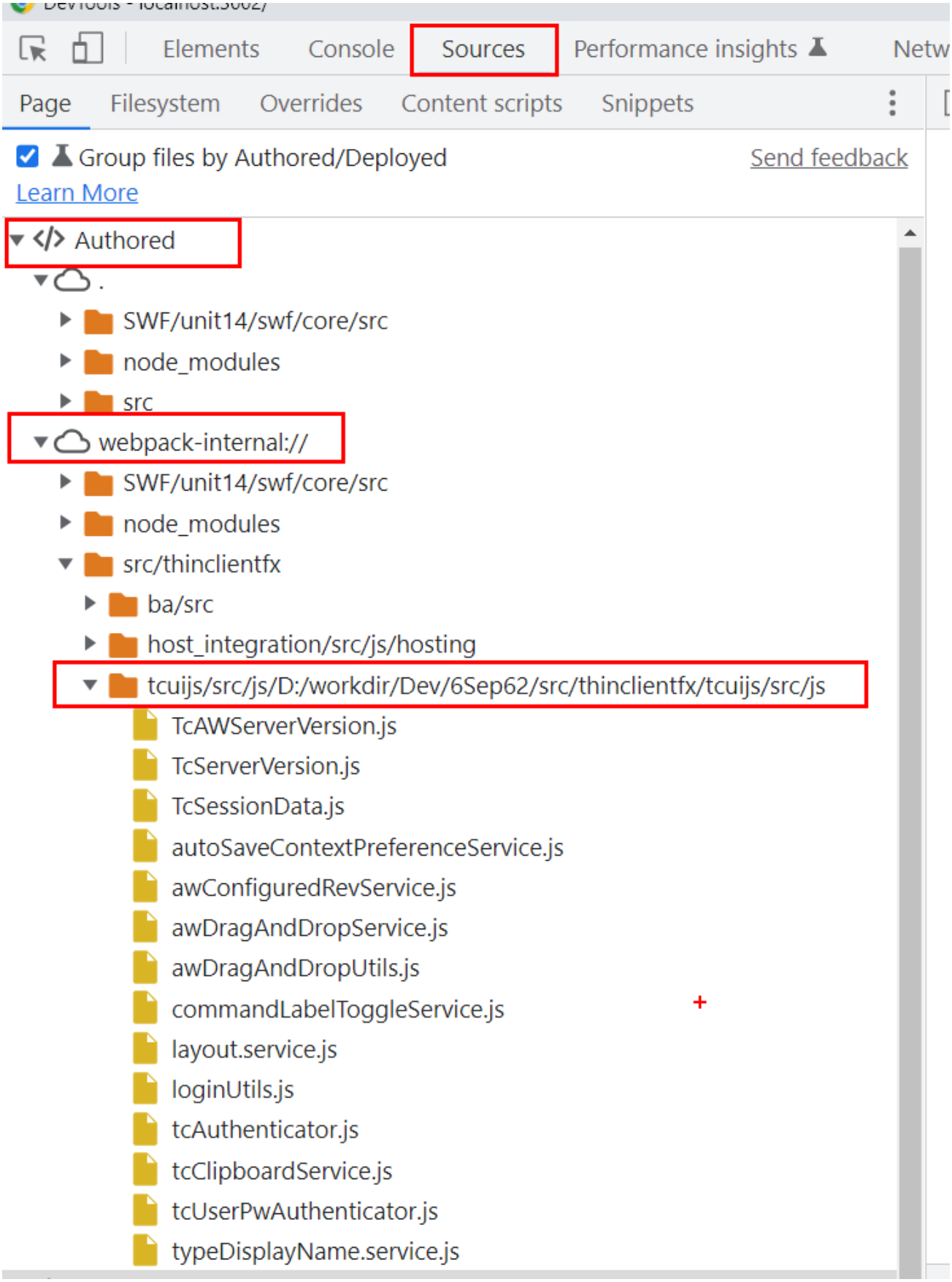
## How to identify files to create Custom bundles using kit.json for specific page

Launch the application using Development build [ ``` aw start``` ] and then look out for specific files loaded in the Network Tab for specific page.




The complete list of files loaded from specific module can be found in the Sources tab.

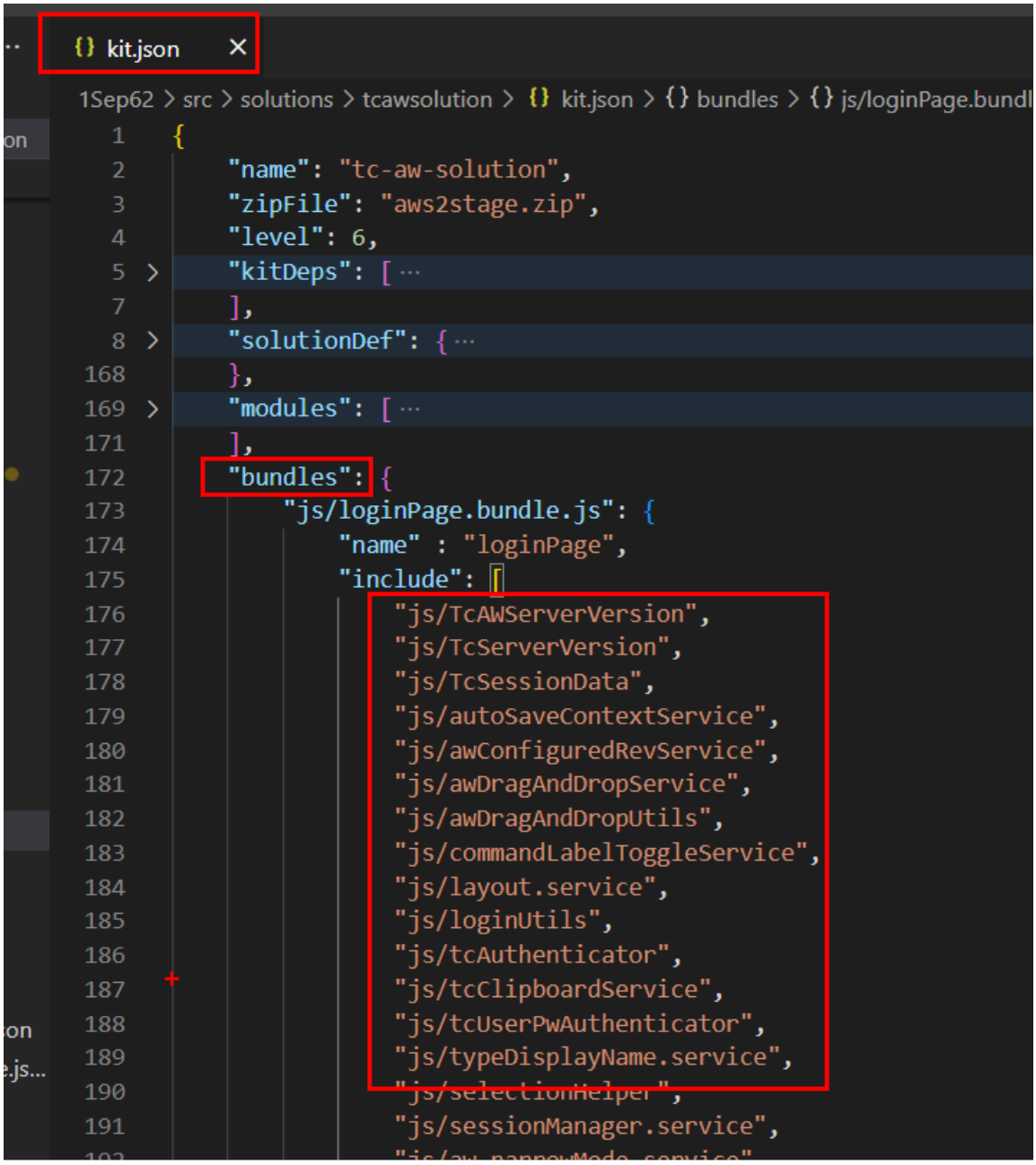

Siemens Web Framework
Unpublished work. © 2025 Siemens





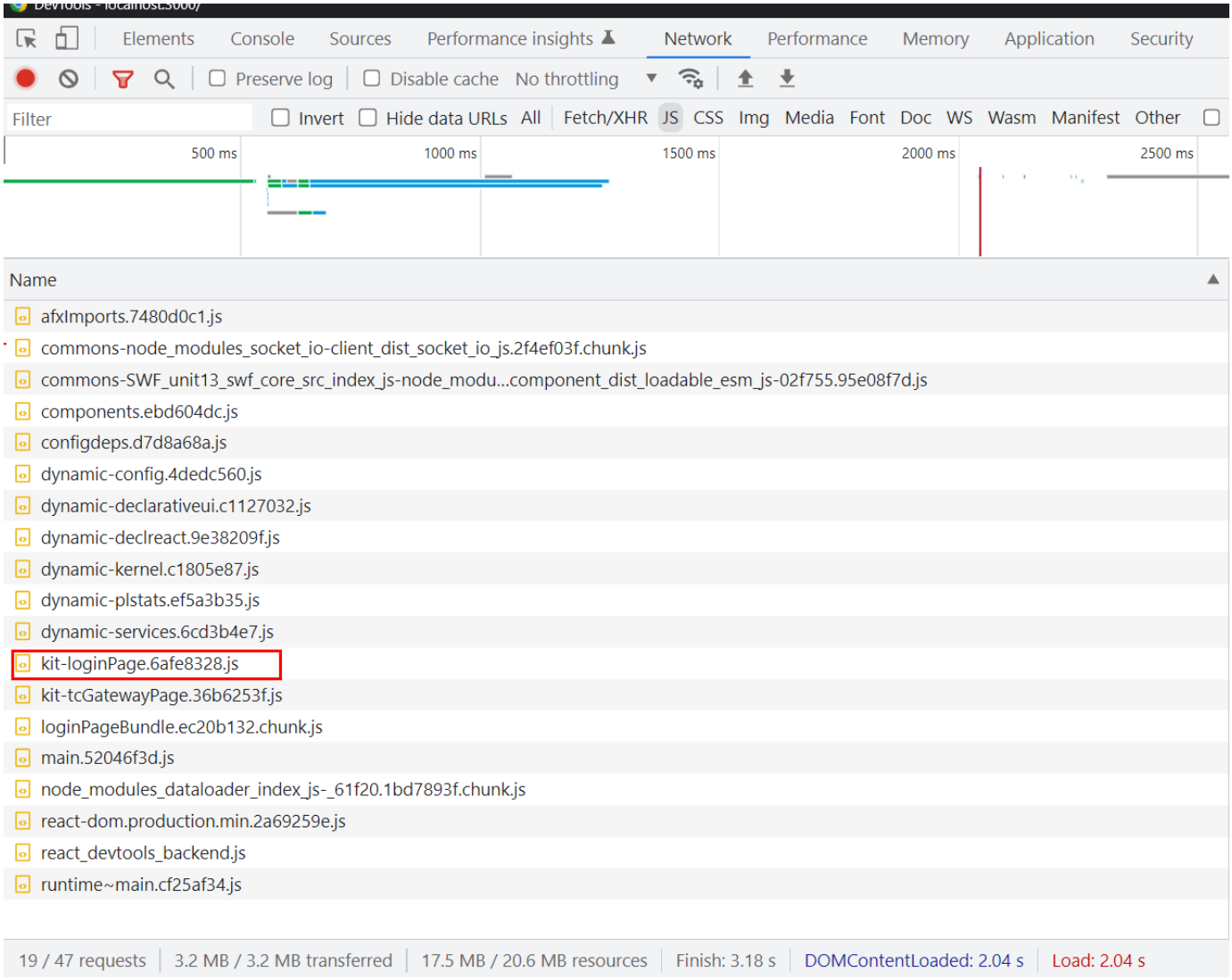
Unpublished work. © 2025 Siemens

Define these file names in kit.json in bundles section with correct syntax. This will generate separate chunk for these files.




Now, run the application using Production build [ ``` aw build && express``` ] and see loginPage bundle is created and loaded.


Unpublished work. © 2025 Siemens




By using this approach you can strategize your bundling approach.

Also check new mechanism of identifying files to create custom bundles **How to identigy files**

# Configure custom production bundles using bundles.json

As of version 7.1 of Siemens Web Framework, you can also configure bundling for your application using a separate `bundles.json` file as an alternative updating the kit.json files to **configure custom productions bundles**.

> **Note**
> In future releases, this will become default approach to customize production bundles.

## Enabling bundling based on bundle.json

To configure production bundles:

1. Place a `bundles.json` in the root directory of your application (at the same level as the `build.json` file).
2. set the `USE_CUSTOM_BUNDLE_CONFIGURATION` environment variable to `true` before running the `aw build` command. When this variable is set, the build will automatically pick up the `bundles.json` file. In absence of this variable, build will fack back to `kit.json` configuration.

## Configuring your bundles

The `bundles.json` file supports both image and modules (view model and JavaScript code) bundle customization.

A bundles.json file can have following properties:

- `images` A collection of image bundles. Each bundle must have a unique name and it can have different images listed as an array.
- `modules` A collection of module bundles.Each bundle must have a unique name and it can have different module names listed as an array.

## Example of bundles.json file

```
{
    "modules": {
        "bundle1_js": [
            "js/browserUtils",
            "js/logger",
            "js/breakpointService"
        ],
        "bundle1_vm": [
            "viewmodel/UiViewViewModel",
            "viewmodel/AwIconViewModel",
            "viewmodel/AwButtonViewModel"
        ]
    },
    "images": {
        "bundle1_img": [
            "image/miscDownArrow_uxRefresh12",
            "image/miscInProcessIndicator16",
            "image/miscExpandedTree12",
            "image/miscCollapsedTree12"
        ],
        "bundle2_img": [
            "image/cmdSearch16",
            "image/miscSiemensLogo24",
            "image/indicatorChipNotTrackingChanges24",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "image/miscSortedDescending10"
            ]
        }
    }
```

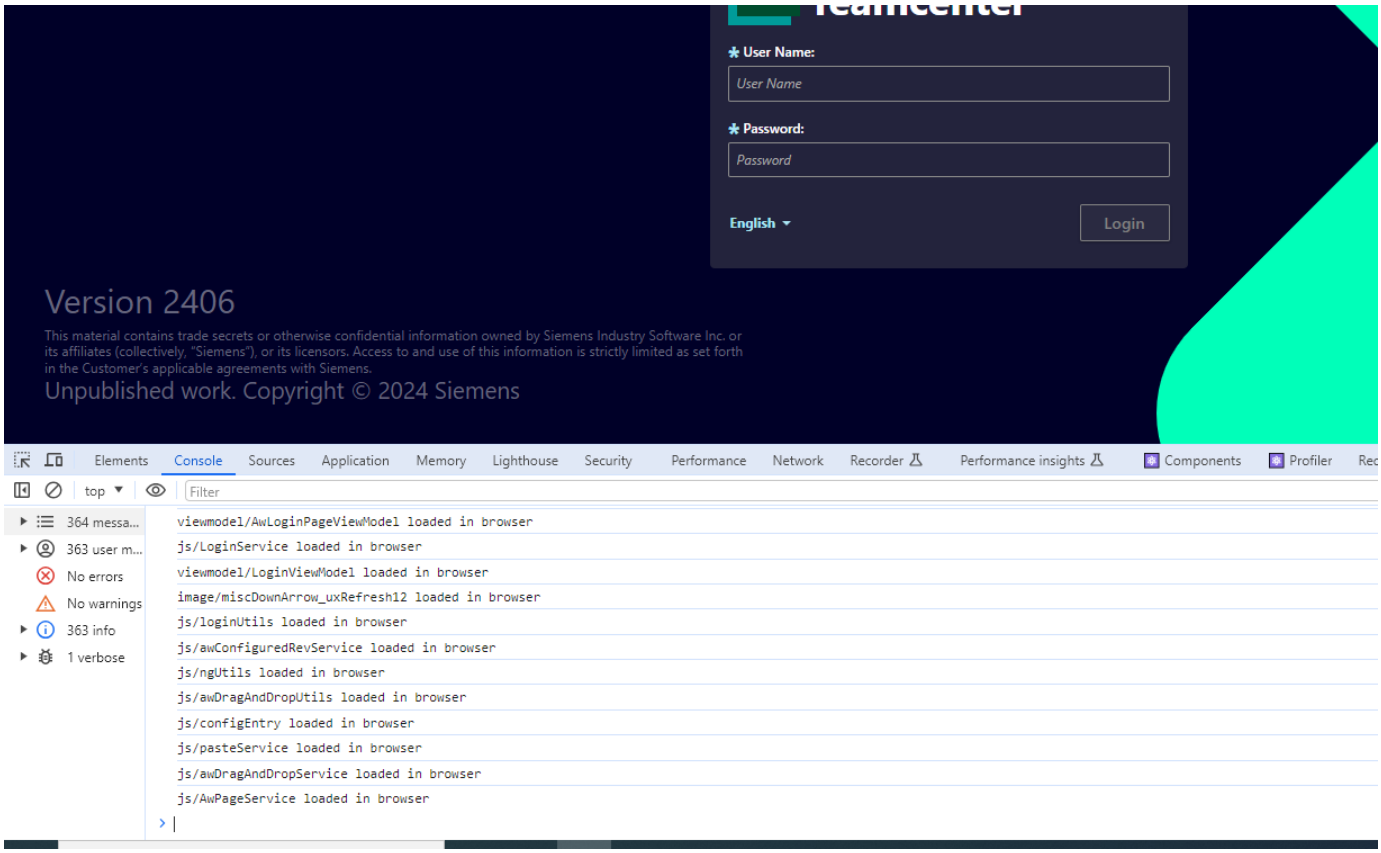
## How to identify files to create custom bundles

To enable this functionality set the following environmental variable.

```
SET PRINT_FILE_LOAD_DATA=true
```

Once environment variable is set, run `aw build` command and open the application in your browser.

The file information will be dumpled in browser console, like the following:




# Using the Command Line Interface

Siemens Web Framework CLI is a command-line interface to manage and automate Siemens Web Framework projects. Its aim is to make development with Siemens Web Framework faster and more productive by:

- reducing the amount of boilerplate code that users have to write by generating as much code as possible
- providing ways to audit code and limit human errors by guiding developers with simple prompts when creating new artifacts
- providing a compatibility layer for project-wide operations like building, testing, etc. instead of relying on having the correct scripts in your package.json file
- providing ways to automate virtually any operation to perform on an Siemens Web Framework project
- automating migrations and code refactoring

After installing the package, you will be able to use the `aw` command from any command prompt. Executing just `aw` without any argument or option will display the built-in help:

```
aw - The Siemens Web Framework and Active Workspace Command Line Interface.

VERSION
  @swf/cli/7.0.0-0 win32-x64 node-v20.5.1

USAGE
  $ aw [COMMAND]

COMMANDS
  add         Add declarative artifacts.
  audit       Audit the current project.
  build       Build the current solution.
  clean       Cleanup temporary files generated at build-time.
  doc         Search for a specific string in documentation.
  help        Display help for aw.
  info        Display information on the specified artifact.
  init        Initialize a new project in the current (empty) directory.
  migrate     Perform a code migration or automated refactoring on the
whole project.
  quickstart  Initialize a new project with a sample page, download NPM
packages, and start the development server.
  report      Generate reports.
  search      Search for declarative artifacts containing the specified
string.
  set         Set/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.
  update      Update this CLI to the latest version.
  upgrade     Upgrade the current project to a newer version of Siemens Web
Framework.
```

For more information on a specific command, run `aw help` followed by the name of a command.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Tip**

Some users reported that the executable name aw conflicts with another executable (aw.bat) which is installed by Siemens NX. If you run into this or a similar naming conflict issue, you can launch Siemens Web Framework CLI also using any of the following aliases: `awcli`, `swf`, `swfcli`.

## Implementing custom commands

The Siemens Web Framework CLI can also be extended with custom commands, that can be implemented in any Siemens Web Framework project.

To implement a custom command at project level, you just need to add a file containing the implementation of your command under a `cli/commands` folder in your project. Note that this file must have a `.mjs` extension.

As an example, create a `cli/commands/hello.mjs` file containing the following code which will simply print `Hello, World` when executed:

```
import { SwfCommand } from '@swf/cli/lib/swf-command.js'

export default class Hello extends SwfCommand {

  static description = 'Just a test command to say hello...';

  async run() {
    this.log('Hello, World!');
  }
}
```

When the `aw` command is executed from your project, this command will be loaded automatically from the `cli/commands` folder and it will be listed among the available commands (the command ID is determined by the file name):

```
aw - The Siemens Web Framework and Active Workspace Command Line Interface.

VERSION
  @swf/cli/7.0.0-0 win32-x64 node-v20.5.1

USAGE
  $ aw [COMMAND]

COMMANDS
  add          Add declarative artifacts.
  audit        Audit the current project.
  build        Build the current solution.
  clean        Cleanup temporary files generated at build-time.
  doc          Search for a specific string in documentation.
  help         Display help for aw.
  info         Display information on the specified artifact.
```


Unpublished work. © 2025 Siemens

```
  hello       Just a test command to say hello...
  init        Initialize a new project in the current (empty) directory.
  migrate     Perform a code migration or automated refactoring on the
whole project.
  quickstart  Initialize a new project with a sample page, download NPM
packages, and start the development server.
  report      Generate reports.
  search      Search for declarative artifacts containing the specified
string.
  set         Set/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.
  update      Update this CLI to the latest version.
  upgrade     Upgrade the current project to a newer version of Siemens Web
Framework.
```

**Tip**

You can also override existing commands at project level in this way. Note that if you override a command that provides subcommands, such as `aw add`, your custom command will take precedence over the existing out-of-the-box command, and you will not be able to execute any of the existing subcommands, such as `aw add command`.

## Using code generators

Siemens Web Framework provides a command-line interface tool that can be used to generate important artifacts of the declarative framework. It is a way to quickly bootstrap a customization.

The command `aw init` provides all the starter files for an SWF application, while `aw quickstart` installs required dependencies and spawns a development server with some starter code. This command is also used in the **Your First Hello World Application tutorial**. It generates a solution, module, default authenticator, clipboard, and states.

In addition to these creator scripts, artifacts can be added using the `aw add` script.

| Artifact Type | Description |
|---|---|
| authenticator | Generates an authenticator. |
| clipboard | Generates a clipboard. |
| command | Generates a zero compile command. |
| component | Generates a new component (view and view model). |
| i18n | Generates a new i18n entry to the i18n section of a view model. |
| kit | Generates a kit.json. A kit.json is a combination of modules included in the site build. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | |
|---|---|
| location | Generates a new location route (state). |
| module | Generates a module. A module is a build configuration that determines which code to include and how to process it. |
| property | Generates a new ViewModelProperty to the data section of a view model. |
| route | Generates a new route (state). |
| solution | Generates a solution. |
| sublocation | Generates a new sub-location route (state). |
| theme | Generates a theme. A theme can change the colors used in your site. The default themes are Light and Dark. |
| workspace | Generates a workspace. |

## Command

You can access this tool by running the following command:

```
aw add <artifact_name>
```

**Tip**

Ensure that SWF CLI is installed globally. Refer to **this** page.

Let's create your first web application using the Siemens Web Framework CLI tool.

## Initialize a new SWF project

Skip this step if you are working on an existing SWF project.

1. In an empty directory, run:

   ```
   aw init
   ```

2. When prompted to enter the name of the package, type `swfCliTest` and press Enter.

   ```
   Specify the name of your project/package: swfCliTest
   ```

3. When prompted to choose if you wish to install SWF packages from a local folder, choose Yes and press Enter.

   ```
   Do you want to install SWF packages from a local folder? Yes
   ```

4. When prompted to enter the path of local SWF packages, enter the absolute/relative path and press Enter.

```
Specify the path to the folder containing .tgz SWF packages:
D:\work\swf\cli\test\data\tgz
```

5. When prompted to choose if you wish to install packages, choose Yes and press Enter.

```
Install packages: Yes
```

The project is initialized successfully.

### Add a solution

Skip this step if you are working on an existing SWF project.

1. In the initialized project, run:

```
aw add solution
```

2. When prompted to enter the company name, type and press Enter.

```
Specify the company name: Siemens DI SW
```

3. When prompted to enter the copyright text, type and press Enter.

```
Specify the copyright text: Copyright © 2024 Siemens Industry Software
```

4. When prompted to specify the main module, type the name and press Enter.

```
Specify the name of the main module of the solution: swfCliTestMain
```

A solution will be added to the project.

### Add a module

Use the CLI to create a new module.

1. Start by running the command:

```
aw add module
```


Unpublished work. © 2025 Siemens

2. When prompted for a module name, enter `helloWorld` and press Enter.

```
Specify the name of your module: helloWorld
```

   Note that a default kit is picked up, which you can change using `aw set kit`.

The module helloWorld has been added to your kit.

### Add a location

Use the CLI to create a new state in the helloWorld module.

1. Start by running the command:

```
aw add location
```

   Notice that the newly created module was picked up. You can set the default module using `aw set module`.

2. When prompted for the name of the location route, enter `hello` and press Enter.

```
Specify the name of your location route: hello
```

3. When prompted for the page title at the route, enter `Hello` and press Enter.

```
Specify the title to display at the top of the location page: Hello
```

4. When prompted for the page sub-title, enter `Test Page` and press Enter.

```
Specify the sub-title to display in the browser tab, after the name of
your application: Test Page
```

A location will be created in the module's states.json. Now proceed to create a component and add it to the view of this created location.

### Add a component

Use the CLI to create a new component.

1. Run the command:

```
aw add component
```

2. When prompted for the name of the component, enter `HelloComp` and press Enter.

```
Specify the name of your component: HelloComp
```

3. Replace the "view" defined in your states.json from the previous step with `HelloComp`. You can also choose to add some tags to the `HelloCompView.html` file.

You have created HelloCompViewModel.json, HelloCompView.html, and added a new state hello in the module helloWorld.

You can access your new state at **http://localhost:3000/#/hello** after rebuilding using the following commands:

```
```shell
    aw build site_showcase
    aw start
    ```
```

## Add a command

1. Run the command:

```
aw add command
```

2. When prompted for the name of the command, enter "openCmd" and press Enter:

```
Specify the name of your command: openCmd
```

3. When prompted for the name of the icon, select "cmdZoom" and press Enter:

```
Specify the name of your command icon: cmdZoom
```

4. When prompted for the title of the command, enter "Open Command" and press Enter:

```
Specify the title of your command: Open Command
```

5. When prompted for the type of the command, select "simple" and press Enter:

```
Specify the type of command: simple
```

6. When prompted to choose if the command is contained in a group command, enter "No" and press Enter:

```
Is this command contained in a group command?: No
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

7. When prompted for the placement of the command slot, select "toolbarRightSlot" and press Enter:

```
Specify the anchor identifying the placement of your command slot:
toolbarRightSlot
```

8. When prompted to choose the command action, select "JSFunction" and press Enter:

```
Specify the type of the action executed by the command: JSFunction
```

9. When prompted to choose the presentation style of a group command, select "menu" and press Enter:

```
Specify the presentation style of a group command: menu
```

This will create a simple JS command openCmd. If you check the commandsViewModel, openCmd, its handler, placements, and action are added to it.

```
{
   ...
   "commands": {
       ...
       "openCmd": {
           "iconId": "cmdZoom",
           "title": "Open Command"
       }
       ...
   },
   ...
   "commandHandlers": {
       ...
       "openCmdHandler": {
           "id": "openCmd",
           "activeWhen": {
               "condition": "conditions.true"
           },
           "visibleWhen": {
               "condition": "conditions.true"
           },
           "action": "openCmd_1726469137593"
       }
       ...
   },
   ...
   "commandPlacements": {
       "openCmdPlacement": {
           "id": "openCmd",
           "priority": 10,
           "uiAnchor": "toolbarRightSlot",
           "displayType": "menu"
       }
```

```
    },
    ...
    "actions": {
        ...
        "openCmd_1726469137593": {
            "actionType": "JSFunction",
            "method": "info",
            "inputData": {
                "message": "Action called by command \"Open Command\"."
            },
            "deps": "js/logger"
        }
        ...
    }
    ...
}
```

### Add a theme

1. Run the command:

```
aw add theme
```

2. When prompted for a theme name, enter the theme name and press Enter. This will be used as the title of your theme and scss file.

```
Specify the name of your theme: yellow
```

3. When prompted for a base color, enter the hex code of the color and press Enter. This will be used as the primary color of your theme with different shades as secondary colors.

```
Specify the base color of your theme in RGB format: #FFFF00
```

A theme named yellow is created in the module.

## Implementing unit tests

As of Siemens Web Framework version 5, you can use **Jest** to implement unit tests for your Siemens Web Framework code.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Running unit tests

Place your unit tests in a `test` folder inside your modules, and then run them by executing the following command from the root of your project:

```
aw test <path/to/module/folder>/test
```

## Basic template (synchronous code)

You can use the following code as a basic template to test synchronous code:

```
/* eslint-env jest */

import xyz from 'js/xyz'

/**
 * Test for 'xyz'
 * @module test/xyz.test
 */
describe( 'Testing something important', () => {
    beforeEach( function() {
        // setup something for all the tests in this suite
    } );

    test( 'Doing something interesting', () => {
        // test something
    } );
} );
```

Note that a `describe` function:

- can have many `it` functions.
- can have many `beforeEach` functions.

## Basic template (asynchronous code)

You can use the following code as a basic template to test asynchronous code:

```
/* eslint-env jest */

import xyz from 'js/xyz'
import { waitFor } from '@testing-library/react';
import eventBus from 'js/eventBus';

/**
 * Test for 'xyz'
 * @module test/xyz.test
```


Unpublished work. © 2025 Siemens

```
 */
describe( 'Testing something important', () => {
    // Preferred approach
    test( 'Doing something interesting', async () => {
        // Pattern #1: Simple
        const abc = await callAsyncFunction();
        expect( 'check something about abc' );

        // Pattern #2: Working with subscriptions (no promise or rejection)
        await new Promise( ( resolve, reject ) => {
            xyz.subscribe( 'eventName', () => {
                resolve();
            } );
            // do something to cause the event to be triggered
            callWithoutPromiseOrCallback();
        } );
        expect( 'check something' );

        // Pattern #3
        // When async code doesn't have promise or callback, you can use
waitFor.
        // Note, this waits 1 second by default & then continues. You can
control the wait.
        eventBus.publish( 'eventName', { data: 'eventData' } );
        await waitFor( () => {
            expect( 'check async stuff happeneded' );
        } );
    } );

    // Legacy approach requiring a 'done' callback function
    test( 'Doing else something interesting', done => {
        // test something
        callAsyncFunction().then( abc => {
           expect( 'check something abc' );
           done();
        } ).catch( err => {
          // didn't work
        } )
    } );
} );
```

## Frequently-asked questions

The following sections list some frequently-asked questions concerning unit testing a Siemens Web Framework project with Jest.

## How can I override the return value of a function?

```
jest.spyOn( module, 'functionName' ).mockReturnValue( 'hello' );
jest.spyOn( module, 'functionName' ).mockReturnValueOnce( 'hello' );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

## How can I override a function?

```
jest.spyOn( module, 'functionName' ).mockImplementation( xyz => {
    return `${xyz}!!!`;
} );
```

## How can I override the result of a Promise?

```
jest.spyOn( module, 'functionName' ).mockResolvedValue( 'hello' );
jest.spyOn( module, 'functionName' ).mockRejectedValue( new
Error( 'Ugh' ) );
```

## How can I check expected values?

```
expect(object).toBe({});
```

For more information, see **Jest documentation**.

## How can I mock a component that is reused by my component?

```
jest.mock('viewmodel/AwCommandListViewModel', () => ({
 __esModule: true,
 default: () => 'AwCommandList'
}));
```

## How can I test changes that happen in an `onMount` hook?

Mock anything async to ensure it resolves immediately and then you can do something like this:

```
const { rerender } = render( <AwCommandBar2 anchor={'myanchor'}/> );
await act( async () => {} );
rerender( <AwCommandBar2 anchor={'myanchor'}/> );
```

## How can I test code that depends on the application context?

If your component depends on the application context you must use a fake context in unit testing, otherwise, your unit test will be impacted by all other unit tests that are running.

The `renderWithCtx` function has been added to simplify this process. The following code renders the `AwCommandList` component with a context that only has the `flag` property:

```
import { renderWithCtx as render } from '@swf/core/test/testUtils';

render( <AwCommandList commandList={ AwCommandListTestData.baseCommandMultiple
Handlers }></AwCommandList>, {
    initialState: {
     flag: true
 }
} );
```

To ensure that your component responds to context changes correctly, you can manually create a **Redux** store and dispatch changes to it, as follows:

```
const store = createStore( _dispatchCtx, {
    flag: true,
    flag2: true
} );
render( <AwCommandList
commandList={ AwCommandListTestData.commandsCheckingSeparateFlags }></
AwCommandList>, { store } );
act( () => {
    store.dispatch( { type: 'update', path: 'flag2', value: false } );
} );
```

## How can I work with mocks?

Mocks must happen outside of any `describe` block. This is required because the mock must be setup before the `import` happens.

```
// Outside your describe block
jest.mock( 'js/localeService' );
```

You should mock dependencies completely (as needed) and then setup a spy for only the specific methods that will be called.

For more information, see the **Jest documentation**.

## What can I do if tests return an error like `process.exit called with 1`?

This occurs when your test runs into a promise reject that is not handled. To debug this, set a breakpoint in the `unhandledRejection` block of the `jest.setup.js` file.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### `setTimeout` doesn't seem to work properly, what can I do?

If you are facing issues with setTimeout not working, then remove `jest.useFakeTimers()` in your `beforeEach` block, if present.

### Useful links

- **Jest Getting started**
- **How to disable tests/suites**
- **Module mocking in Jest**
- **Mocking functions and modules with Jest**

## Running audit checks

Siemens Web Framework provides an audit functionality to validate the declarative **view** and **view model** artifacts along with a few JavaScript validations during development to detect errors much like a compiler can.

Furthermore, custom linting checks are available to analyze the code statically during CI and inside Visual Studio Code.

The audit check flags the errors and anti-patterns for the following:

- The declarative view (HTML) syntax and the **UI element** usage.
- The declarative view model (JSON) against its schema and further logical verifications.
- JavaScript files

### Prerequisites

Audit checks rely on **ESLint** to work and on a dedicated set of rules specific to Siemens Web Framework that are provided in the **@swf/eslint-config-swf** package.

Make sure that this package is available in your project and reference it in your ESLint configuration file, like this:

```
/* eslint-env node */
module.exports = {
    extends: [ '@swf/eslint-config-swf' ]
};
```


Unpublished work. © 2025 Siemens

## Running audit checks from the command line

To run audit checks from the command line, run the following command in the root of your project:

```
aw audit
```

This command will run all audit checks on all project files.

## Enabling real-time auditing in Visual Studio Code

You can get real-time feedback by enabling real-time auditing in Visual Studio Code. To enable this functionality, proceed as follows:

1. Download and install the **ESLint extension** for Visual Studio Code.
2. Add the below configuration in your Visual Studio Code **settings.json** file.

```
{
    "eslint.validate": [ "javascript", "json", "html" ]
}
```

3. Restart Visual Studio Code.

When you open a view, view model, or JavaScript file, audit checks will be executed and you will get real-time feedback anti-pattern errors or warnings.

## Adding Custom Audit Rules

You can easily create your own Audit rules and scope them across files, by leveraging functionality provided by ESLint (see the **ESLint documentation** on custom rules).

First, add an `.eslintrc.json` file in your project if not already present. This file is where you define the project-level ESLint configuration. Now, note that this file can be stored in any directory to override the configuration defined in the parent declaration. The rules are then overriden for the directory and its consequent files and subdirectories.

Then, create a directory called `rules` (it can be named anything) and define the JS file of the rule.

Here's the structure of a typical rule file:

```
module.exports = {
    meta: {
        // Metadata about the rule
        type: "problem", // or "suggestion", "layout"
        docs: {
            description: "Description of the rule",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            category: "Best Practices" // or "Stylistic Issues", "Possible
Errors"
            // Add more documentation as needed
        },
        schema: [/* Add schema definition if the rule supports options */],
        messages: {
            // Define error messages for this rule
            // You may have multiple messages for different scenarios
            // E.g., "Expected ... but found ..."
        }
    },

    create(context) {
        // Return an object with visitor methods
        // These methods define what the rule does when traversing the AST
        // See: https://eslint.org/docs/developer-guide/working-with-
rules#visitor-pattern
        return {
            // Visitor methods for different types of AST nodes
            // E.g., Identifier, VariableDeclaration, FunctionDeclaration,
etc.
            // Inside these methods, you can perform your rule logic
            // You have access to the context object, which provides
information about the code being linted
            // and allows you to report errors or warnings using
context.report()
        };
    }
};
```

Now, this structure suffices if the rule does not depend of any other file or metadata of the file. But, many a times for some rule enforcements, like say a rule that would check if a particular js dependency you're importing really exists; or if you wish to introduce a whitelist and make use of it (there is a utility object that reads from the default whitelist - staticCodeAnalysis_dev.json file), in this case, your rules need access to these metadata. Siemens Web Framework provides some utilities to help retrieving such metadata. To access them, add the following to your code:

```
    /* Previous Code */
    /* In create function: */
        const cacheAccess = context.settings.cache // this variable now
has the object which carries key-values of all utility functions that are
defined through SWF.

        console.log(Object.keys(cacheAccess)) // this is optional and it
would give you a list of all the utility functions we supply

    /* Following Code */
```

Then these utility objects can be used in your code for detecting any violations.

Now that the logic of your rule is ready, you can declare and scope this rule in our `.eslintrc.js` file.

In your object definition, add the following code:

```
module.exports = {
    // Your existing ESLint configurations
    // Add any other rules or configurations you need

    overrides: [
        {
            files: ["your/file/pattern/*.js"], // Adjust the file pattern
as needed
            rules: {
                // Add your custom rule within this override block
                // You can follow the same format as before
                "<relativePathOfYourRule>": "error"
            }
        },
        // Add more overrides if you need to apply the rule to additional
files
    ]
};
```

Again, see the **ESLint documentation** for additional information.

Finally, if you wish to integrate this rule with audit, you should also ensure that your audit.js script scope also has the files you wish ESLint to look at.

# Configuring JSON IntelliSense and validation

Each of the JSON declarative artifacts (ViewModel, commandsViewModel) has a JSON schema file associated that is used to describe and validate the structure and the content of JSON data. The schema defines the elements, attributes and data types.

Siemens Web Framework build system has audit tasks to check validity of declarative artifacts. Additionally you can configure Visual Studio Code settings to get immediate feedback while authoring the declarative artifacts.

## Enabling JSON IntelliSense

IntelliSense is a general term for a variety of code editing features including: code completion, parameter info, quick info, and member lists. IntelliSense features are sometimes called by other names such as "code completion", "content assist", and "code hinting." For more details **refer**.

You can configure Visual Studio Code to automatically validate Siemens Web Framework JSON files based on their schema. Apart from validation, it is also possible to display descriptions for common properties and provide suggestions for allowed properties in a certain context.

To enable JSON IntelliSense i Visual Studio Code, proceed as follows:


Siemens Web Framework
Unpublished work. © 2025 Siemens

1. Open Visual Studio Code in the root directory of your Siemens Web Framework project.
2. From Visual Studo Code menu, select **File > Preferences > Settings**.
3. In the **Settings** page, select **Workspace** and search for **JSON**.
4. Look for the **JSON:Schemas** setting and click **Edit in settings.json**. This will create a new **settings.json** file on your machine if not available already.
5. Specify Siemens Web Framework JSON schema files as follows:

```
{
    "json.schemas": [
        {
            "fileMatch": [
                "/commandsViewModel.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeCommandsViewModelSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/viewmodel/*ViewModel.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeViewModelSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/kit.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeKitSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/module.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeModuleSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/workspace.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeWorkspaceSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/workspace_*.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeWorkspaceContributionSchema-1.0.0.json"
        },
        {
```


Unpublished work. © 2025 Siemens

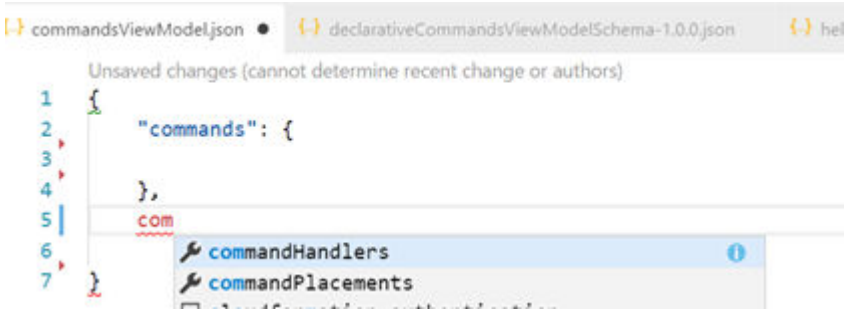
```
            "fileMatch": [
                "/site_*.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeSiteSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/build.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeBuildSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/decorators.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeDecoratorsSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/delta.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeDeltaSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/aliasregistry.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeAliasRegistrySchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/dragAndDrop.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeDragAndDropSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/paste.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativePasteSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/propertyRendererTemplates.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativePropertyRendererTemplatesSchema-1.0.0.json"
        },
        {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "fileMatch": [
                "/secondaryWorkareaTabs.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeSecondaryWorkAreaTabsSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/states.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeStatesSchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/syncStrategy.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeSyncStrategySchema-1.0.0.json"
        },
        {
            "fileMatch": [
                "/tabsViewModel.json"
            ],
            "url": "./node_modules/@swf/tooling/conf/
declarativeTabsViewModelSchema-1.0.0.json"
        }
    ]
}
```

## Result

For properties and values you get suggestions as you type with IntelliSense. You can also manually see suggestions with the **Trigger Suggestions** command (**Ctrl+Space**). It also performs structural and value verification:




When you hover green underline you get the helping messages.

When you hover over properties and values, you will be provided additional context information.

## Enabling runtime debug logs

Siemens Web Framework uses the **debug** library, which provides extensive and powerful logging capabilities.

Logging can be enabled at development time so that the built application will always print debug logs, or at run time, directly in the browser.

To enable *all* debug logs at development time:

1. Set the `DEBUG` environment to `\*`:

   ```
   set DEBUG=*
   ```

2. Build your project.


Unpublished work. © 2025 Siemens

**Tip**

To disable logs, execute the following command to unset the `DEBUG` variable:

```
set DEBUG=
```

To enable *all* debug logs at run time:

1. In your browser, open the developer tools by pressing `F12`.
2. In the `Console`, enter the following text and press `ENTER`:

```
localStorage.debug = '*';
```

**Tip**

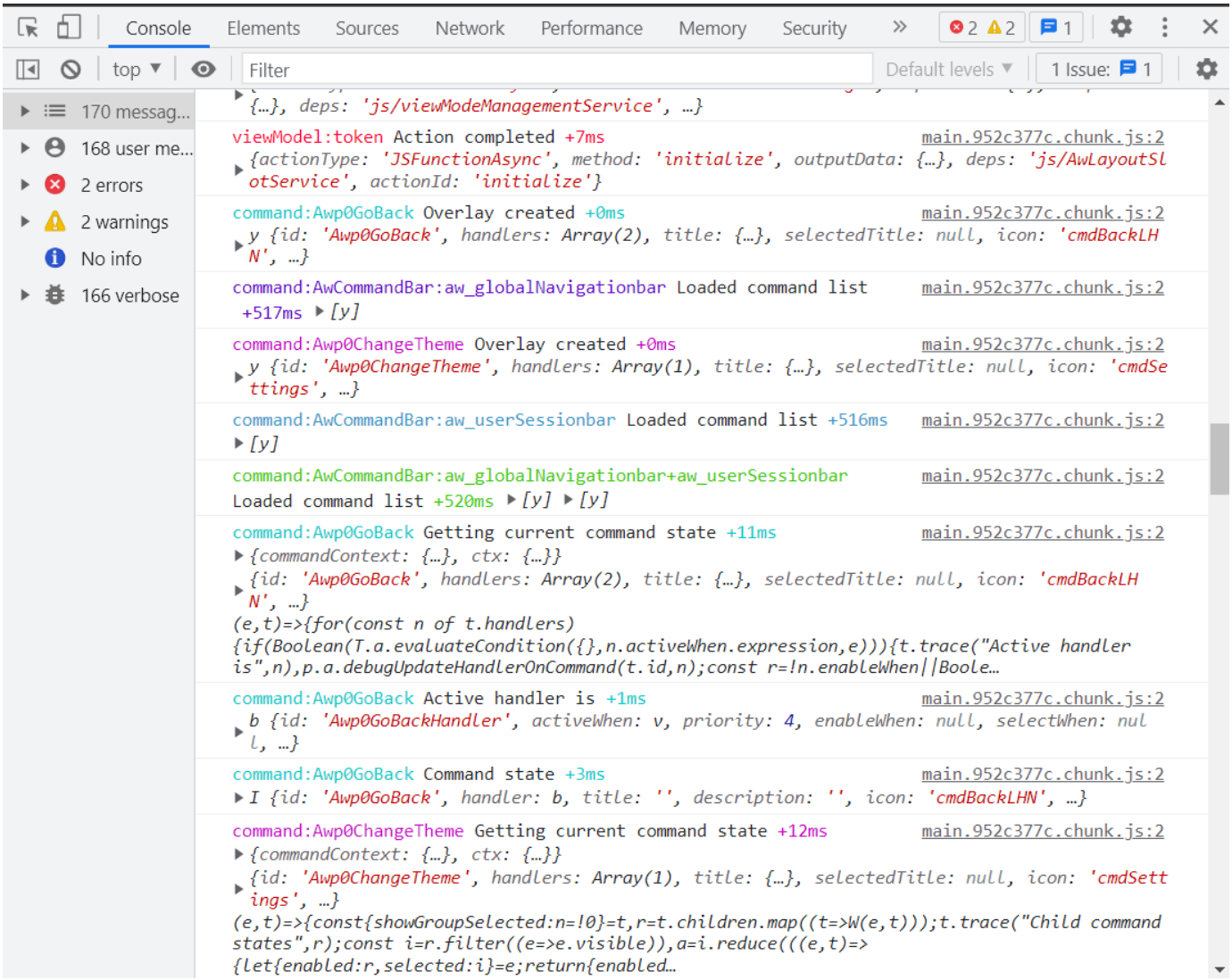
To disable logs, enter the following text in the `Console` and press `ENTER`:

```
localStorage.debug = null;
```

## Example of debugging output

The following screenshot shows an example of console output when debug logs are enabled:


Unpublished work. © 2025 Siemens




## Available debug logs

Setting the value of the debug variable to * enables *all* debug logs, which can be quite verbose. Instead, you can set the variable to a more restrictive value to selectively enable only certain logs.

The following table lists all possible log settings that can be enabled.

| Value | Effect |
|---|---|
| command:* | Enable all logging related to commands. |
| command:commandConfigurationService | Enable only generic logging related to commands. |
| command:tcCommandVisibilityService | Enable logging related to the command visibility service. |


Unpublished work. © 2025 Siemens

| | |
|---|---|
| command:commandConfigurationService:* | Enable logging for all activity related to specific command changes. |
| command:commandConfigurationService:Awp0Checkout | Enable logging only related to the `Awp0Checkout` command (you can specify any command ID). |
| async | Enable logging related to `asyncService`. |
| bootstrap | Enable logging of bootstrap-related messages. |
| contributionService | Enable logging related to `contributionService`. |
| exportService | Enable logging related to `exportService`. |
| ocLazyLoad | Enable logging related to lazy loading. |
| soaService | Enable logging related to `soaService` (Active Workspace specific). |
| splmAnalyticsService | Enable logging related to `splmAnalyticsService`. |
| viewModel:* | Enable logging related to view models. |
| viewModel:token | Enable logging related to updates to the view model token. |
| viewModel:conditions:<component_name> | Enable logging related to conditions defined in component's ViewModel |
| Banner | Enable logging related to Global Banner. |
| expressions | Enable logging related to expressions defined in exist-when, visible-when, aw-class, batch action, onEvent or in any section of the components view/viewModel |
| selection | Enable logging related to selection model updates over component hierarchy |

## Siemens Web Framework DevTools

*Siemens Web Framework DevTools* (formerly-known as *Declarative Chrome Extension*) is an extension for the Google Chrome browser that helps the user to debug the data flow between different actions, events, data providers etc. It also provides advanced breakpoint capabilities and lets you add breakpoints into actions, commands, routes, and more.

## Installation

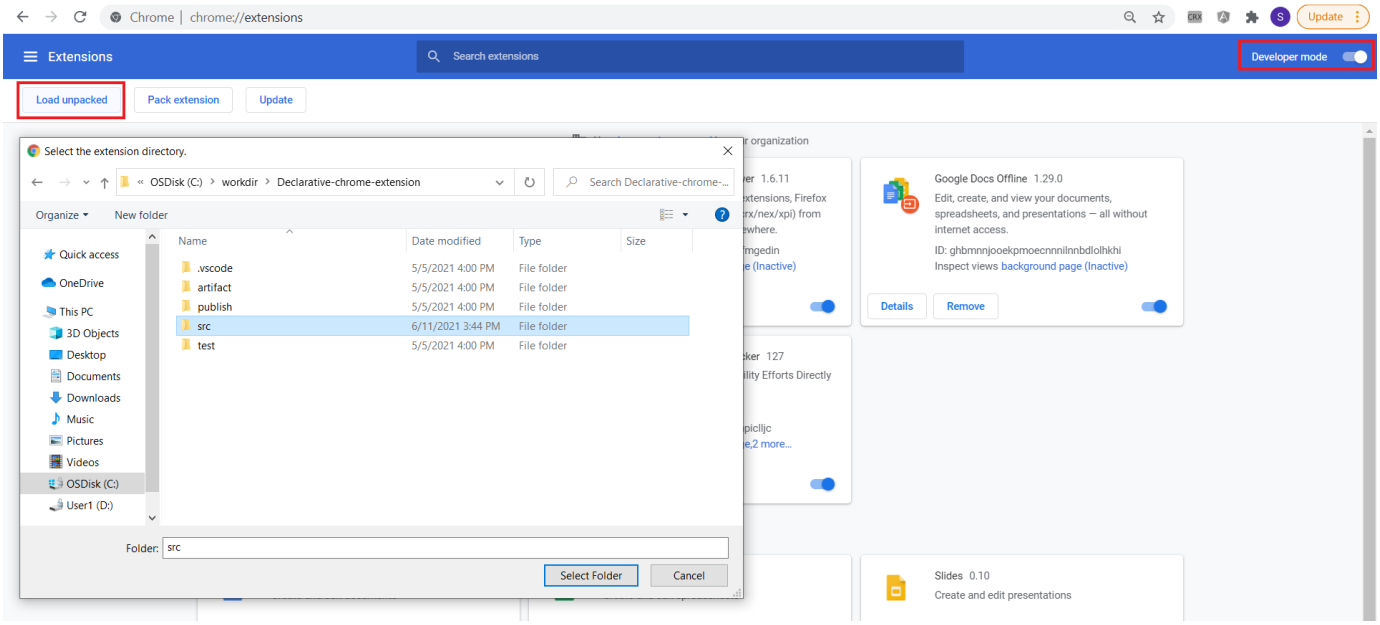
Different versions of the extension are compatible with different versions of Siemens Web Framework, as listed in the following table:

| Siemens Web Framework Version | Extension Version |
|---|---|
| prior to Siemens Web Framework 3.3 | v1.2 |
| from Siemens Web Framework 3.3 to 4.0 | v2.1 |
| from Siemens Web Framework 4.1 to Siemens Web Framework 4.x | v2.2 |
| from Siemens Web Framework 5 onwards | v3.2 |

The latest version of the extension can always be installed from Google Web Store.

For installing older versions, follow these steps after downloading the corresponding zip file:

1. Extract the contents of zip file provided.
2. Browse to the *src* folder in the `Declarative-chrome-extension\` folder to extract the extension.
3. Open Chrome Browser and navigate to `chrome://extensions` and enable **Developer Mode**, on right side.
4. Click on **Load unpacked** and provide the above path (to the `src` folder) to install the extension.



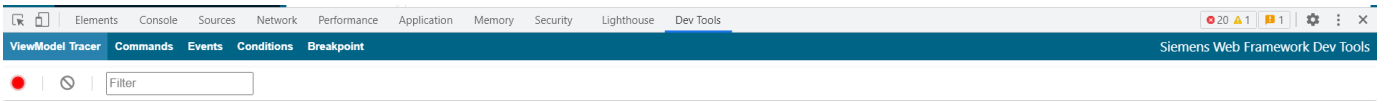

After this, you should be able to use the extension.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Logging Capabilities

As Siemens Web Framework is a declarative framework, we need some mechanism to track the data which is flowing around the system.

Siemens Web Framework' DevTools is divided into multiple sections:

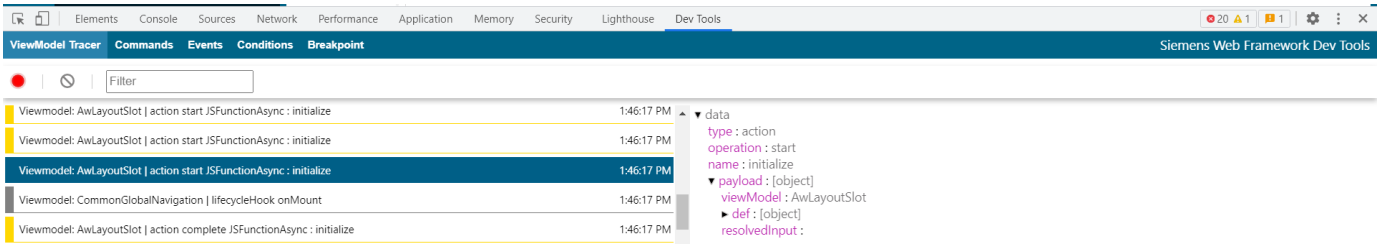



Each section logs the data corresponding to its genre. Below are the sections available.

## ViewModel Tracer

This section logs all the activities i.e. actions, events with event-data around all the view Models currently active i.e.

- Actions both input and output.
- Events with event data.
- DataProvider – input to the data provider (temporarily disabled)
- Lifecycle hooks - For init, mount, update, unmount




Following is the format each activity is presented into the ViewModel tracer:

```
ViewModel: $<viewmodelName> | action|dataprovider|event:  $<action|event|
dataprovider> name
```

The activities are logged with the time they have occurred. The user can select a particular logged action or event and on the right side of the page details of that particular action is logged i.e. event-data or action input.

Siemens Web Framework DevTools provides a powerful filtering capability, which can be used to filter out specific information you are interested in:

- you can type in a view model name in the filter box to filter only the activities around that viewmodel.

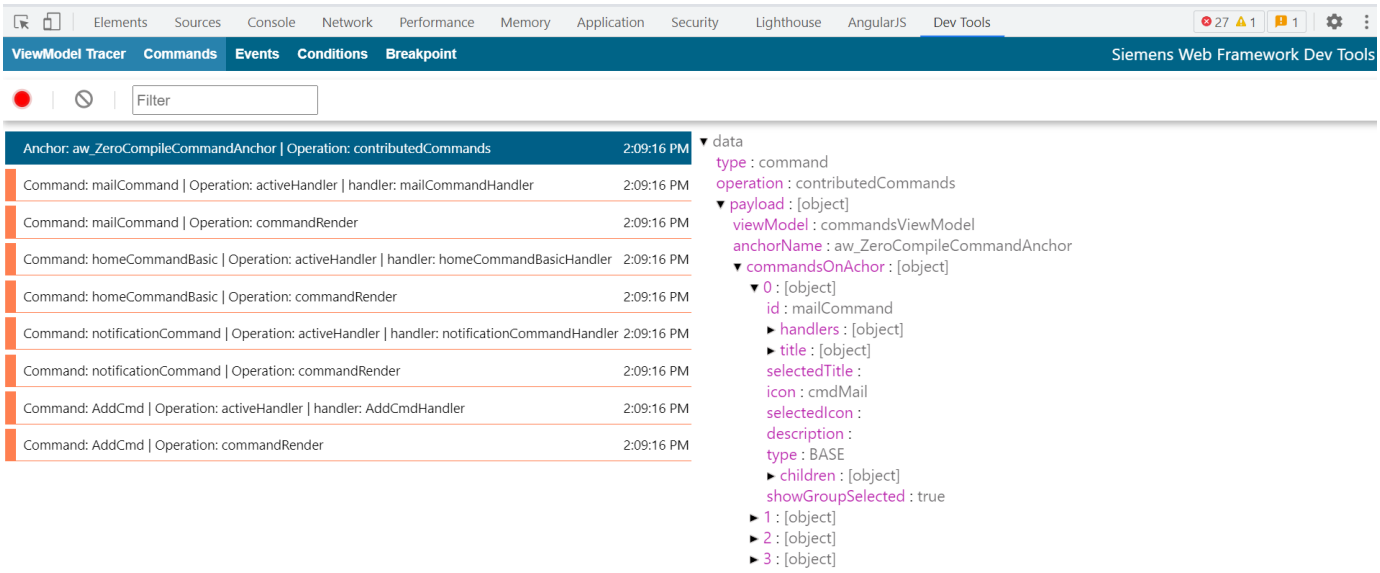

Unpublished work. © 2025 Siemens

- you can type in comma "," view model name in the same box to filter only the activities around those view models.
- you can type any keyword which might be available in any of activity payload.

## Commands

This section logs information about commands and command bar. If user wants to log some information about commands, he/she should open this tab and refresh the page where the command or command bar is location. Command/command bar related information is logged only when page is refreshed.

- All the anchor names with all the commands contributed to it.
- All the commands when they are rendered/re-rendered.
- Each command with their command handler.




Following is format of the information logged in 'Commands' section.

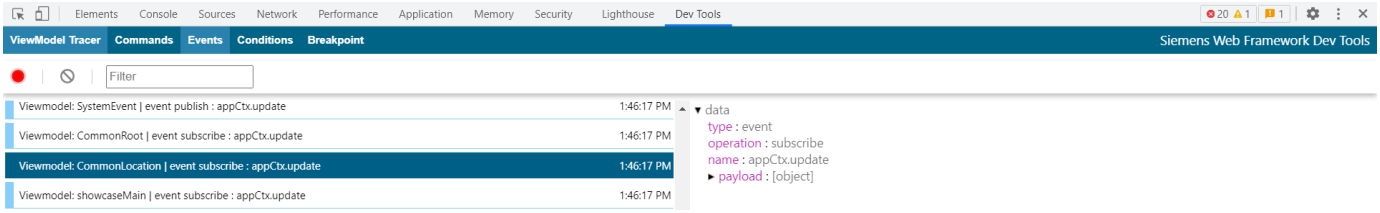
```
Anchor|Command: $<name> | Operation: <activeCommands|contributedcommands|
activeHandler>
```

This section also provides the same filtering capability as mentioned on 'view model tracer' section.

## Events

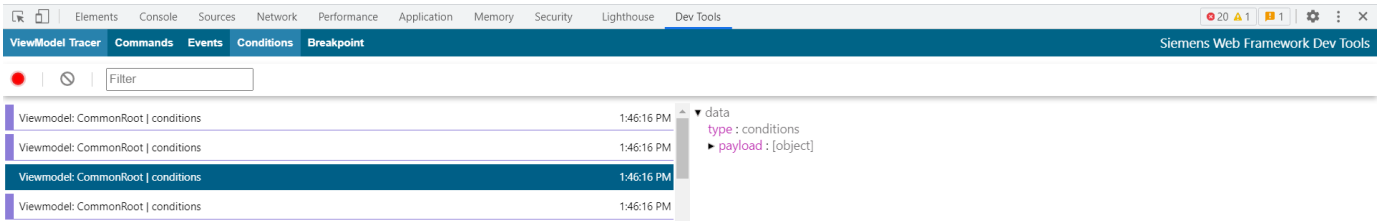
In Siemens Web Framework we heavily use global 'eventBus' to communicate across widgets, services. This event section logs all the events fired across all the widgets or services in the application.


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Conditions

This tab has currently limited capability. It tries to log all the conditions which are evaluated by the system at run time. It does not currently show the condition name associated with the condition expression.

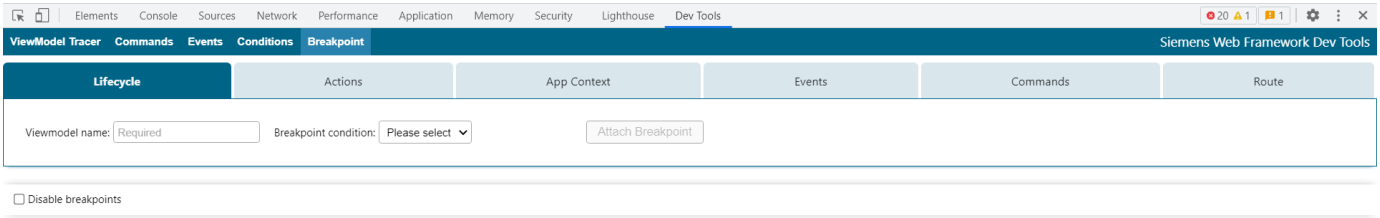



## Breakpoint

The last tab of dev tool is breakpoint tab. It provides user the capability to add breakpoint into some very key declarative areas of the code without knowing much about the declarative engine.

The areas are as following:

- Lifecycle
- Actions
- Application Context
- Events
- Commands
- Routes




**Warning**

Breakpoints are lost when the browser is refreshed. If the browser is refreshed for some reason, make sure to restart the chrome DevTools or Press F12 twice.

## LifeCycle

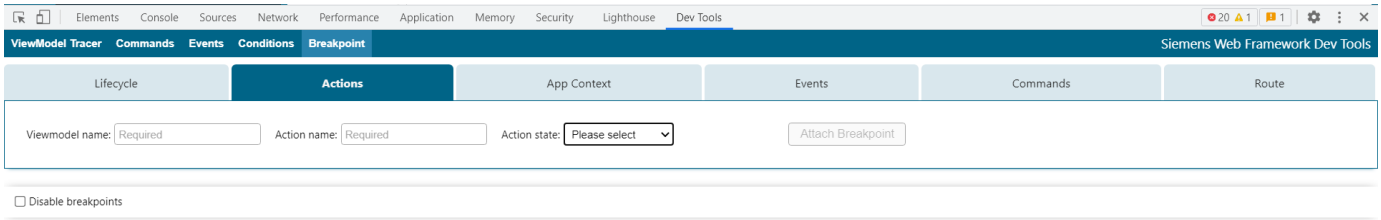
This allows user to add breakpoints at specific stages of view model life cycle. User need to enter the following

- viewmodel name: Declarative ID of a view/viewmodel.
- Breakpoint Condition:
  - Initialize: When the given component is initialized.
  - Mount: When the given component is initialized and mounted into DOM.
  - Update: When the given component is updated.
  - Unmount: When the component is unmounted.

## Actions

This allows user to add breakpoint at specific stages of an action of a viewmodel. User need to enter the following

- viewmodel name: Declarative ID of a view/viewmodel.
- Action Name: Name of the action user wish to debug.
- Action State
  - Pre: The breakpoint will hold when the input data of the action is resolved but the action is yet to fire.
  - Post: The breakpoint will hold after the action is executed and 'outputData' of the action is resolved.
  - Events: If the action has some event associated then the breakpoint would hold just before the events fires.
  - Action Messages: If the action has some action message associated then the breakpoint would hold just before the action messages are resolved.
-




## Application Context

This allow user to debug the change of global state. User need to enter the following

- key: The name of key which is registered/yet to register in global state and user want to monitor it.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- State
  - register: When the key is registered in the global state.
  - unregister: when the key is unregistered from the global state.
  - modify: when the value of the key is modified.

## Events

This allow user to debug a certain event with its event data. Using this user can add a breakpoint at the subscription end of a event in a certain viewmodel.

User need to enter the following:

- viewmodel name: Declarative ID of a view/viewmodel.
- Event name: Name of the event.

**Note**
ViewModel name is not mandatory in case of a global event.

## Commands

This allow user to add a breakpoint for a certain command. Once the command is clicked the breakpoint would break once user clicks on that certain command.

## Routes

This allow user to track the any state change across the application. User need to enter the following:

- From state: User need to enter the state name. whenever user leaves that state the breakpoint would break.
- To State: This is an optional parameter. User need to enter a state name and whenever user enters that state, the breakpoint would break.

## Investigating common problems

Without using the DevTools, a developer has no way to know if the output data section has been resolved or not.

DevTools help you see the resolved output data of an action once the action is resolved.

In particular, the DevTools are able to:

- log the data passed into a data parser


Unpublished work. © 2025 Siemens

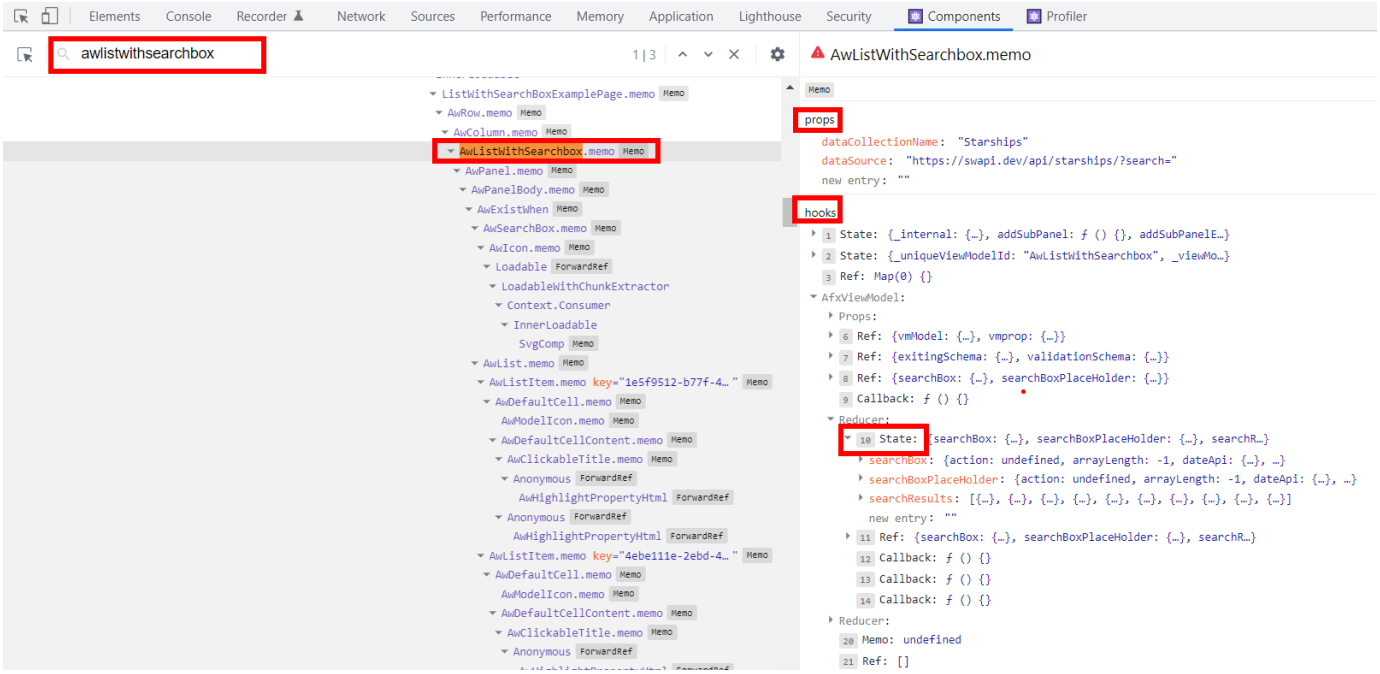
- log the commands contributed against a specific command bar
- log the currently-active commands picked up by the system against a specific command bar
- log the current command handler for a specific command

## React Development Tools extension for Google Chrome

There is a Chrome Extension for helping inspect and debug React applications called **React Developer Tools**. This extension will save you time and help your developer productivity if you become proficient in its use. Here are some links to tutorials and videos to help get you started with it:

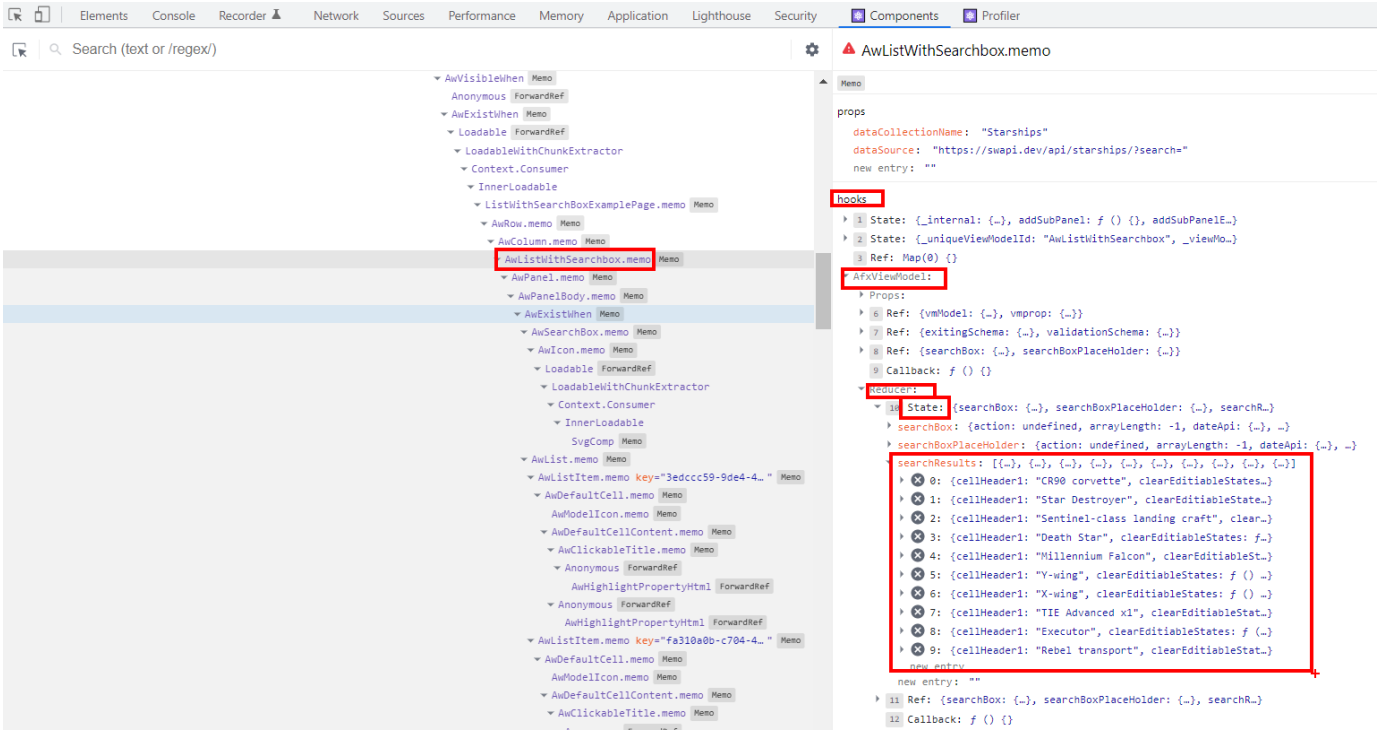
- **Chrome Web Store: React Developer Tools**
- **Tutorial: How To Debug React Components Using React Developer Tools**
- **Pluralsight Guide: Debugging Components with React Developer Tools**
- **egghead.io video: Debug React Components with Developer Tools in Chrome**
- **React Developer Tools video: - Installing and Basic Usage**

Once the extension is installed, the **Components** tab becomes available. The components are shown in the component hierarchy/tree and can be expanded/collapsed. Furthermore, the name of the component that you are interested in inspecting can be searched, and standing on that component. You can inspect its props as well as data (local component state) in the same tool. This is shown below:

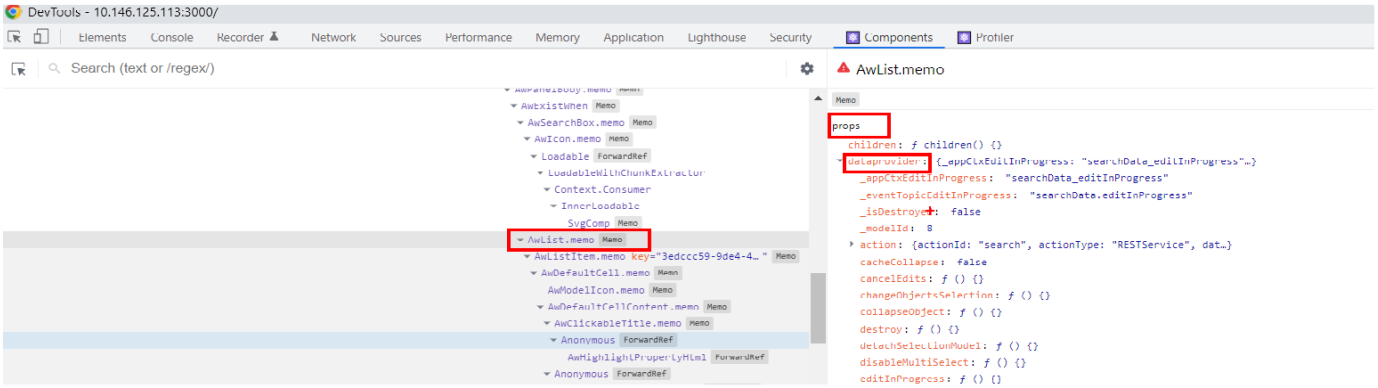



The below image shows the local state of the component:

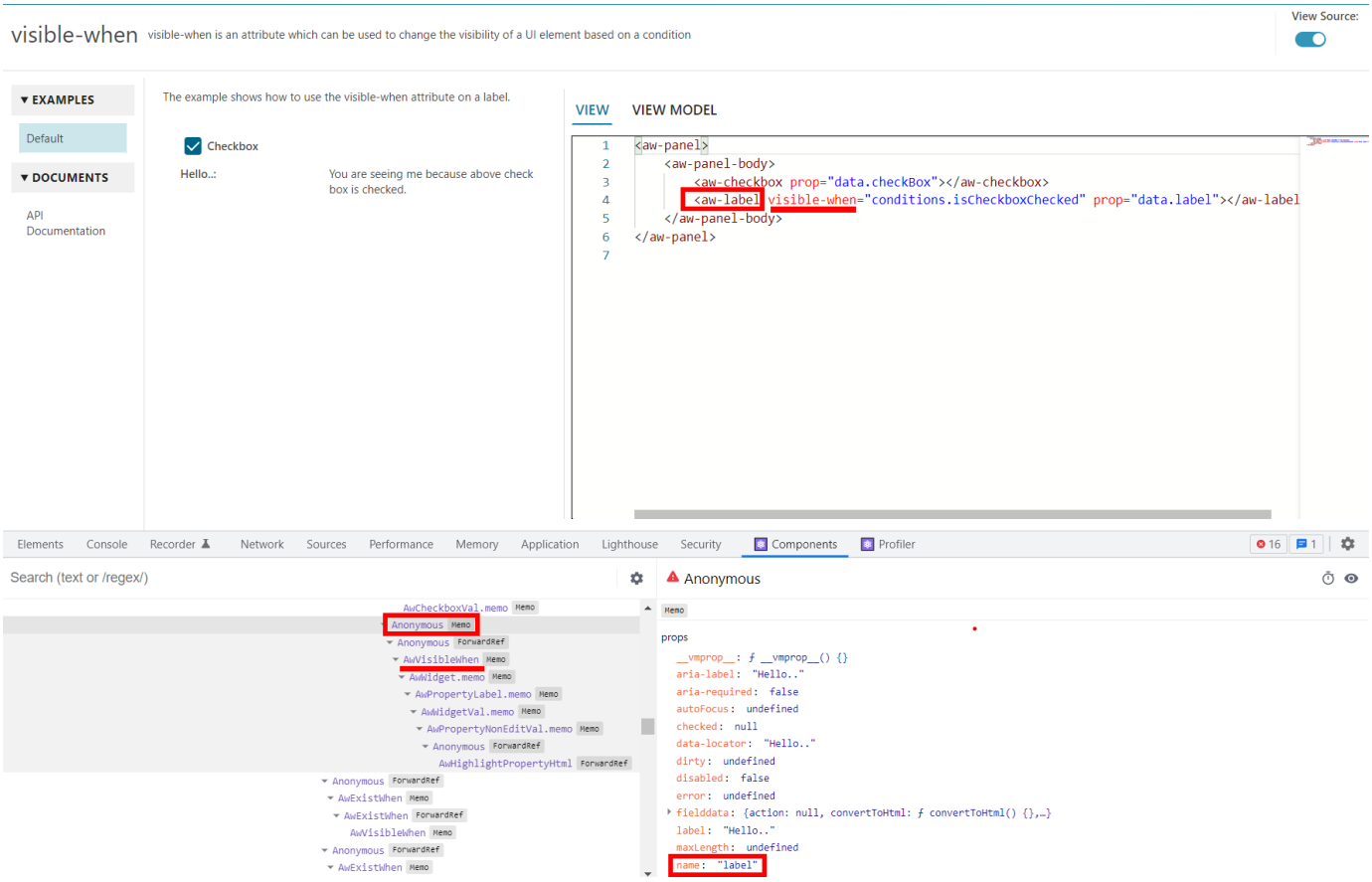

Siemens Web Framework
Unpublished work. © 2025 Siemens




The below image shows the props received by the component:




The attribute level directives in Angular like exist-when. visible-when are implemented as High Order Components, and hence they may show with a different name, and the main component would show up with a name as **Anonymous** like in this case:



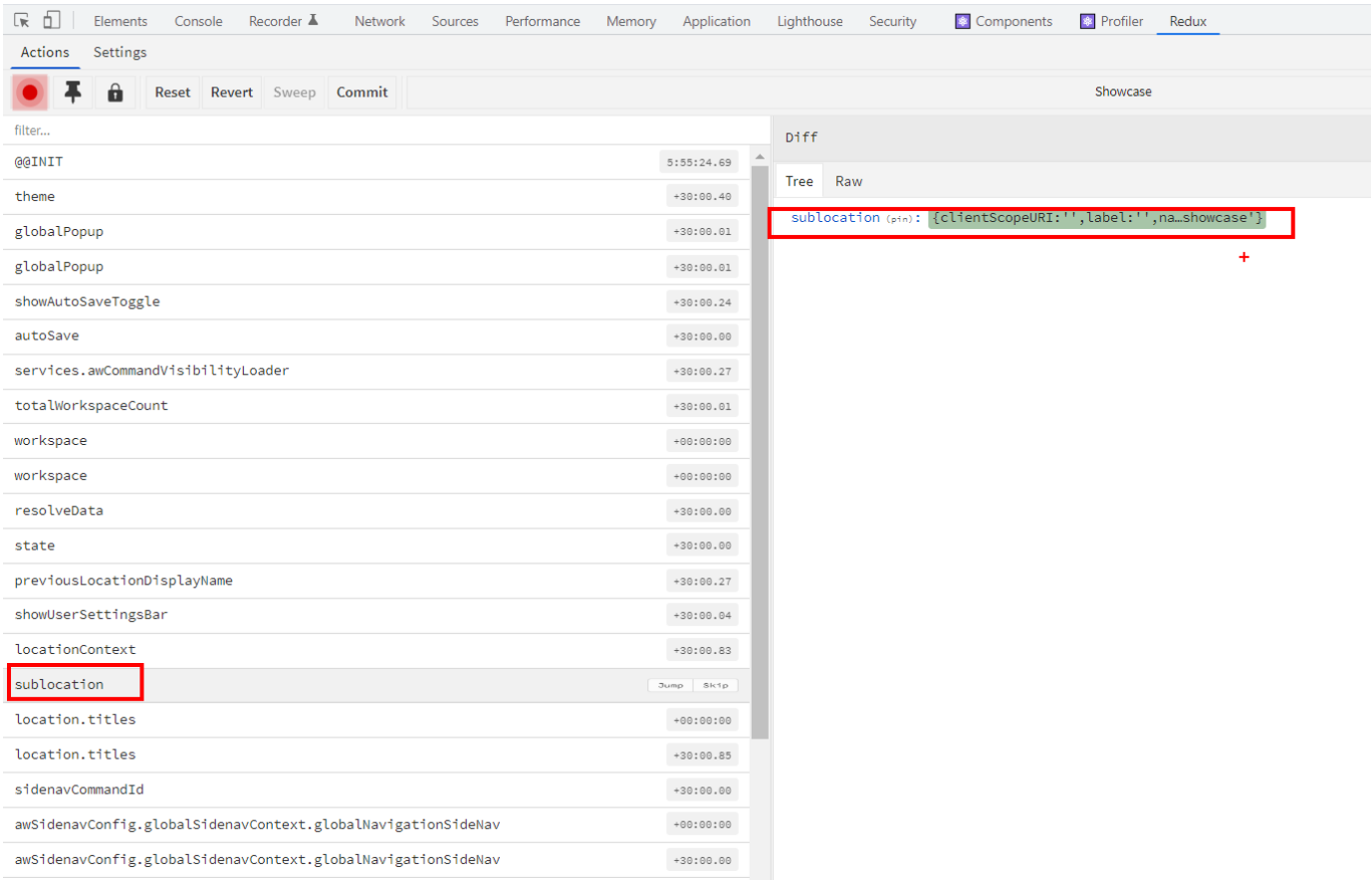

# Redux Development Tools extension for Google Chrome

There is a Chrome extension for helping inspect application-level state (application context) called **Redux Developer Tools**. Here are several links to tutorials and videos to help get you started with it:

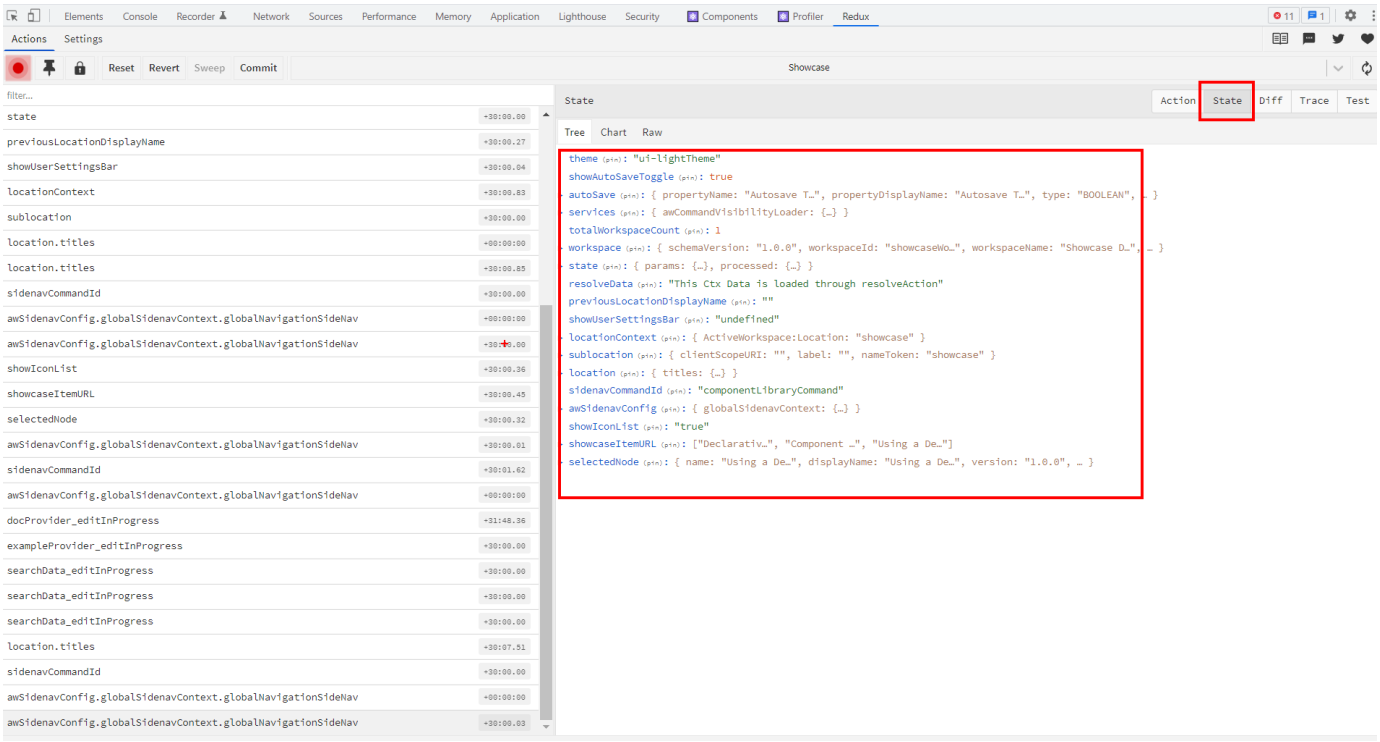
- **Chrome Web Store: Redux Developer Tools**
- **Redux developer tools for dummies**
- **Redux DevTools tips, trick for faster debugging**


Siemens Web Framework
Unpublished work. © 2025 Siemens

This extension records the updates to application context, which can be inspected like this:



This tool also shows the complete value in the application context at any point in time:




# Recommended Visual Studio Code extensions

In recent year, Visual Studio Code slowly became the go-to editor for frontend development for many developers. Although you can use whatever editor or IDE you want for Siemens Web Framework projects, Visual Studio Code is recommended because:

- It is free and open source
- It provides support for frontend languages like JavaScript and TypeScript out of the box
- It provides all the typical features of a programming editor
- It is maintained and released very often with improvements and bug fixes
- It can be extended through contributed extensions

The following Visual Studio Code extensions are recommended when working on Siemens Web Framework projects:

- **Beautify**
- **Code Spell Checker**
- **CSS Peek**
- **Cucumber (Gherkin) Full Support**


Siemens Web Framework
Unpublished work. © 2025 Siemens

- **EditorConfig for VS Code** (required to honor `.editorconfig` files)
- **ESLint** (required to honor `.eslintrc.js` files)
- **Git History**
- **Git Lens**
- **IntelliSense for CSS class names in HTML**
- **npm**
- **npm Intellisense**

Additionally, the following extensions could also be useful in some cases:

- **Eclipse Keymap** Useful if you are used to the Eclipse IDE.
- **Folder Indexing** Useful for speeding up file loading.
- **Path Intellisense** Useful for auto-completing file names.
- **SVG Viewer** Useful to visualize SVG files.

Unpublished work. © 2025 Siemens

# 9. Upgrading

Siemens Web Framework strives to maintain compatibility across different versions, in particular in relation to its core declarative contracts, nonetheless, as with every software, sometimes it is necessary to make some changes that have some impact on the code and configuration written by users.

This section provides step-by-step instructions on how to migrate to a particular version of Siemens Web Framework. Whenever possible, upgrade scripts are provided to automatically perform the changes or at least provide some guidance on what needs to be changed.

## How to Upgrade

This section is divided based on specific Siemens Web Framework releases. To upgrade from a version to another, follow the instructions from your current version to the version you want to upgrade to.

For example, if you are on version 4.0 and you want to upgrade to version 4.3, follow the instructions to upgrade to version 4.1, 4.2, and finally 4.3, in sequence.

## Upgrading to version 4.x

This section explains how to upgrade to version 4.x of Siemens Web Framework.

**Warning**

It is assumed that your project has already been upgraded to the latest minor or patch release of version 3 of Siemens Web Framework. If it has not, complete previous upgrade steps before continuing the upgrade.

The following pages explain more in detail the steps to perform to upgrade to each specific Siemens Web Framework 4.x release. Follow them in sequence depending on your upgrade scenario. If a version is not listed, no upgrade steps are necessary for that particular version.

To upgrade to version 4.0, see to the following pages:

- ES6 conversion
- Webpack Build
- Unit Test Conversion
- Service Conversion
- Manual Service Conversion

To upgrade to other minor releases of version 4, see the following pages:


Unpublished work. © 2025 Siemens

- Upgrading to version 4.1
- Upgrading to version 4.2
- Upgrading to version 4.3


# Webpack Build

## Why Webpack?

As part of **Siemens Web Framework 3.3.0**, Webpack is being introduced as the module loader in replacement of RequireJS. This enables code to be written with ES6 module syntax, which was not supported in RequireJS. Webpack also provides the ability to transpile ES6 modules to be compatible with IE, since ES6 modules are not natively supported by IE.

## What is Webpack?

At its core, webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.

Source: **https://webpack.js.org/concepts/**

## How to use Webpack?

Starting in Siemens Web Framework 3.3.0, webpack will be enabled by default. No action should be needed unless the AFX application you are using is testing Webpack for the first time. In fact Afx's usage of WebPack is abstracted from Afx development. The Siemens Web Framework may choose to swap out the usage of WebPack to say Rollup or other bundler technology and the Afx consumer developer ought not to be impacted. Review the below "Notes" and "Webpack Configurations" sections for more details on determining if any changes are necessary to enable webpack.

```
Note : Starting with Siemens Web Framework 3.3.0, the RequireJS build is
deprecated and will be removed in Siemens Web Framework 4.0.0.
```

## How To Use Webpack DevServer ( express --devServer / AFX-Gatweway with --devServer )

- DevServer is only available when using AFX-Gateway.
- The devServer flag will enable Webpack DevServer, which will watch your source files for changes and reload the browser when the changes are finished being built. This is recommended for faster iterative development.
- After making a code change while running 'devServer', you do not need to run any build commands or manually refresh the browser. This will be done automatically.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## How to implement Webpack DevServer (Non AFX-Gateway consumers)

- If your application is not using AFX-Gateway, you can implement devServer by referencing the implementation code here:

```
/**
 * @param {Object} app - express application
 * @param {Object} logger - logger
 * @param {Object} siteDir - the parent dir of assets.
 * @return {Object} devServer instance
 */
module.exports = async( app, logger, siteDir ) => {
    const webpack = require( 'webpack' );
    const webpackConfig = require( `${siteDir.replace( /\\/g, '/' )}/
webpack.config.js` );
    webpackConfig.plugins.push( new webpack.HotModuleReplacementPlugin() );
    webpackConfig.entry.push( 'webpack-hot-middleware/client?reload=true' );
    const compiler = webpack( webpackConfig );
    // Strip '.' from beginning of public path if exists.
    let publicPath = webpackConfig.output.publicPath;
    publicPath = publicPath.startsWith( '.' ) ? publicPath.substring( 1 ) :
publicPath;
    let devServerInstance = require( 'webpack-dev-middleware' )( compiler, {
        noInfo: true,
        publicPath: publicPath,
        stats: {
            colors: true,
            all: false,
            errors: true,
            warnings: true,
            logging: 'warn'
        },
        watchOptions: {
            ignored: [ '**/test/*Test.js' ]
        }
    } );
    app.use( devServerInstance );
    app.use( require( 'webpack-hot-middleware' )( compiler ) );

    // Disable caching to ensure the latest code is loaded when Webpack
initiates a browser refresh
    app.use( ( req, res, next ) => {
        res.set( 'Cache-Control', 'no-store, no-cache, must-revalidate,
private' );
        next();
    } );

    logger.info( '+ webpack devServer enabled' );

    return devServerInstance;
};
```

- In your package.json include the following dependencies:


Unpublished work. © 2025 Siemens

- "webpack-dev-middleware": "3.7.0"
- "webpack-hot-middleware": "2.25.0"

- It is recommended to control the execution of the devServer code behind a flag since devServer should only be used for development.

## FAQ

### How can I fix the console warning: "Unexpected import path 'depName'"?

Webpack must be able to statically analyze dynamic dependencies at build time. The above console message will be given as a result of webpack being unable to find a static reference to the file being dynamically loaded. When this message is given, it means webpack could not find the file in any bundle it created at build time and failed to load the dependency.

If this console message is seen, it is recommended that the dependency mentioned in the message is added to a new/existing bundle defined in the appropriate kit.json, or added to the 'extraDeps' section of the appropriate kit.json. In the below code you will find other alternatives to resolving the console message.

```
var a = 'js/someDependency';

// Avoid using imports of variables, this makes it impossible for webpack
to statistically analyze what dependency
// The application will need at runtime for the below lines.

require( [ a ], function() {} );
requirejs( [ a ], function() {} );
import( a ).then( function() {} );

// Instead use a hard-coded string for the dynamic dependency
import( 'js/someDependency' ).then( function() {} );

// Or if a hard-coded string can not be used, add the dependency as an
entry in the extraDeps array of kit.json
// and use the afxDynamicImport function. Note: afxDynamicImport is defined
globally and can be used without needing to import a file to use it.

afxDynamicImport( [ a ], function() {} );
```

### I am encountering import issues when using a bundled AMD file

Webpack is unable to interpret files where there are multiple AMD modules concatenated into the same file. It is recommended to use a UMD format of these files or an unbundled AMD form of these files.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Vendor code (repo/lib) is not working with Webpack enabled

If you encounter an issue with vendor code that is only occurring with Webpack enabled, we suggest you visit the vendor's github page or documentation to find out if the code is compatible with Webpack, or what workarounds are needed to make the code work with Webpack. In most cases, the issue encountered can be found in the vendor's Github issues section with suggestions on how to resolve the problem.

## Source map debugging and devServer are not working with IE11

IE11 does not support the default source map type used in builds. If you need source maps for debugging in IE, run the following command in your terminal before building, "set SOURCE_MAP_TYPE=source-map". This option is not used by default since it can increase build time ~15 seconds. Webpack DevServer is not currently supported as of Siemens Web Framework 3.3/AW4.3, but is planned to be supported in the next release.

## I am receiving a "JS Heap out of memory" error when building with Webpack

This is a former issue which has been fixed as of Siemens Web Framework 3.3.0-380. If you are still encountering this issue please open a ticket.

## Webpack Configurations

- `entryFiles` [May Be Required] Override the application entry html files ( Default: [index.html], Defined in kit.json )

```
//kit.json
{
    "bundler": {
        "entryFiles": ['index.html']
    }
}
```

- `chunkMinSize` [Optional] Override minimum Chunk Size in production mode ( Default: 1000000, Defined in kit.json )

```
//kit.json
{
    "bundler": {
        "chunkMinSize": 1200000
    }
}
```

- `chunkMaxSize` [Optional] Override maximum Chunk size in production mode (Default: 3500000, Defined in kit.json)

```
//kit.json
{
    "bundler": {
```


Unpublished work. © 2025 Siemens

```
        "chunkMaxSize": 2200000
    }
}
```

- `provides` [May Be Required] Provider Entries : - To load libraries globally like jquery. This configuration needs to added into bootstrap.json for the library.

```
//bootstrap.json
{
    "config": {
        "paths": {
            "jquery": "lib/jquery/jquery.min"
        },
        "provides": [
            {"window.jQuery": "jquery"},
            {"jQuery": "jquery"}
        ]
    },
    "import": [
        "jquery"
    ]
}
```

- `extraDeps` [May Be Required]: Specify dependencies that are using dynamic import statements with variables. Ex: import( variable ); ( Default: [], Defined in kit.json )

```
//kit.json
{
    "extraDeps": [
    "js/someDepA",
    "js/someDepB",
    "js/someDepC"
    ]
}
```

## ES6 Conversion

As part of the Beyond Angular roadmap, the first step is to convert existing JavaScript code to ES6 modules and support all ES6 features for development.

Going forward, any JavaScript code that is getting added to the system must use the ES6 import pattern for dependency management. We have provided a script to help teams with the conversion on the existing code. We are also adding checks in the Siemens Web Framework audit scripts so that new code being written conforms to the ES6 format.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Example of ES6 conversion

**Before**

```
define( [ 'app', 'lodash' ], function( app, _ ) {
    var exports = {};
    exports.testExport = function() {
        return 'testMethod';
    };
    exports.testExport2 = function() {
        return 'testMethod 2';
    };
    return exports;
} );
```

**After**

```
import _ from 'lodash';

var exports = {};
export let testExport = function() {
    return 'testMethod';
};
export let testExport2 = function() {
    return 'testMethod 2';
};
export default exports = {
    testExport,
    testExport2
};
```

## How to convert codebase to ES6

To help you migrate to ES6, we have authored a utility (amdtoes6), which converts existing ES5 files to ES6 files. Run the utility on your code base and check the console output to see if you need to convert the files that are not handled by the utility. The name of the utility is `amdtoes6`

## How to run the utility

- To get help on the utility, use the following commands.

```
amdToes6 --help
```

If this does not work, run the script through the node command.

```
node node_modules/afx/build/js/amdToES6.js --help
```

- To run the utility on the source code, add the sourcePath parameter to it. Regex is supported.

```
amdToes6 --sourcePath=<relativeFilePath/GlobPath> (For Eg. src/
uielements/**/*.js - convert all files in uielements module)
```

Or

```
node node_modules/afx/build/js/amdToES6.js --sourcePath=<relativeFilePath/
GlobPath>
```

- Examples
  - To convert the single amd file to es6, run the command `amdToes6 --sourcePath=src/uielements/src/js/aw-avatar.directive.js` command.
  - To convert the js folder from module1, run the command `amdToes6 --sourcePath=src/`/module1/src/`/*.js` command.
  - To convert the js and test folders from module1, run the command `amdToes6 --sourcePath=src/`/module1/src/`/*.js,src/**/uielements/test/*.js` command.

## Known Limitations of Utility

In some cases, the utility throws both errors and warnings.

- An error log is shown in RED color. Error-ed files are not converted by the utility.
- The Warning log is shown in YELLOW color. The files (listed in the Warning log) are converted by the utility but side effects could occur based on how the consumer is using it.

Listing all such instances here:

- Exporting variables or Literals If the js is exporting a variable, the script uses the following pattern to convert it to ES6.

  **Before**

```
define( [ 'lodash' ], function( _ ) {
    var exports = {};
    //Exported Variables
    exports.source = '$source';
    //Exported Function
    exports.testExport = function() {
        return 'testMethod';
    };
} );
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

**After**

```
import _ from 'lodash';
var exports = {};
//Exported Variables
export let source = '$source';
//Exported Function
export let testExport = function() {
    return 'testMethod';
};

export default exports = {
    source,
    testExport
}
```

**Note**

If other ES5 format files are changing the value of this variable, then it will not work. In this case, the ES5 file needs to be converted to ES6 or expose the variable through setter or getters. In this case, the script will show the following warning:

```
[14:35:36]: warn: In src\js\browserUtils.js
[isIE,isNonEdgeIE,isMobileOS] lieterals are exported and they may not
be work if someone is setting the value explicitly, It's better to
replace with getter/setter functions.
```

- Returning top-level function In the source code, if you are returning a function, then the script does not convert such files. This is because ES6 does not support exporting anonymous functions. The only way to support this is to use `export default`. But, if you use the export default, then the ES5 callers of this function will break. To overcome this, you also need to convert the ES5 callers. The script throws the following error in this scenario:

```
[20:11:10] error: Not able to convert src\js\serviceExample.js Top level
return is not object/identifier. Convert this file manually and also
convert all ES5 callers to ES6
```

- Multiple Define blocks If the source code contains multiple define blocks, then it is possible to convert it through a script. This pattern is sometimes used in unit tests. You need to refactor the code to remove such usage.

## Unit Test Conversion

As part of preparing our unit tests for Beyond Angular, we need to refactor all our unit tests to achieve the following.

- Use `angular.mock.inject` in a non dependent way.


Unpublished work. © 2025 Siemens

- Reduce/Eliminate the usage to `angular.mock`.

## Refactoring Use cases

```
beforeEach( function() {
    angular.mock.module( function( $provide ) {
        $provide.service( 'iconService', function iconServiceMock() {
            return {
                getTypeIcon: function() {
                    return 'Nobody Home';
                }
            };
        } );
    } );
} );
```

For the unit test which has mock `app.getInjector` already, refactor it properly to make both mock and normal `app.getInjector` function work properly.

```
var injector = app.getInjector();
var mySpy = spyOn( injector, 'get' ).and.callFake( function( serviceName ) {
    // Your mick logic
    if( serviceName === 'mockAuthenticator' ) {
        return mockUserPwAuth;
    }
    // pass through logic to make system not breaking.
    return mySpy.and.callThrough()( serviceName );
} );
```

## Unit test refactoring script

Use the automation script provided to refactor unit tests or convert them manually by following the above guidelines.

```
node ./build/js/convertTestToUseInjector --sourcePath=src/**/*.js
```

**Note**

If the script errors out for a specific pattern in your test file, handle it manually


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Service Conversion

## Prerequisite: Unit Test Refactor

If you have not refactored, refer to the **Unit Test Refactor** to refactor the unit tests.

```
Unit Test Refactor is required for using the latest AFX
```

## Service Conversion Steps

1. Run `convertService` script to convert the level 1 services (level 1 means service doesn't have any dependency or all dependent services are already converted). After you run the script level 2 dependencies will become the level 1 and continue the same process until all services converted.

If you are AFX developer run

```
node ./build/js/convertService.js --baseSourcePaths=src/**/* --
sourcePathToConvert=src/**/*
```

If you are application developer run

```
node node_modules/afx/build/js/convertService.js
--baseSourcePaths=node_modules/afx/src/**/*,src/**/* --
sourcePathToConvert=src/thinclientfx/tcuijs/**/*
```

- --baseSourcePaths : path of the all dependent services and services to convert (separated by comma)
- --sourcePathToConvert : services to convert

1. If no error and everything runs fine
   - Open git tool, compare, the change, make sure all the conversion is correct.
   - Remove the `app.factory` at the end, If there are no any instances where you are using the service through angular injection like directive or controllers, you may be need to keep there for sometime until you remove the angular injections.
   - At all the caller use ES6 import way to use the service.
2. If the `convertService` has error print
   - Follow **manual service conversion** to do manual refactor.
   - 99% of the error case will be inside the `factory/service` function it has execution statement which is doing initialization of work, you need to decide:
     - Either put it to top level of your module.
     - Or convert your service to ES6 class and use it like `MyClass.instance`.


Unpublished work. © 2025 Siemens

3. If the `convertService` has warning print
   - If warning says there is `app.getInjector` usage inside service replace the `app.getInjector` by `import`. If you are using the `app.getInjector` to avoid the circular dependency then you might be need to think about design refactor.In extreme case you might be think about `declUtils.loadDependentModule`
   - If warning says there is `requirejs` usage inside service, replace the `requirejs` by `declUtils.loadDependentModule` or `declUtils.loadDependentModules` because when afx decides to remove the `requirejs` usages it will be easy to remove from one place only.
4. Repeat the same process (Go to step 1) until all services converted.

### FAQ

- **I am adding new service should i use `app.factory / app.service` for service registraion?**

  **Answer:** No, Don't use the `app.factory / app.service` for service registraion. Newly added service should use ES6 pattern. Simply import the service wherever you needed. Also try to use native JS features like use Promise instead of AwPromiseService/$q.

## Manual Service Conversion

### Refactor Services

All the code inside `app.factory` and `app.service` practice, has two different types

- Definition code

```
// conditionService.js
exports.attachDynamicCondition = function( declViewModel, propertyName,
conditionObj, dataCtxNode ) {
    // ....
}
```

  For this we will move out of `app.factory` directly.

- Execution code

```
// appCtxService.js

// Put the state parameters into the context
exports.registerCtx( 'state', $state.params );

// When the state parameters change
$rootScope.$on( '$locationChangeSuccess', function() {
    // Update the context
    exports.registerCtx( 'state', {
        params: $state.params,
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        processed: processParameters( $state.params )
    } );
    eventBus.publish( 'LOCATION_CHANGE_COMPLETE' );
} );
```

For this:

- If it is not related to AngularJS context, move it out to top level.
- If it is related to AngularJS context, put it to `_init` function for now.

After above steps, most of our code should look like:

```
import aaa from 'aaa';
//...
let exports;

export let functionA = function() {
};

// NOTE: if have init left over
export let _init = function() {
}

export default exports = {
    functionA,
    ...
}

// AngularJS leftover...
export let moduleServiceToInject = 'bService';

app.factory( 'bService', function(){
    // if really has init
    _init();
    return exports;
})
```

## Refactor AngularJS Leftover

- For the `_init` function, we will put proper refactor to:
  - Limit their scope, not spread at critical dependency tree or everywhere
  - Do correct refactor, not breaking any functionality
- For the one which ought to have `_init` function:
  - We will convert it to ES6 class and treat it like OOTB AngularJS Service, which means use it like `aaaService.instance`.
- For `app.factory`/`app.service`, it will be reserved for at least 1 release for not impact our user when they are still in name injection practice.


Unpublished work. © 2025 Siemens

### FAQ

- **Can I convert a module to class later on? What is the impact on consumers?** Yes we can, and all the usage will be change from:

```
import appCtxSvc from 'js/appCtxService';

// Do it in any place
appCtxSvc.getCtx('aaa');
```

  to:

```
import AppCtxSvc from 'js/appCtxService';

// Remeber to do it in the context where angularJS is ready
somefunction() {
    AppCtxSvc.instance.getCtx('aaa');
}
```

  The best practice here is create a new service to reuse old method, rather than overwrite the old method. Anyway it should not happen at the context beyond angular.

- **How consumers will know which one is class and which one is module? How can we make it easy?**

  Customer should know all details below. That is not the new question when compare with existing:

  - What I am importing?
  - How many interfaces we have in that imported code?
  - How to use it? ( angular way or non angular way today)

  When source reference enabled, VS Code IntelliSense can help to identify that. But before that no change, no improvement, no regression.

## Upgrading to Version 4.1

After the introduction of lifecycle hooks from version 4.1 and deprecation of the `.contentLoaded`, `.contentUnloaded` and `awPanel.reveal` event subscriptions, a utility (`putLifecycleHooks.js`) was developed to remove the deprecated events from view model files. This documentation mentions the procedure to execute the utility.

### How to run the utility

1. Navigate to the project in command prompt.


Siemens Web Framework
Unpublished work. © 2025 Siemens

2. If the project to be updated is AFX then just run

```
node build/js/putLifecycleHooks.js
```

### Evaluating results after the utility is run

- If there are more than one instance of each of contentLoaded and/or contentUnLoaded then the view model is left undone and such cases are reported under `cases where batch action is required`.
- If the `event` object in `onEvent` has more than 2 keys ( other than 'eventId' and 'action' ) then such cases have to be handled manually and are reported under `cases not processed`.
- If the event-name (in event-name.contentLoaded OR event-name.contentUnloaded) does not match with the view model name then such cases have to be handled manually and are reported under `cases not processed`.
- Instances of `awPanel.reveal` cases are reported under `awPanel cases`.

### Instances which can not be converted to life cycle hooks

There are certain instances which have to left unchanged due to various constraints like the way the view/viewModel is called/loaded. These instances are of the following variety:

- If the event name ( for ex. in event-name.contentLoaded ) does not match with the view model name (which contains this instance), it has to be checked whether that event is being called/listened in the corresponding view. If yes, then that instance of contentLoaded/contentUnloaded will stay as it is, that is no change.
- In awPanel.reveal cases if the view/viewModel is being included/called in some other view by 'aw-command-sub-panel', then that instance has to be left as it is.
- There are cases when the event-name in event-name.contentLoaded (or with .contentUnloaded) does not match with viewModel name and also not included in the corresponding view, there is no change to be made.

## Upgrading to Version 4.2

The following pages list all the steps to perform to upgrade to Siemens Web Framework v4.2:

- **HTML Self-Closing Tag Conversion**
- **Removing Extended Tooltip Configurations**
- **Upgrading Action Message Configurations**


Unpublished work. © 2025 Siemens

## HTML Self-Closing Tag Conversion

The Siemens Web Framework 4.2 need upgrade to jQuery3.5.1 to fix a security vulnerability. It's a breaking change, the non-void self-closed tag is not supported anymore.

For example:

```
<div/>
<aw-widget prop='data.myProp'/>
```

The applications need to update their HTML to use dedicated closing tags:

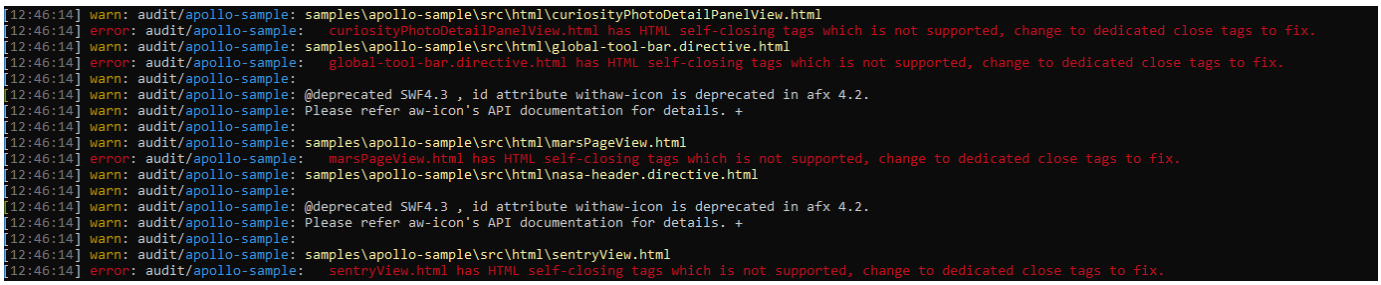
```
<div></div>
<aw-widget prop='data.myProp'></aw-widget>
```

It is recommended to use the below command to perform some static code analysis that is reinforced by the framework. The audit check flags up errors or anti-patterns detected. The audit check for a module will kick in only if audit is enabled for a particular module.

```
aw audit
```

For example: If some files don't have proper closing tag as shown below. `aw audit` command will report errors as shown in the below picture.

```
<div/>
<aw-break/>
<aw-widget prop='data.myProp'/>
<aw-include name="filteringEvent"/>
```




### Migration utility

Siemens Web Framework 4.2 provided a utility script to help applications with the conversion of the existing code. It supports HTML files and JS files with the embedded HTML snippet.


Unpublished work. © 2025 Siemens

These HTML native void tags are still allowed to use self-closing tags.

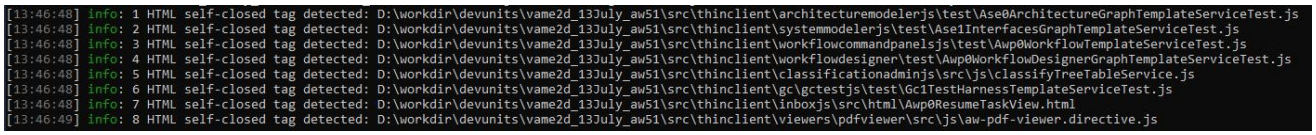
```
area|br|col|embed|hr|img|input|link|meta|param
```

### How to run the utility

To run the utility, files should be in writeable mode.

1. Run the script in dry run mode.

```
htmlCloseTagConvertor --sourcePath=src --dryrun=true
```




2. Find out the impacted files.
3. Rerun the script.

```
htmlCloseTagConvertor --sourcePath=src/uielements (convert all files
in uielements module)
```

- To get help on the utility, use the following commands.

```
htmlCloseTagConvertor --help
```

- To run the utility on the single source folder:

```
htmlCloseTagConvertor --sourcePath=src/uielements (convert all files in
uielements module)
```

- To run the utility on the multiple source folders:

```
htmlCloseTagConvertor --sourcePath=src/uielements,src/command
```

- To dry-run the utility without doing the real file update:

```
htmlCloseTagConvertor --sourcePath=src --dryrun=true
```

### Known Limitations of Utility

1. When converting the HTML snippet embedded in JS file, the utility can't handle the conversion correctly if the HTML open tag and close tag are not logically in one statement, or if it only has open tag and no close tag. Typically it happens in dynamically computed HTML snippets. The user has to check the converted JS files and manually correct if there are conversion errors.

   For example, variable `iconTag` only has svg open tag, no close tag. In variable `iconTag2`, it'll wrongly convert the img close tag to svg close tag.

   ```
   var iconTag = ImageCacheService.instance[ key ].replace( '<svg ', '<svg
   class="aw-base-icon" ' );
   var iconTag2 = '<img class="aw-base-icon" src="' +
   exports.getTypeIconFileUrl( typeIconFileName ) +
       '" draggable="false" ondragstart="return false;" />';
   ```

   This will be wrongly converted to:

   ```
   var iconTag = ImageCacheService.instance[ key ].replace( '<svg ', '<svg
   class="aw-base-icon" ' );
   var iconTags = '<img class="aw-base-icon" src="' +
   exports.getTypeIconFileUrl( typeIconFileName ) +
           '" draggable="false" ondragstart="return false;" ></svg>';
   ```

2. When converting the HTML file, the utility can't convert the html self closing tags if it contains symbol `> (greater than)` used in any attribute.

   For example, attribute exist-when in aw-include tag below has used > symbol, the script will ignore this occurrence. Such cases need to be handled manually.

   ```
   <aw-flex-row height="fill">
       <aw-include name="sample-include" exist-
   when="ctx.selectedNode.examples[0].previewURL.length > 0" />
   </aw-flex-row>
   ```

   The audit will also ignore this occurrence.

## Removing Extended Tooltip Configurations

From **Siemens Web Framework v4.2.0** onwards, any command which uses `extendedTooltipDefault` will be thrown as error in the audit run.

Following will be the message on prompt:

```
Command <commandId> no longer requires extendedTooltip configuration to disp
lay title or description in tooltip.
```


Unpublished work. © 2025 Siemens

Hence, command configuration specified below is invalid:

```
{
    "commands": {
        "commandId": {
            "title": "commandTitle",
            "extendedTooltip": {
                "view": "extendedTooltipDefault"
            }
        }
    }
}
```

Also the 'title' and 'description' is compulsary to be specified in the command. In order to make such changes in the commands view model files, a utility was developed to carry out the change.

**Prerequisites:**

1. If the utility is not to be run on AFX modules but on an application, then make sure to build the application first, in order to get compiled i18n module generated.
2. Include the excel file which contains data about commands, in the root of the project.

**How to run the utility:**

1. Navigate to the project in command prompt.
2. If the project is an application then run command :

```
node node_modules/afx/build/js/removeExtendedTooltipConfig.js
```

3. If the project to be updated is AFX then just run

```
node build/js/removeExtendedTooltipConfig.js
```

**Evaluating results after the utility is run:**

1. An excel file ‘updatedcommandslog.xlsx’ will be generated in the root of the project folder. This contains all the commands ( command ids, commands view model path, command description ) which have been processed and updated successfully during the run.
2. On the prompt, a log will appear after the run is complete. This displays information for commands which could not be updated, and hence have to be handled manually. (See section ‘Failures Expected’)

**Failures Expected:**

1. Command has `title` attribute which is one of the following^1^:

```
{
    "commands": {
        "commandId": {
            "title": ""
        }
    }
}
```

```
{
    "commands": {
        "commandId": {
            "title": "commandTitle"
        }
    }
}
```

2. i18n message file as pointed in command's title attribute is not available in the same module^2^.
3. A command's description is already present in compiled i18n or the i18n file in the same module but the corresponding description key is not used even once in the commands view model^3^.
4. A command is present in the excel file but could not be found in any commands view model under the project.

**Notes**

1. The `title` attribute should only be specified as following. Otherwise, it will be errored out during utility run.

```
{
    "commands": {
        "commandId": {
            "title": "{{i18n.titleKey}}"
        }
    }
}
```

2. This is an edge case because it may happen that the same description or title is used in a commands view model present in some other module but these two modules are not interdependent. Hence the description key needs to defined explicitly in the i18n object of the commands view model. The same entry needs to be put in the respective message file separately.
3. If the description key is not used even once in the commands view model then a user's descertion is needed to use the sme description key.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Upgrading Action Message Configurations

**Note**

The concept of configuring Action Messages in the ViewModel is same. Only difference is from Siemens Web Framework 4.2 onwards framework will not process any error which is not configured by end-user.

### Why is the change needed?

Siemens Web Framework handles all the errors by default and shows error messages on the screen. These can be server error, framework error or application configured error, even if application team haven't configured it. This has got apps team into a situation, where the control on displaying error message does not lie with the application who owns the use case.

It causes:

- Showing up errors which does not make sense to the user.
- Showing up errors which is not even needed to be shown.
- Duplicate/Multiple error messages is shown in many cases.

### What is the change?

Now, the control to show the error message has been given completely to the use case, which means, the functionality needs to decide, if the error thrown by framework needs to be shown to client or not. It anyway gets logged in browser console.

### What is the impact to application?

In some cases, application teams relying completely on framework to show a error message, and haven't configured their own error message, would get impacted.

### What is expected from application teams?

- Test use cases manually to check if an error was being thrown in previous version of their app w.r.t to Siemens Web Framework 4.2 onwards.
- Evaluate if this error is indeed necessary to be displayed to the user. If yes, is the language following UX guidelines?
- Configure error message display in application code only if needed.


Unpublished work. © 2025 Siemens

### Snippet for mapping the server error to messages

In the below example, consumer need to use `errorCode` interpolation to map the error coming from server to client.

```
{
  "myAction": {
    "actionType": "JSFunctionAsync",
    "method": "myFunction",
    "inputData": {},
    "outputData": {},
    "deps": "...",
    "actionMessages": {
      "failure": [
        {
          "message": "myFailureMessage"
        }
      ]
    },
    "messages": {
      "myFailureMessage": {
        "messageType": "ERROR",
        "messageText": "{{errorCode}}"
      }
    }
  }
}
```

## Upgrading to Version 4.3

As of version 4.3, the `aw-chart` and `aw-column-chart` are no longer included in the `afx` kit like in the previous versions. This was due to licensing issues, as both components require a licensed version of the Highcharts library to be used in applications.

If you have to use these components in your application (and you have a licensed version of the Highcharts library), then you have to specifically add a dependency to the `afx-charts` kit in the `kitDeps` array of your `kit.json` solution kit, like this:

```
{
    "name": "mySolution",
    "level": 5,
    "kitDeps": [
        "afx",
        "afx-charts"
    ]
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Warning**

Make sure the components used in building the distributed applications are complaint with a licensed version of their dependent libraries. If the application is being used for an internal or in-house project or for learning purposes and it is not distributed, then there is no such restriction.

# Upgrading to version 5.x and 6.x

This section explains how to upgrade to version 6.x of Siemens Web Framework.

**Warning**

It is assumed that your project has already been upgraded to the latest minor or patch release of version 4 of Siemens Web Framework. If it has not, complete previous upgrade steps before continuing the upgrade.

The following sections explain more in detail the steps to perform to upgrade to each specific 6.x release. Follow them in sequence depending on your upgrade scenario. If a version is not listed, no upgrade steps are necessary for that particular version.

**Note**

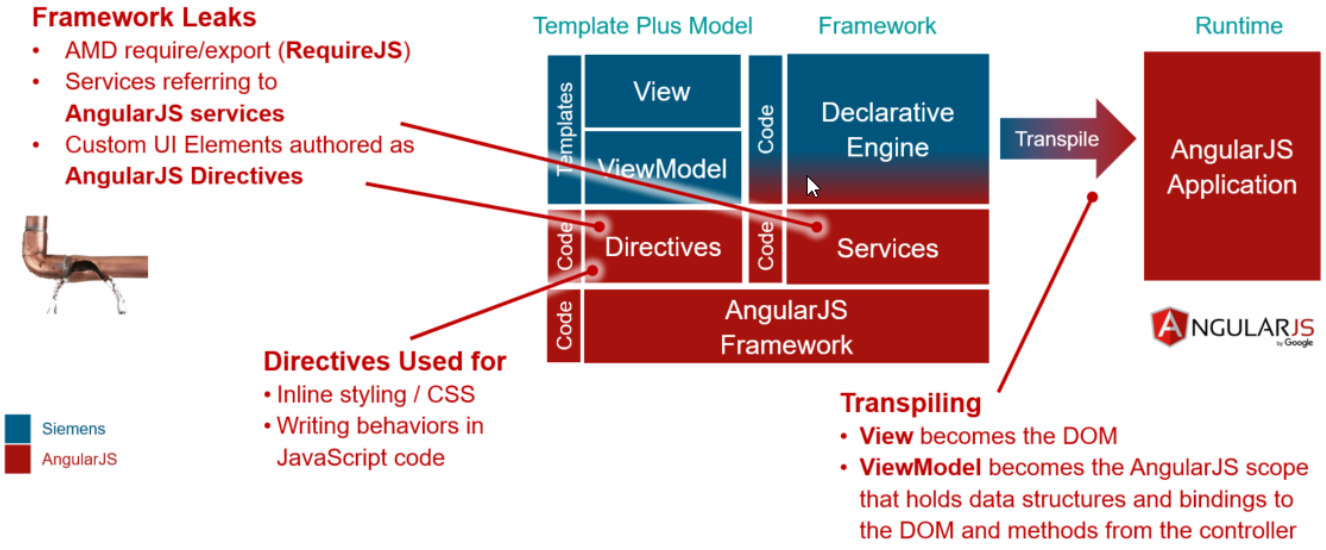
Versions 5.0 and 5.1 of Siemens Web Framework were restricted previews that were released only to specified Siemens Web Framework consumers. As a matter of fact, the upgrade steps necessary for version 5.x are also necessary for version 6.x and direct upgrade from version 4.x to 6.x is supported.

## Beyond Angular Rationale

Siemens Web Framework was originally built using RequireJS and AngularJS. While this proved to be a good foundation at the time, the initial version of AngularJS was released way back in October 2010 and it has now reached its end of life, and with the introduction of modern module loaders and bundlers like WebPack, RequireJS didn't quite seem appropriate for a modern web UI.

Furthermore, one of the biggest problems of previous versions of the Siemens Web Framework was the fact that there were quite a few leaks in the abstraction layer that was provided to consumers, as outlined in the following diagram:


Unpublished work. © 2025 Siemens




## The *Beyond Angular* Transition

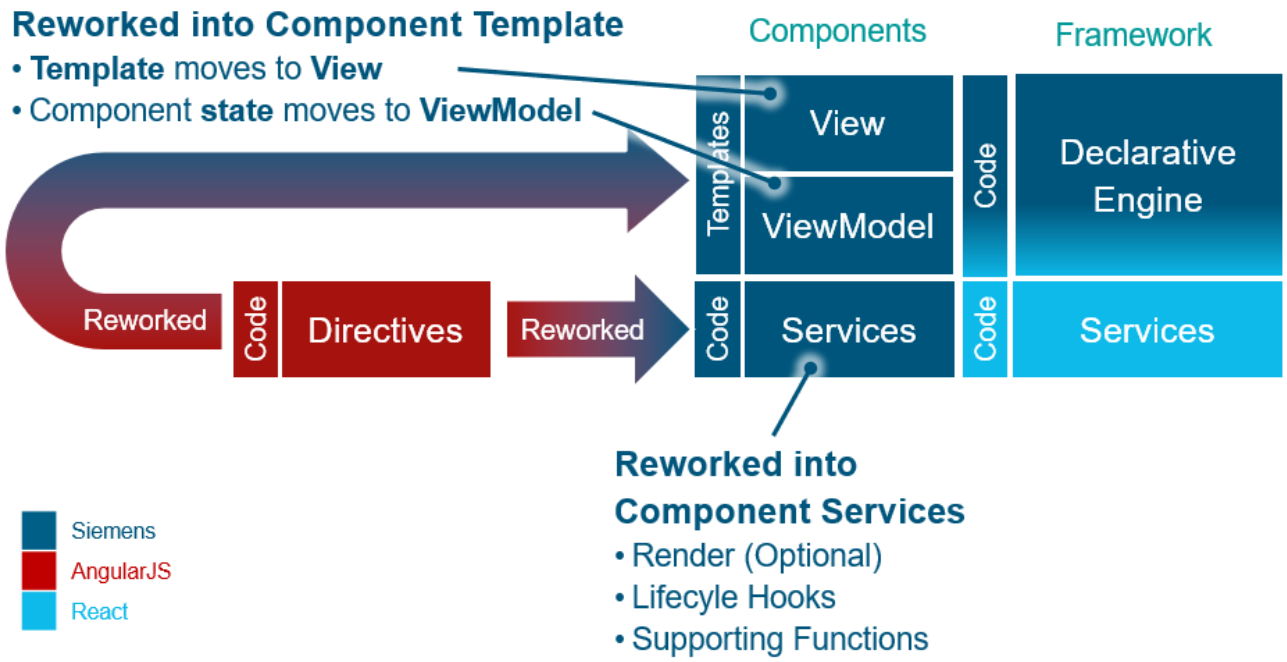
As of version 4.0, a few significant change were introduced into the framework to ease the transition to a newer, more modern architecture:

- RequireJS was removed and **WebPack** was introduced
- The codebase of the framework and consumer applications was migrated to **ES6**
- Services were **reworked** to remove direct dependencies to AngularJS

After that, as of Siemens Web Framework version 5, AngularJS was completely removed and replaced with **React**, a more modern but also significantly different framework from AngularJS.

This meant that all AngularJS directives have been reworked into *Components*, which are now first-class citizens, and can be authored in a 100% framework-agnostic way:


Siemens Web Framework
Unpublished work. © 2025 Siemens



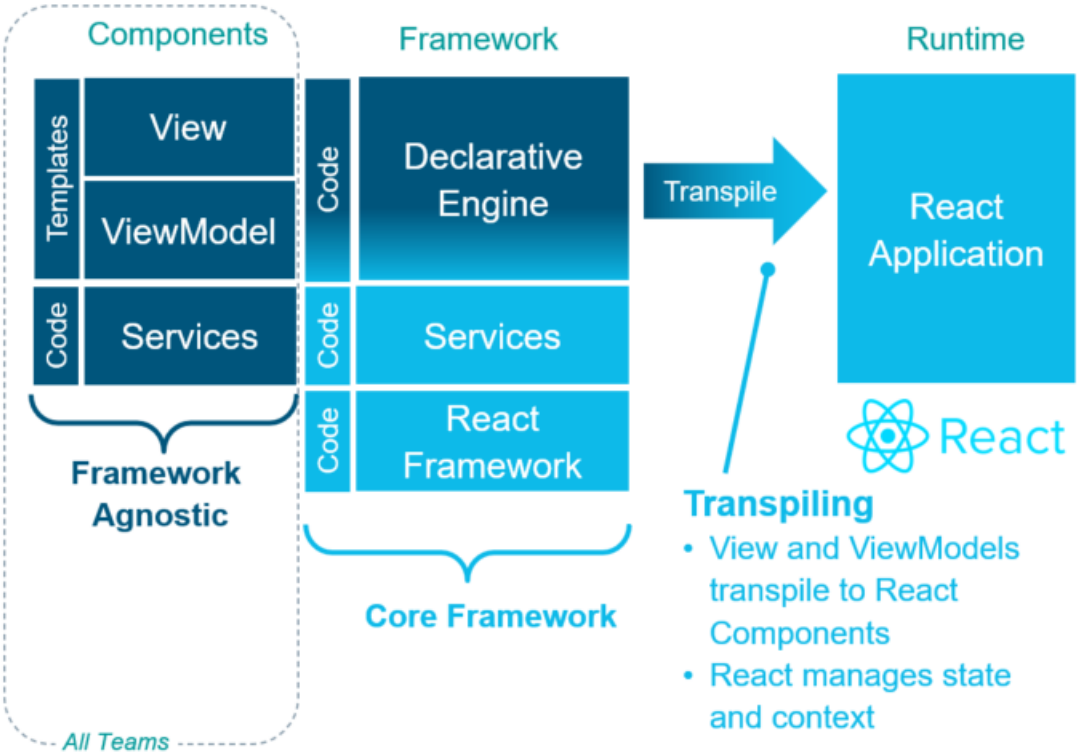

## Current Architecture

As of version 5, the architecture of Siemens Web Framework consists on:

- A fully framework-agnostic part, represented by the same declarative artifacts that have been available in the framework since its inception (Views, view models, etc.) and pure JavaScript Services.
- The inner core of the framework relies on React but does not expose any of its API, and provides suitable wrappers to ensure that there are no abstraction leaks. This small portion of React-dependant code, the *Declarative Engine*, is accessed exclusively by the Siemens Web Framework internals and not exposed to consumers.

At run time, declarative artifacts and framework-agnostic service code are transpiled into a React application, which can be debugged as such at run time.


Unpublished work. © 2025 Siemens




## What's new?

Besides completing the transition of the underlying framework from AngularJS to React, version 5 of Siemens Web Framework introduces several significant new functionalities that greatly enhance the framework and address some of the concerns that were reported most frequently.

### Reusable components

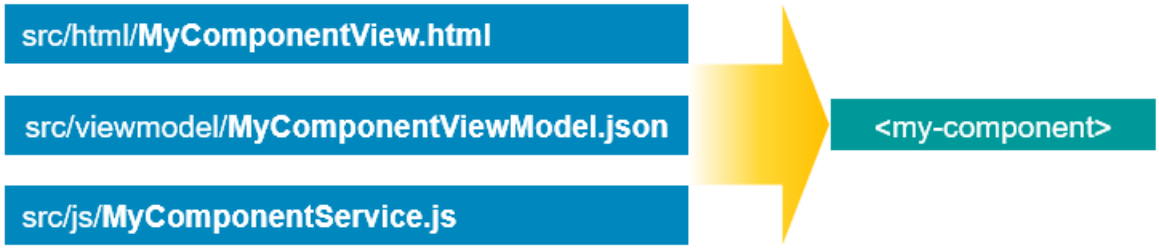
One of the most important requests was the possibility to extend Siemens Web Framework by authoring new **UI elements** in a framework-agnostic way.

So far, the only way to create a new UI element was to develop an AngularJS directive, bypassing the abstraction layer that Siemens Web Framework provided on top of AngularJS.

As of version 5, it is now possible to create reusable **components** in a declarative way. Note that no new artifact was introduced, and a component is comprised of a **view model** and a **view** (plus any **service** code referenced by view model **actions**).

As part of the build process, a view model and its corresponding view are converted into a React component that can be referenced in any view template as a UI element:


Siemens Web Framework
Unpublished work. © 2025 Siemens




Additionally, as an alternative to the traditional view template, it is also possible to configure a *render function* via the new `render` **lifecycle hook**.

**Note**

*Every view model* defines the contract and behavior of a component and will be converted to a React component at run time. To maximize reusability, it is now recommended to avoid constructs that rely on the global **application context** or named **events** in favor of passing data via props and leveraging lifecycle hooks.

## Unidirectional data flow

Perhaps the biggest architectural change when moving from AngularJS to React is the shift from two-way binding, that was relatively common in AngularJS, to **unidirectional data flow**.

This concept must be taken into account when data needs to be shared across various components, as it is now necessary to *pull the state up* to the common parent. Then, data gets passed down as read-only to all the child components.

## Development build

As part of the transition Siemens Web Framework now internally relies on **Create React App** and React Scripts. As a result, it is now possible to run any Siemens Web Framework application in development mode by executing:

```
aw start
```

In this mode, a development web server will be started and it will listen to changes to any of the source files and immediately perform a near-instantaneous incremental build, thereby allowing developers to see their changes in a live environment immediately, without having to rebuild.

## Runtime debugging with React and Redux development tools

Because at run time every view and view model is converted into a React component, it is now possible to use **React Developer Tools** for debugging and inspecting your application at run time and also **Redux**


Unpublished work. © 2025 Siemens

**DevTools** to monitor changes to the global **application context** (which is internally implemented as a **Redux** store).

## Functional Limitations

As of version 6.0, The following functionalities are no longer supported by Siemens Web Framework and will no longer work:

- It is no longer possible to use *two-way data binding* because such binding cannot be realized with unidirectional data flow. To change the state that is owned by a parent component in child components, you can use the new Atomic Data and Fields constructs.
- It is no longer possible to use complex bracket syntax in conditional expressions, such as `ctx[commandContext.myKey].myFlag`. To achieve similar results, consider using the new Derived State construct.
- The common frame generator is no longer supported; use the default common frame and **layout slots** instead.
- The `viewAndViewModelRepoConfiguration` and `imageRepositoryConfiguration` settings in `kit.json` files currently do not work. views and view models are now transpiled into components only at build-time and therefore it is no longer possible to process them at run time.

Additionally, the following functionalities do not work yet in version 6.0, but they may be restored or substituted with similar functionalities in future versions:

- It is not possible to configure **custom bundles** for production builds. At present, bundles will be created automatically when building in production mode.
- The UI Builder editors currently do not work.

## Evaluating the impact of the upgrade

You can use the `sourceInspect` utility to inspect your Siemens Web Framework project and assess the impact of upgrading to version 6. This utility is included in Siemens Web Framework version 4.5, 5.x or 6.x, and generates a set of reports in CSV format containing details about your source code that can be used to determine the areas that will likely be affected by the upgrade.

### Running the `sourceInspect` utility

The utility is available in a different location depending on the version of Siemens Web Framework that you are using.


Siemens Web Framework
Unpublished work. © 2025 Siemens

On a Siemens Web Framework 4.5 project, run the utility as follows:

```
node node_modules\afx\build\js\sourceInspect --sourcePath <directory-to-scan>\**\* --destPath <output-directory> --assessComponents
```

On a Siemens Web Framework 5.x or 6.x project, run the utility as follows:

```
node node_modules\@swf\tooling\js\sourceInspect --sourcePath <directory-to-scan>\**\* --destPath <output-directory> --assessComponents
```

Where:

- `<directory-to-scan>` is a directory (within the current Siemens Web Framework project or another project) that you want to scan and generate reports for.
- `<output-directory>` is the directory where report files will be saved to.

## Generated reports

After it is executed, the `sourceInspect` utility generates the following report files:

| File Name | Contents |
|---|---|
| AllDataMutationsInFunctions.csv | Lists all the functions containing direct data mutations, their complexity in LOC, and the names of the mutated variables. |
| AngularJSReferences.csv | Lists all the references to AngularJS APIs and SWF wrappers for AngularJS services. |
| AwAdd.csv | *For internal use* |
| AwXRT.csv | *For internal use* |
| Breadcrumbs.csv | Lists all breadcrumbs used and their type. |
| CommandBarAnchorsWithServerConditions.csv | Lists all the command anchors containing commands with visibility conditions depending on the server (Active Workspace specific). |
| CommandBarAnchorsWithServerConditionsAndUsage.csv | Lists all files referencing command anchors containing commands with visibility conditions depending on the server (Active Workspace specific). |
| CommandSubPanels.csv | Lists all the command subpanels in view model files. |


Unpublished work. © 2025 Siemens

| | |
|---|---|
| ComponentTracker.csv | Lists all "components" in terms by scanning all view models and directives, and their dependencies. |
| DataMutationsFromJSON.csv | Lists all data mutations occurring in functions and which JSON files are calling those functions. |
| DirectCtxAssignments.csv | Lists all direct assignments to context properties in JavaScript files. |
| DirectDOMManipulations.csv | Lists all DOM manipulations by file, based on common APIs like createElement, setAttribute, and jQuery usages. |
| ExportedApiUsage.csv | Lists all usages of a given "Provider API" from a given "Provider Service" within a "Consuming File". |
| ModuleComplexity.csv | Lists the complexity of all SWF modules by providing some statistics on the number of lines of JavaScript, HTML, JSON, CSS files, and more. |
| StateJson.csv | Lists all the states in all states.json files providing details on their type (location, subLocation, etc,), view, and custom controller (if any) |

## Upgrading your code based on your reports

Once the reports are generated, you can leverage their data to get insight on some upgrade actions to perform on your project, as described in the following sections.

## ComponentTracker.csv

This is one of the most useful reports, as it provides information on:

- All view models and their category, first-level service dependencies, and how complex they are (in terms of LOC)
- All AngularJS directives, which would need to be **Converted to components**
- Dependencies on other components

In particular, the column listing the dependencies can be useful to locate certain Siemens Web Framework UI elements whose contract has changed and therefore are likely to require manual refactoring, such as:

- **aw-source-editor**
- **aw-file-upload**


Siemens Web Framework
Unpublished work. © 2025 Siemens

- aw-master-details-panel
- aw-panel-section and aw-command-panel-section
- aw-repeat
- aw-init
- aw-chart
- aw-checkbox and aw-radiobutton

### AngularJSReferences.csv

Use this report to understand where you need to Remove AngularJS-specific code from your project.

### StateJson.csv

Use this report to determine the changes impacting Sublocation usage such as removing or replacing custom controllers.

### DirectCtxAssignments.csv

Use this report to identify which files contain direct assignment to application context that need to be converted to use the application context API methods instead (see changes to data mutations for more information).

### AllDataMutationsInFunctions.csv, DataMutationsFromJSON.csv,

Use these two reports to review and refactor code which is performing data mutations.

### Breadcrumb.csv

Use this report to identify the files where breadcrumb components are used, as they must be refactored due to the changes to the breadcrumb contract.

### DirectDOMManipulations.csv

Use this report to identify common DOM manipulations performed used some common JavaScript or jQuery methods. You need to review all these cases and either try to remove direct DOM access or make sure that you access the DOM properly using the new patterns available.

### CommandSubPanels.csv

Use this report to identify which file contain references to Command Sub Panels, because they must be reviewed and adapted (see Changes to aw-command-sub-panel for some examples).


Unpublished work. © 2025 Siemens

### ExportedApiUsage.csv

This report contains useful details on where certain services are used. In general, it can be used to quickly identify how many files rely on some specific service or service method, which can be helpful to know when refactoring your own code. Also, it can be used to detect usage of `popupService` and replace those usages with the **new popup action** that was introduced.

### Next Steps

After going through the reports and performing the suggested upgrade actions depending on your impact, you may need to perform additional upgrade actions depending on your usage of Siemens Web Framework. For more information, see the following pages and verify whether they apply to your project or not:

- **Changes to view models**
- **Changes to Conditions**
- **Changes to command templates**
- **Changes to RESTService action errorCode object**
- **Changes to form validation**
- **Changes to data selection**
- **Changes impacting Sublocation usage**
- **Loading SVG images**
- **Migrating unit tests to Jest**

## Upgrading your project configuration

### Before you start

Before starting the upgrade procedure, make sure that your project is fully working with the latest patch of your current version. It can also be useful to keep your old project accessible and running in another location on your system to compare the files that you are going to change.

### Prerequisites

**NodeJS** v16 is needed for Siemens Web Framework 6.0.

This version of NodeJS includes NPM v8, which automatically installs peer dependencies mentioned in the `package.json` files of the libraries. This can cause error when installing the Siemens Web Framework packages. To workaround this issue, NodeJS provides a `legacy-peer-deps` option which can be passed to the `npm install` command. This will fallback to old behavior where peer dependencies will not be installed. For more information, see **legacy-peer-deps**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Tip**

You can specify the `legacy-peer-deps` option when running running `npm install` or include it in your `.npmrc` file.

## Updating the `package.json` file

In your `package.json` file:

- Remove `afx` from the `dependencies` section.
- Add `@swf/core` to the `dependencies` section, specifying the latest available version.
- Modify any package containing SWF kits that your project relies on specifying a version that is compatible with `@swf/core`.
- If specified, removed `live-server` from the `devDependencies` section. This third-party component is no longer required.
- Modify or add the NPM scripts as shown in the following example.
- Specify a `config-overrides-path` and set it to `node_modules/@swf/core/config-overrides.js`.
- Specify a `browserslist` as shown in the following example.

## Example `package.json`

```
{
    "name": "my-project",
    "version": "1.0.0",
    "description": "A sample SWF project.",
    "scripts": {
        "audit": "node node_modules/@swf/tooling/js/audit.js",
        "build": "react-scripts build",
        "clean": "rimraf out *.log *.tgz node_modules/.cache",
        "eslint": "eslint src test --ext .js",
        "generate": "node node_modules/@swf/tooling/js/generator",
        "pregulp": "node node_modules/@swf/tooling/js/createCache.js",
        "start": "react-scripts start",
        "test": "react-scripts test --watchAll=false"
    },
    "config-overrides-path": "node_modules/@swf/core/config-overrides.js",
    "dependencies": {
        "@swf/core": "next"
    },
    "browserslist": {
        "production": [
            ">0.2%",
            "not dead",
            "not op_mini all",
            "not IE <= 11"
        ],
```

```
        "development": [
            "last 1 chrome version",
            "last 1 firefox version",
            "last 1 safari version",
            "not IE <= 11"
        ]
    },
    "author": "Siemens AG",
    "license": "UNLICENSED"
}
```

### Installing dependencies

After modifying the `package.json` file:

1. Delete the`package-lock.json` file.
2. Delete or rename the `node_modules` folder.
3. Run `npm i --legacy-peer-deps`.

### Modifying other configuration files

Other configuration files within your project may need additional modifications, depending on the configuration of your original project.

#### `.eslintrc.js`

Set the contents of your `.eslintrc.js` file to the following:

```
module.exports = {
    extends: [ '@swf/eslint-config-swf' ],
    rules: {
        '@swf/swf/no-hardcoded-jsxtext': [ 'error' ],
    },
    'overrides': [
        {
            // Test files are fine to be english only
            'files': [
                '`/test/`',
                './test/**',
            ],
            'rules': {
                '@swf/swf/no-hardcoded-jsxtext': [ 'off' ]
            }
        },
    ],
};
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

### `build.json`

In the `srcPaths` section, change the path related to `afx` to point to `@swf/core` instead:

```
{
  "srcPaths": [
      "src",
      "node_modules/@swf/core/@(repo|src|test)"
  ]
}
```

## Changes to Declarative Contracts

The following pages list all the changes to Declarative Contracts for Siemens Web Framework v5.x and v6.0:

- **Changes to ViewModels**
- **Changes to Conditions**
- **Changes to command templates**
- **New popup action**
- **Changes to RESTService action errorCode object**

### Changes to ViewModels

Every view model now acts as a place for defining the **local state** of the component. The scope of this state is bound to the given view model component instance.

The following new sections have been introduced in view models to support **reusable components**:

- **ctx**
- **props**
- **selectionModels**

### Changes to conditions

The following section outlines the changes introduced as of Siemens Web Framework 5 affecting conditions.


Unpublished work. © 2025 Siemens

### Function calls in conditions

Function calls in conditions will generally continue to work, although converting to an object-based condition is suggested. For example, a condition such as the following will continue to work.

```
"ctx.selected.modelType.typeHierarchyArray.indexOf('x') > -1"
```

### Complex bracket syntax usage

Some conditions using bracket syntax (`ctx[commandContext.myKey].myFlag`) are not supported. These use cases open the possibility of changing what ctx is being used, which cannot be supported in React. See limitations of **derived state** for more information.

To support these use cases, we suggest migrating the creation of the command context to the **derived state**, simplifying the logic in the command condition while making the logic fully unit testable. For example, with this condition:

```
ctx[commandContext.viewKey].canEdit
```

We can add this ctx configuration into the view model:

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "render": {
            "method": "myRenderFunction",
            "deps": "js/myService"
        }
    },
    "ctx": {
        "commandContext": {
            "parser": {
                "method": "getCommandContextMemo",
                "deps": "js/myService"
            }
        }
    }
}
```

and this logic into a service:

```
export const getCommandContextMemo = ( vmDef, prop ) => {
    return new DerivedStateResult( {
        ctxParameters: [ prop.viewKey ],
        additionalParameters: [ prop.viewKey ],
        compute: ( { ctx }, viewKey ) => {
            return ctx[viewKey];
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        }
    } );
};
```

This will result in a new commandContext property on the local ctx that can be passed into the command bar component:

```
<aw-command-bar anchor="myAnchor" context="ctx.commandContext"></aw-command-
bar>
```

### Object-based condition reuse

Object-based conditions will only be able to compare the result of another condition to a boolean. In this example:

```
{
    "expression": {
        "$and": [
            {
                "$source": "ctx.mselected.length",
                "$query": {
                    "$eq": 1
                }
            },
            {
                "$source": "conditions.isArm1PasteObjectAsChildCmdVisible",
                "$query": {
                    "$eq": true
                }
            },
            {
                "$source": "conditions.isVRSpecificCopiedObject",
                "$query": {
                    "$eq": false
                }
            },
            {
                "$source": "ctx.awClipBoardProvider",
                "$query": {
                    "$source": "modelType.typeHierarchyArray",
                    "$query": {
                        "$in": "{{ctx.selectedModelTypeRelations}}"
                    }
                }
            }
        ]
    }
}
```


Unpublished work. © 2025 Siemens

Reusing the `isArm1PasteObjectAsChildCmdVisible` and `isVRSpecificCopiedObject` will continue to work as they compare the result of that condition to a boolean. A case such as the following is no longer supported:

```
{
    "expression": {
       "$source": "conditions.isArm1PasteObjectAsChildCmdVisible",
       "$query": {
           "$eq": 1
        }
    }
}
```

Cases that check multiple conditions within a single `$source` are no longer supported.

```
{
    "expression": {
       "$source": "conditions.isArm1PasteObjectAsChildCmdVisible ||
conditions.myRefCondition",
       "$query": {
           "$eq": true
        }
    }
}
```

## Changes to command templates

In previous versions, command templates were defined using an HTML `div` tag in a string directly the `commandsViewModel.json`. Additionally, CSS classes and `visible-when` attributes could be specified directly on the `div` tag:

```
{
    "template": "<div class='aw-ngp-discontinuedSubElementsBadge' visible-
when=\"ctx.ngp.scopeObject.status.amountOfDiscontinuedSubElements > 0\"
>{{ctx.ngp.scopeObject.status.amountOfDiscontinuedSubElements}}</div>"
}
```

Now, command templates are more declarative and they accept the template text, CSS class, and `visible-when` all as key/value JSON pairs, like this:

```
{
    "template": {
        "text":
"{{ctx.ngp.scopeObject.status.amountOfDiscontinuedSubElements}}",
        "class": "aw-ngp-discontinuedSubElementsBadge",
        "visibleWhen": {
            "condition": "conditions.someCondition"
```


Unpublished work. © 2025 Siemens

```
        }
    }
}
```

For a simple template where just text is needed, you can just enter text without wrapping it in a `div` tag:

```
"template": {
    "text": "{{ctx.ngp.scopeObject.status.amountOfDiscontinuedSubElements}}"
}
```

## New popup action

A new **popup action type** was introduced to be used instead of `popupService`.

To migrate your code to use this new action type, you must either:

- Run a dedicated script to automatically migrate all popup usages in views and view models if you referenced `popupService` methods directly in your actions.
- Manually modify your JavaScript code if you used `popupService` in your actions.

> **Tip**
>
> As a general rule, it is recommended to use trigger popups using the new Popup action defined in view models, rather than explicitly calling it from JavaScript code.

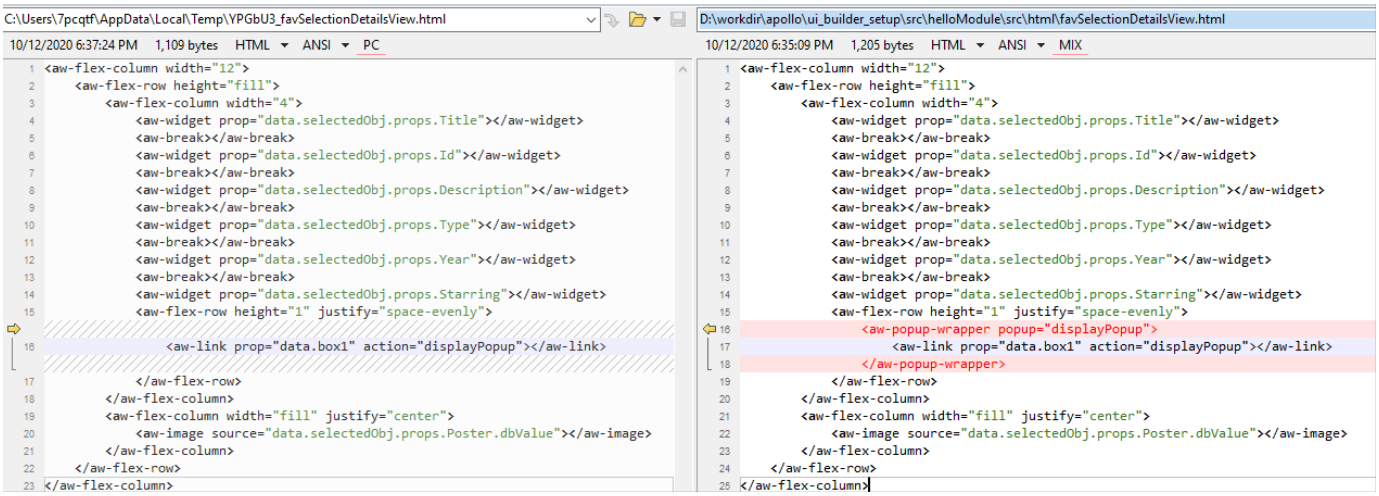
### Script to migrating views and view models

To automatically migrate your views and view models to use the new Popup action, run the following command from the root of your project:

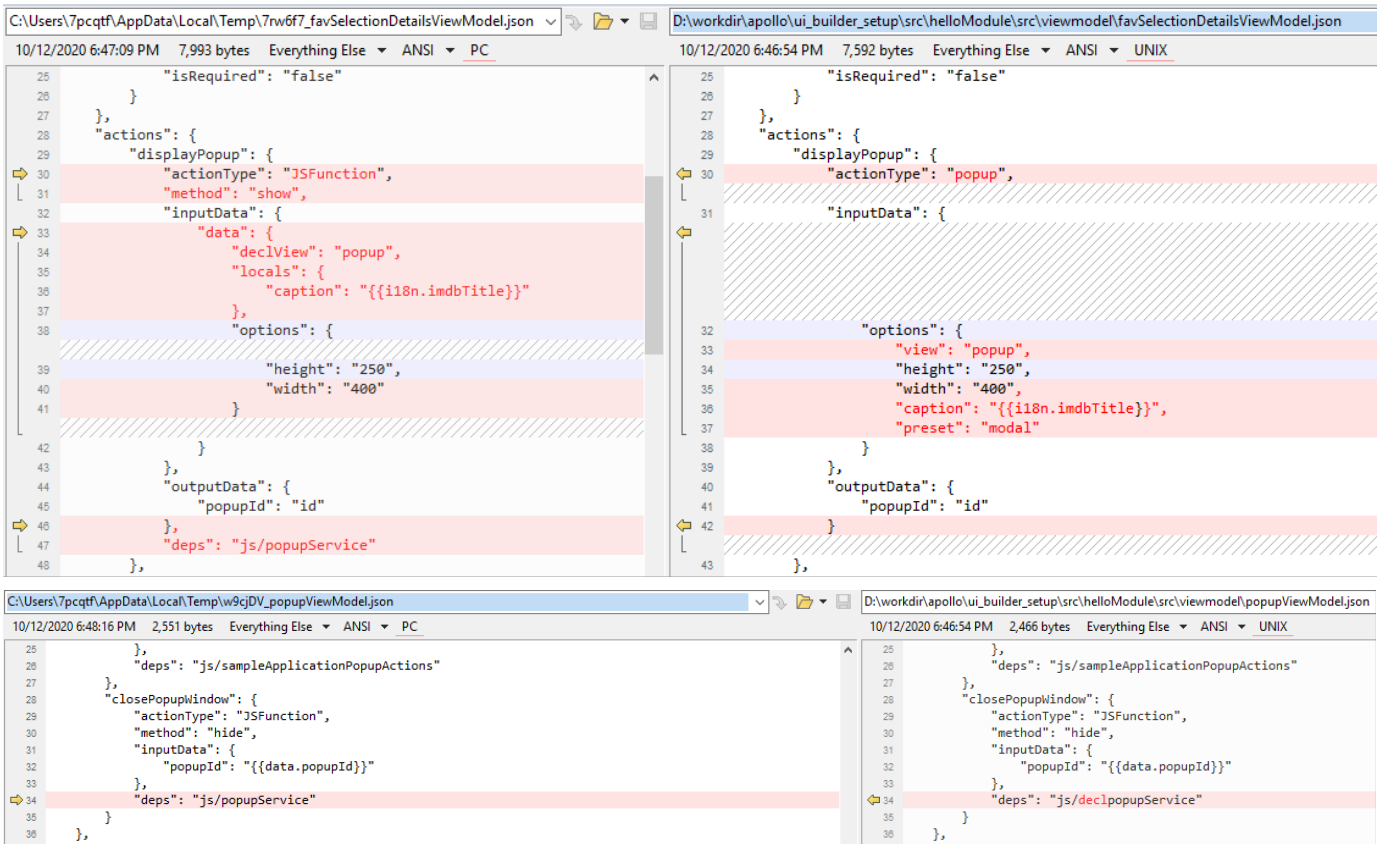
```
node node_modules/@swf/tooling/tool/migratePopup.js
```

Here's an example of the changes made by the script in a view:


Unpublished work. © 2025 Siemens




And in the corresponding view model:

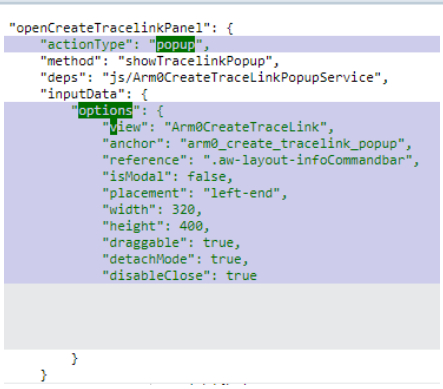
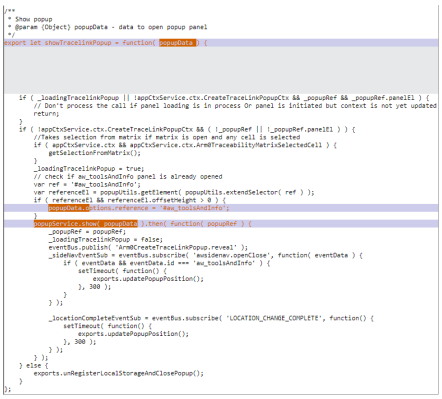
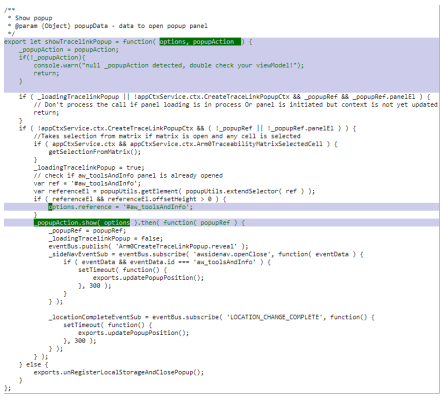



## Migrating JavaScript code

If you used `popupService` in your JavaScript code, you must define a new Popup action specifying the method you used to display the popup.


Siemens Web Framework
Unpublished work. © 2025 Siemens

When the method is called, a second object parameter (`popupAction` in the following example) will be automatically passed to it by the Popup action. This object has the same role of `popupService` and can be used to show and hide your popup:

| Usage Type | Old | New |
|---|---|---|
| Popup Action | |  |
| JavaScript Code |  |  |

## Changes to RESTService action errorCode object

In previous versions of the framework, the response of a RESTService action was accessible when configuring `actionMessages` via the `errorCode.data` object, like this:

```
{
    "actions": {
        "TestGetRequest": {
            "actionType": "RESTService",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/my-api"
                }
            },
            "actionMessages": {
                "failure": [
                    {
```

```
                        "message": "genericError",
                        "condition": "errorCode.data.code !== 'ETIMEDOUT'
&& errorCode.data.code !== 'ENOTFOUND'"
                    },
                    {
                        "message": "timeoutError",
                        "condition": "errorCode.data.code === 'ETIMEDOUT'"
                    },
                    {
                        "message": "notFoundError",
                        "condition": "errorCode.data.code === 'ENOTFOUND'"
                    },
                ]
            }
        }
    }
}
```

Now, the same data must be accessed via `errorCode.response.data` instead:

```
{
    "actions": {
        "TestGetRequest": {
            "actionType": "RESTService",
            "inputData": {
                "request": {
                    "method": "GET",
                    "url": "/my-api"
                }
            },
            "actionMessages": {
                "failure": [
                    {
                        "message": "genericError",
                        "condition": "errorCode.response.data.code !==
'ETIMEDOUT' && errorCode.response.data.code !== 'ENOTFOUND'"
                    },
                    {
                        "message": "timeoutError",
                        "condition": "errorCode.response.data.code ===
'ETIMEDOUT'"
                    },
                    {
                        "message": "notFoundError",
                        "condition": "errorCode.response.data.code ===
'ENOTFOUND'"
                    },
                ]
            }
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

Additionally, note that the `errorCode.response` object *must* have the following properties defined:

- data
- status

## Removing AngularJS-specific code

One of the most important **best practices** to follow when creating applications with Siemens Web Framework is to avoid using APIs from the underlying JavaScript framework. While this has always been a best practice, it has never been enforced at framework-level so it was always possible to use AngularJS-specific code in certain situations.

The following table lists the most common AngularJS technology leaks in each different artifact:

| Artifact | AngularJS technology leaks |
|---|---|
| Services | AngularJS *dollar-services*, like `$scope`, `$watch`, `$digest`, `$on`, `$broadcast`, `$compile`, `$timeout`, `$destroy`, `$injector`, `$evalAsync`, `$parent`, `$applyAsync`, `$locationChangeStart`, `$stateChangeStart` |
| Views | Attributes starting with `ng-` |
| AngularJS Service wrappers that were not *de-angularized* but are still used in code | `AwRootScopeService`, `AwCompileService`, `AwInjectorService`, `AwSceService` |

If your code contains any of those keywords, it must be refactored to remove them and re-tested.

### Mapping of AngularJS constructs to Siemens Web Framework equivalents

The following table lists some common AngularJS-specific construct and their corresponding 1-to-1 replacement or pattern to use instead.

| AngularJS construct | Siement Web Framework equivalent |
|---|---|
| `ng-if` | `exist-when` / `visible-when` |
| `ng-class` | `aw-class` |
| `ng-style` | `style` |
| `ng-repeat` | `aw-repeat` |
| `ng-model` | Use property widgets with **fields** |
| `ng-show` / `ng-hide` | `show-when` |


Unpublished work. © 2025 Siemens

| | |
|---|---|
| `ng-disabled` | `enable-when` |
| `ng-click` | `onClick` callback (JSX) |
| `ng-keydown` | `oneKeyDown` callback (JSX) |
| `ng-switch` | `switch` statement (JSX) |
| `ng-transclude` | `aw-transclude` |
| `$locationChangeStart`, `$stateChangeStart` | See **Reacting to state changes** |

## Converting AngularJS directives to components

Before Siemens Web Framework 5, there was no way to implement **UI elements** in a framework-agnostic way. While Siemens Web Framework already provided a comprehensive list of UI Elements to implement many different kinds of UI applications, in some cases they weren't enough to satisfy specific business needs, and when that happened the only possible way to extend the framework was to implement an AngularJS directive.

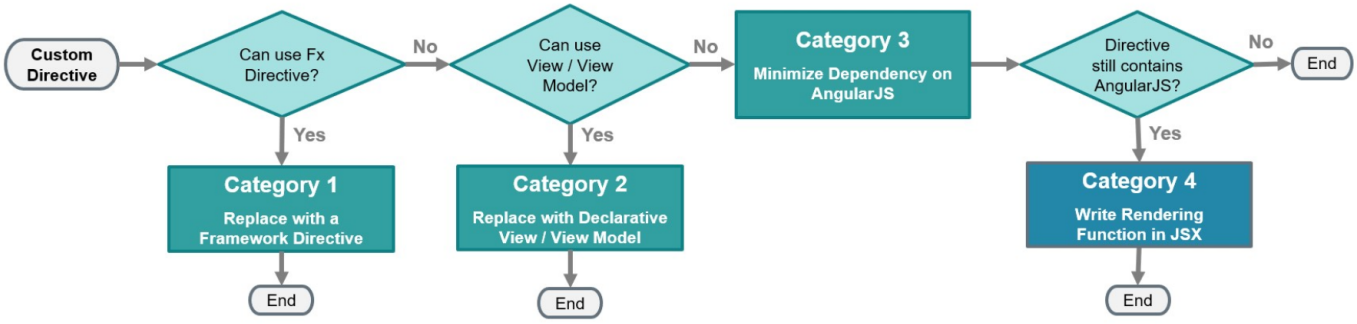
As part of the **Beyond Angular** transition, all directives in Siemens Web Framework has been converted into **components**. If you developed custom directives for your application, then you have to perform a similar conversion.

To help with directive conversion, we propose the angularJS directives be categorized into the following categories:

| Category | Expected outcome of the category |
|---|---|
| Category 1 | Replace it with a framework directive |
| Category 2 | Replace it with a declarative view / view model |
| Category 3 | Minimize dependency on AngularJS & further isolate AngularJS code - Prep action to Category 4 |
| Category 4 | Directives that are to be treated as part of Beyond AngularJS |

The following flowchart summarizes this approach to directive conversion


Siemens Web Framework
Unpublished work. © 2025 Siemens




## Opening questions

### Where and how the directive is being used as a *black box*?

Before attempting to convert a directive, you should determine where data is coming from and how it is flowing. Does the page have a sound component design based on where a directive is being used?

### How hard is the conversion going to be?

The following things can make the conversion of your custom AngularJS directive into a component:

- No well-defined interface and no implementation hiding
- Two-way data binding
- Depending on application context for things that should not be Singletons
- Listening to named events and or broadcasting named events
- Depending on secondary effects of AngularJS dirty-checking using scope and digest cycles (for example, mutating objects contained in an `ng-scope` to affect UI outside published API)

### Have all the dependencies been converted to view/view models?

Start working from the bottom (*leaf* elements of the page) and work your way up by converting directives into views and view models.

## Conversion strategies and tips

The following sections contain some tips on how to approach the conversion of AngularJS directives into components.

### Break the UI into a component hierarchy

- Take a UI mockup or an existing screenshot, draw boxes around every component, and give them all unique names

- Apply the single-responsibility, encapsulation, and loose coupling principles

### Build a static (read-only) version using declarative artifacts

- Define a view model for each component
- Don't use any state at all. Just pass data from parent to child as props
- Implement a view Template or render function to render

### Identify the minimal (but complete) representation of the UI state

- The key principle is DRY: Don't Repeat Yourself
- Ask three questions about each piece of data:
  - Is it passed in from a parent via props? If so, it probably isn't state.
  - Does it remain unchanged over time? If so, it probably isn't state.
  - Can you compute it based on any other state or props in your component? If so, it isn't a state.

### Identify where your state should live in the component hierarchy

- For each state, which component mutates or owns this state?
- For each piece of state in your application:
  - Identify every component that renders something base on that state
  - Find a common owner component, usually the lowest common parent component
  - If you can't find a common component which it makes sense to own the state, create a new component solely for holding the state and add it somewhere in the hierarchy above the common components

### Add inverse data flow (callback functions)

Pass component state modification functions down to the components that need to modify the state

### Converting a directive

The following section describes in more detail some of the key steps for converting an AngularJS directive into a Component.

### Create a view model for the directive

1. Name it using PascalCase, e.g. `AwTextBoxViewModel.json`.
2. Upon analysis, decide if you have any JavaScript object props that require conversion (for example, prop, action, etc.). Add them to the `props` section in the view model.


Siemens Web Framework
Unpublished work. © 2025 Siemens

3. If your component needs to be stateful, add it to the `data` section of the view model.
4. Register Application Context usage in the `[ctx](./declarative_view_model_ctx.md)` section of the view model.

## Decide how you will render your new view model

If your directive is fairly simple, doesn't perform DOM manipulation and it doesn't interact with the DOM directly, create a declarative view template:

1. Create a view, e.g. `AwTextBoxView.html`.
2. Copy the contents over from the directive template into the new view.
3. Replace the out-of-the-box AngularJS directive usage with equivalent Siemens Web Framework components.

Otherwise, create a codeful render function:

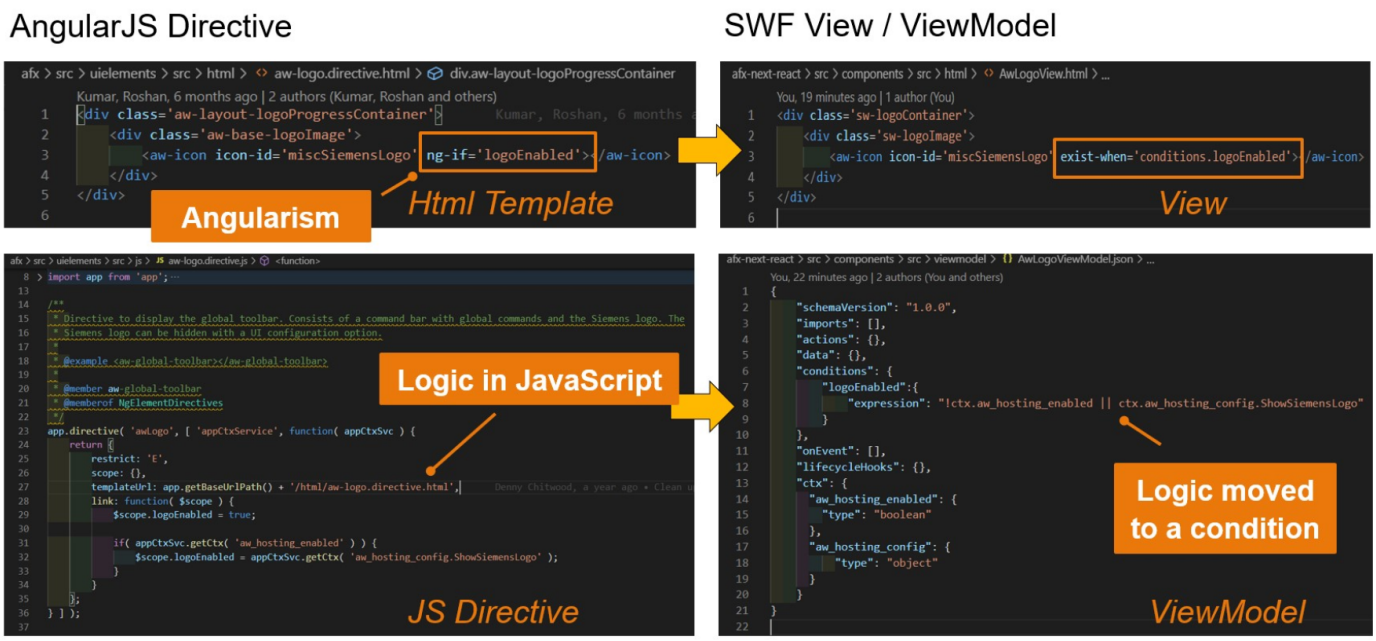
1. Provide an implementation of your render function, and use **JSX** syntax to render its HTML structure, but *do not modify component state* inside the render function.
2. Place it in a service and update the view model render action to point to where the render function lives.
3. Copy and convert the contents over from the directive template to JSX. You can also try **this online tool**, for example.
4. Replace the out-of-the-box AngularJS directives with JavaScript (if/map/functions/expressions).
5. Add imports for any dependencies, e.g. any view model containing the definition of other UI Elements, other services etc.
6. Add actions for any required **lifecycle hooks** if needed, for example:
   - `OnMount`, for requesting data asynchronously on load
   - `OnUpdate`, for requesting data asynchronously based on changes of the dependent state or props (think of this like an AngularJS `$watch`).

## Conversion example: `aw-logo`

Here is an example of a basic directive conversion with the directive files (HTML and JS) on the left-hand side, and the converted view and view model on the right-hand side.

This directive was converted by removing the angularism `ng-if` with the Siemens Web Framework equivalent (`exists-when`). Also, instead of depending on a boolean object placed on the scope, it now depends on a defined declarative condition.


Unpublished work. © 2025 Siemens

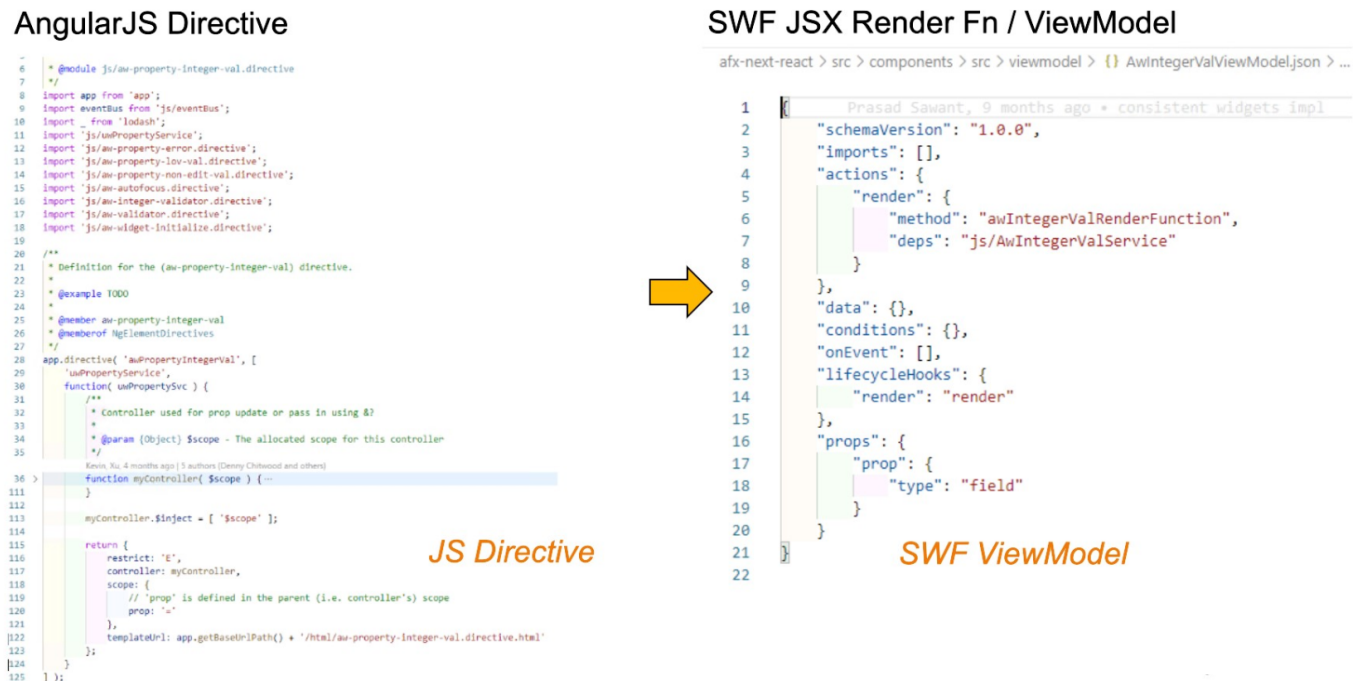



## Conversion example: `aw-property-integer-val`

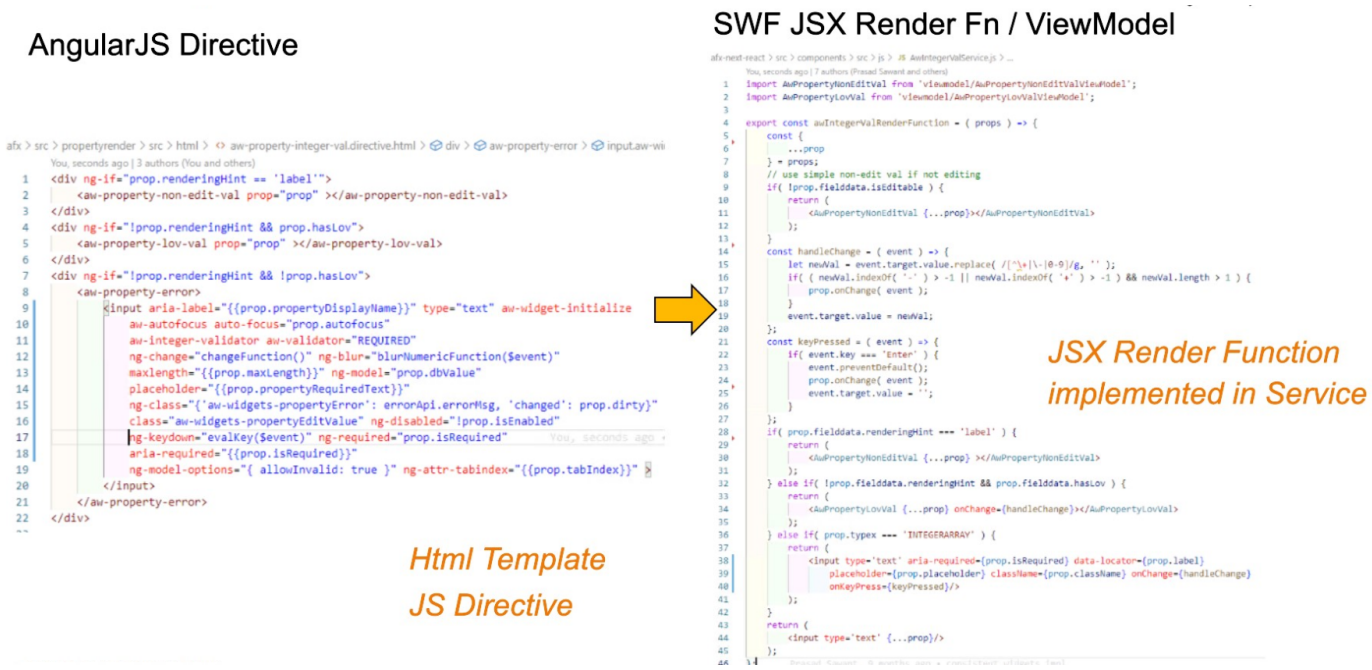
The following directive conversion example has been broken down into two different images.

The first one shows the conversion of the directive JavaScript file into the corresponding view model. Please note that the controller for the directives is currently collapsed. This example uses a render function leveraging JSX syntax. The view model file points to the render function (referenced via the `render` lifecycle hook) and defines the interface to this new component by defining `prop` as one of the attributes it takes in as a type `field`.


Siemens Web Framework
Unpublished work. © 2025 Siemens




In the following image, the conversion is completed by implementing the render function that takes on both the controller's role from the directive and view template.




## Special considerations

The following sections provide some additional suggestions on how to deal with recurring patterns while converting AngularJS directives into components.

## Tips on implementing render functions

- Pass view model properties as fields to Property Widgets when using JSX. The view compiler automatically does this for you.
- Your view model has workable fields
- Pass Actions as objects, not strings

Example:

```
export const AwTestWidgetRenderFunction = ( props ) => {

    // Retrieve fields and actions from the props object
    const { fields, actions } = props;

    // ...

    // Pass fields and actions to a property widget
    return (
        //...
        <AwCheckbox
            {...fields.childAEnabled}
            action={actions.toggleChildAEnabled}>
        </AwCheckbox>
        // ...
    )
}
```

## Dealing with two-way data binding

- If you need to modify the value of a passed-in property, you must have a function passed as property too.
- A dispatch function of the view model owner of the state is acceptable.
- Do not do this mechanically. Think about how the new component should work as a re-usable component, where the state lives, and what properties (props) should be passed in.

## Dealing with AngularJS scope side effects

In AngularJS, if an object was placed on a scope, then wherever it was modified after a digest cycle, that new updated value will be reflected in the UI.

In the new component-based architecture, your component will automatically get re-rendered when state or passed in props change.

Remember that:


Siemens Web Framework
Unpublished work. © 2025 Siemens

- The *only valid way* to update the view model state is through its `dispatch()` function.
- The *only valid way* to update Application Context is through the `js/appCtxService` API.

For more information, see **Changes to data mutations**.

## Changes to Data Management

The following pages list all the changes to data management introduced with Siemens Web Framework v5 and v6:

- **Changes to data mutations**
- **Changes to form validation**
- **Changes to data selection**
- **Automatic field creation**

### Changes to data mutations

Any data modeled as either **application state** or component-level state (e.g. data, dataproviders, view modelCollections) may only be mutated through appropriate API. Do not directly assign or change the data to be passed.

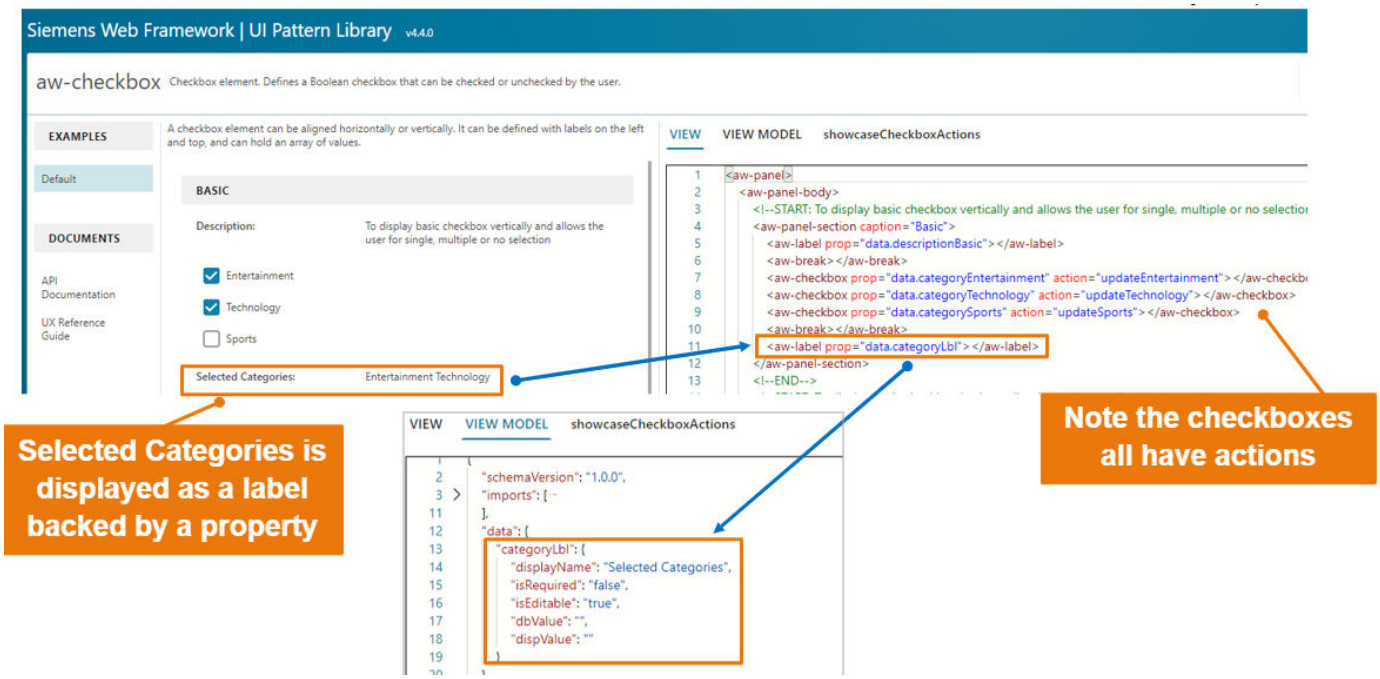
Every view model has a `dispatch()` function that you may invoke to modify the component-level state on the view model. Similarly, to modify the application-level state, you can use the published API exposed by the `appCtxService`.

In addition, any declarative output definition in view models will automatically call the view model's `dispatch()` function. Besides modifying the state, this function will trigger the re-render of the affected component (view model).

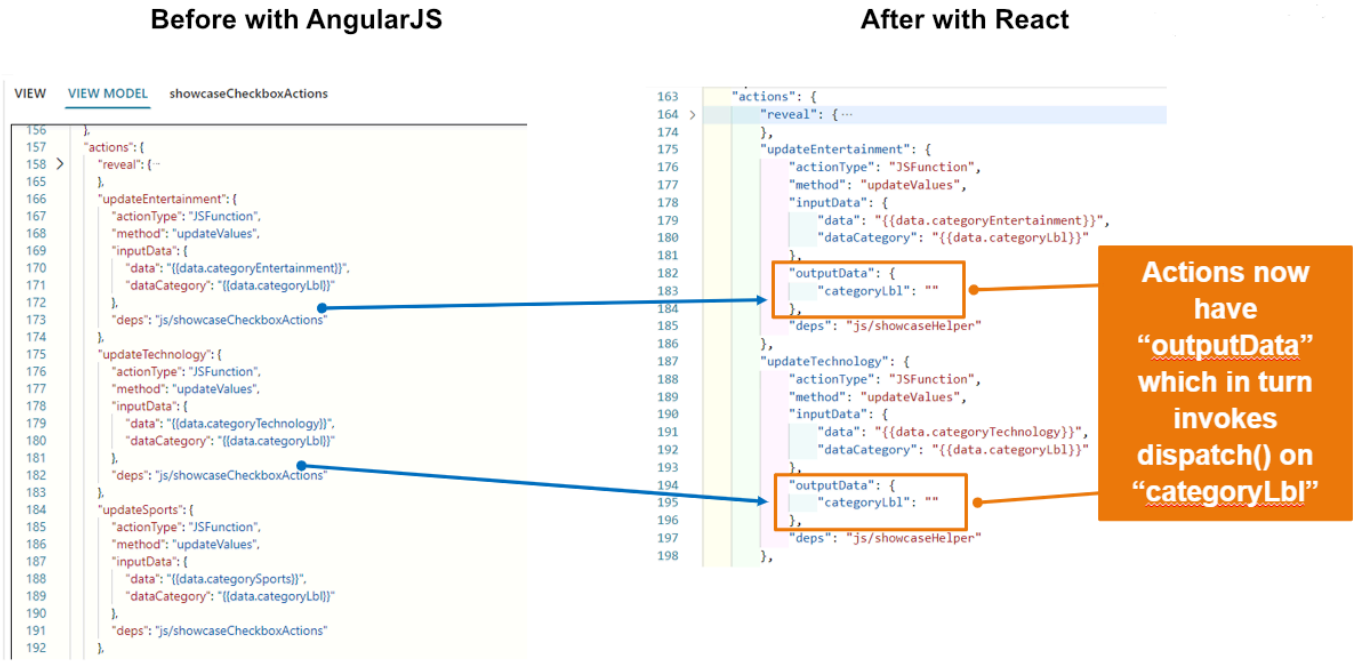
In the following example, data from a view/view model was directly mutated through an assignment. This example shows view model data, `categoryLbl`, being directly modified in a JavaScript function, `updateValues()`, through declarative actions associated with the checkBoxes. The main fix consisted of two things:

1. In the declarative action, define output object sibling to input objects which defines the view modelProperty to be assigned.
2. In the JavaScript function, treat the data *immutable*, copy it or clone it, modify the new version, and then return it.


Unpublished work. © 2025 Siemens

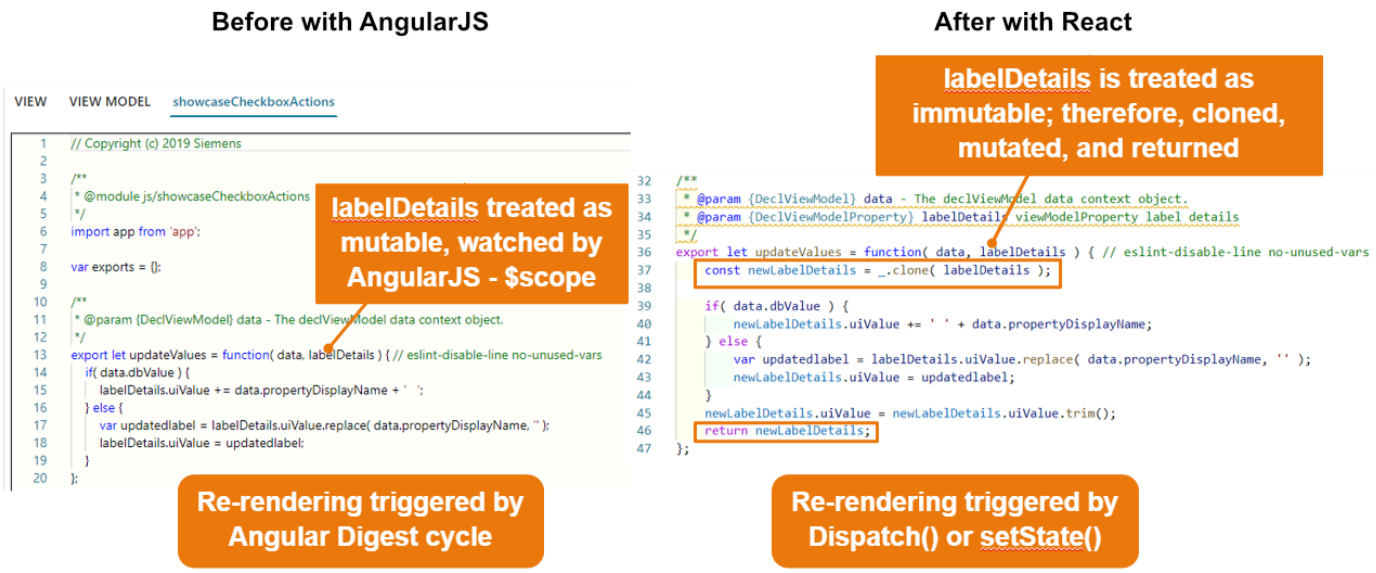



First, add an output object referencing the declarative view modelProperty being assigned:




Then, modify the JavaScript function to treat the passed in view modelProperty as immutable. This means creating a new copy of the passed-in view modelProperty, modifying the copy, and returning it. The

declarative engine will then invoke the appropriate `dispatch()` function to modify the view model's view modelProperty defined in the output section triggering a re-render.




## Changes to form validation

In previous versions, AngularJS constructs like `$valid` and `$dirty` were used to monitor the state of the form and input fields. As of Siemens Web Framework version 5, the complete form validation has been modified and enhanced and the `AwPanel` and `AwCommandPanel` as well as `AwForm` are implemented with an HTML form and it now supports the following functionalities:

| Declarative Construct | Purpose |
|---|---|
| `formProp.attributes.valid` | Specifies if the complete form is valid based on the individual validations performed on the input fields |
| `formProp.attributes.dirty` | Specifies if the form is dirty or has been modified |

Additionally, the `aw-form-validator` custom attribute is no longer supported and no longer needed.

### Example

The following example shows how the code used for validating a form must be updated when upgrading to Siemens Web Framework version 5.

```
thinclient > validationcontractjs > src > html > Crt1CreateValidationContractView.html > ...
<aw-command-panel caption="i18n.Create_AR_Panel_Title" anchor="data.
addObjectPanelPinCommands" context="data">
    <aw-form name="commandPanelForm" visible-when="data.
    activeView!='assignProjectSub'&&data.activeView!='addReferenceSub'">
        <aw-panel-body>
            <aw-add max-recent-count='3' visible-tabs="new"
            type-override-id="CreateAR" include-types="Crt0VldnContract">
            </aw-add>
        </aw-panel-body>
    </aw-form>
    <aw-checkbox prop="data.openOnCreate" visible-when="conditions.
    isNewTabAddButtonVisible" enable-when="data.pinnedToForm.dbValue === true"></
    aw-checkbox>
    <aw-panel-footer visible-when="data.activeView!='assignProjectSub'&&data.
    activeView!='addReferenceSub'">
        <!-- panel-footer button for New tab -->
        <aw-button action="initCreateInputs" visible-when="formAttribute.$valid &&
        conditions.isNewTabAddButtonVisible">
            <aw-i18n>i18n.Create_Button_Title</aw-i18n>
        </aw-button>
    </aw-panel-footer>
</aw-command-panel>
```

Siemens Web Framework 4 and older

```
1  <aw-command-panel caption="i18n.Create_AR_Panel_Title" anchor="data.
   addObjectPanelPinCommands" context="data">
2      <aw-form name="commandPanelForm" visible-when="data.
       activeView!='assignProjectSub'&&data.activeView!='addReferenceSub'">
3          <aw-panel-body>
4              <aw-add max-recent-count='3' visible-tabs="new"
               type-override-id="CreateAR" include-types="Crt0VldnContract">
5              </aw-add>
6          </aw-panel-body>
7      </aw-form>
8      <aw-checkbox prop="data.openOnCreate" visible-when="conditions.
       isNewTabAddButtonVisible" enable-when="data.pinnedToForm.dbValue === true"></
       aw-checkbox>
9      <aw-panel-footer visible-when="data.activeView!='assignProjectSub'&&data.
       activeView!='addReferenceSub'">
10         <!-- panel-footer button for New tab -->
11         <aw-button action="initCreateInputs" visible-when="formProp.attributes.valid
           && conditions.isNewTabAddButtonVisible">
12             <aw-i18n>i18n.Create_Button_Title</aw-i18n>
13         </aw-button>
14     </aw-panel-footer>
15 </aw-command-panel>
16
```

Siemens Web Framework 5 and newer

**Tip**

For more information, see **Form Validation**

## Changes to data selection

A new broadcast mechanism for the component having the source of selection to announce its selections to other components has been introduced via **selectionData**.

If a component with a **data provider** needs to notify that selection was changed to its parent components, it is necessary to specify the `selectionData` object via `inputData`.

```
{
  "dataProviders": {
    "performSearch": {
      "action": "getMovies",
      "response": "{{data.searchResults}}",
      "totalFound": "{{data.totalFound}}",
      "selectionModelMode": "single",
      "inputData": {
        "selectionData": "{{subPanelContext.selectionData}}"
      }
    }
  }
}
```

### `hosting.changeSelection` event

In previous version of the framework, the `hosting.changeSelection` event was used to select items programmatically. From Siemens Web Framework 5 onwards, this event can no longer be used this way, and programmatic selection must be implemented using the `setSelection` method of the **selectionModel** object instead.


Siemens Web Framework
Unpublished work. © 2025 Siemens

#### *dataProviderName*.lovValueChanged event

From Siemens Web Framework 5, although this event is still fired when a new value is selected from a LOV UI element, it is fired before the update is fully completed. Therefore should no longer be used to trigger actions when an item is selected.

Instead, you should configure an `onUpdate` **lifecycle hook** to trigger an action when the value of the specific property changes, like this:

```
{
  "lifecycleHooks": {
    "onUpdate": [
      {
        "action": "myHandlerAction",
        "observers": [
          "viewModel.data.myLOVProperty.newValue"
        ]
      }
    ]
  }
}
```

## Automatic field creation

As of Siemens Web Framework 5, **fields** have been introduced to manage the **changes to data mutations** and enable **inverse data flow** for **view model property** objects, typically in conjunction with property widgets like `aw-widget`, `aw-textbox`, and so on.

Technically, every property widget previously accepting a view model property object as its `prop` attribute now expects a field object instead.

```
<aw-widget prop="data.myName"></aw-widget>
```

However, the conversion from view model property objects to field objects is managed automatically by the framework, so typically no changes are required to your code as long as a field objects is created.

#### Requirements for automatic field creation

The most important thing to remember is in which cases a Field object is created. Field objects are created:

- For each a view model property object that is defined at the top level within the `data` section of a view model.
- For each view model property object that is part of a **view model object** that is defined at the top level within the `data` section of a view model.


Unpublished work. © 2025 Siemens

- For each view model property object that is part of an array that is defined at the top level within the `data` section of a view model.

Note that this applies for view model property objects defined statically in the data section (see the following example) and view model property objects, arrays of view model property objects, and view model objects created in different ways (e.g. programmatically or via a **data parser**). The view model property objects are saved directly at the top level of the `data` section, for example via the `outputData` mapping of an action.

**Tip**

If you want more control over field creation, or you need to manage complex object hierarchies containing fields at different levels, you should use **atomic data**.

### Examples

Consider the following `data` section of a view model:

```
{
    "data": {
        "prop1": {
            "type": "STRING",
            "dbValue": "test",
            "displayName": "Property #1"
        },
        "_prop2": {
            "type": "STRING",
            "dbValue": "test2",
            "displayName": "Property #2"
        },
        "myObj": {
            "prop3": {
                "type": "STRING",
                "dbValue": "test3",
                "displayName": "Property #3"
            }
        }
    }
}
```

In this case:

- The field `fields.prop1` will be created automatically for `data.prop1`.
- No field will be created for `_prop2` because it starts with an underscore, and no view model property is created.
- No field will be created for `prop2` as it is not defined at the top level directly on data (and no view model property is created either).


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Changes to routing and layout

The following pages list all the changes to routing and layout introduced in Siemens Web Framework v5 and v6:

- **Reacting to State Changes**
- **Changes impacting Sublocation usage**

## Reacting to state changes

`$locationChangeStart` and `$stateChangeStart` are no longer supported. The alternative is to use the `onBefore` hook as shown below.

Before:

```
eventBus.subscribe( 'locationChangeStart', ( eventData ) => {...} );
```

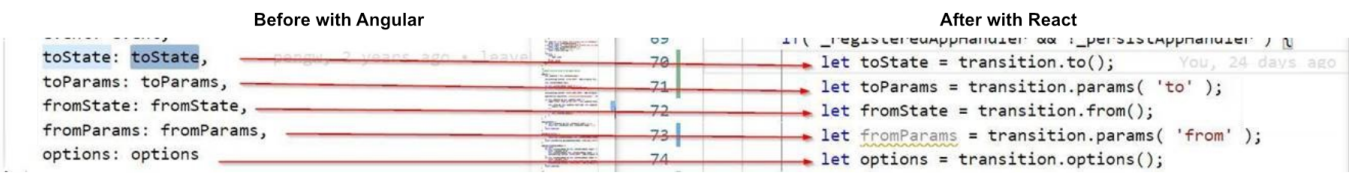
After:

```
    AwStateService.instance.transitionHooks.onBefore( {}, ( transition ) =>
{
        if( isLocationChangeStartEvent( transition ) ) {
            return customLogicForLocationChangeStart( transition );
        }
        if( isStateChangeStartEvent( transition ) ) {
            return customLogicForStateChangeStart( transition );
        }
    } );
```

where `isLocationChangeStartEvent` and `isStateChangeStartEvent` can be directly imported from `leavePlace.service.js`.

```
import { isLocationChangeStartEvent, isStateChangeStartEvent } from 'js/
leavePlace.service';
```

The arguments being fetched from `eventData` from the two events can be fetched from the `transition` object in Beyond AngularJS in the following way:



If the state transition was prevented using `event.preventDefault`, the same has to be prevented in the new construct using `return false`.

### Changes impacting sublocation usage

The **locations** and **sublocations** are defined in **states.json**, a configuration defined per module basis. While there is no change in the way we would configure a location/sublocation; there are the following points that are worth calling out explicitly:

- `templateURL` or `controller` or `dependencies` is not supported. The name of the view to be loaded when the URL defined in states.json is hit needs to be defined with the name `view`.
- `data` section in states.json is still supported. However, it is recommended to define the same configuration in the data section in the view model defined in the `view` property in states.json

**Note**

Specifying `type` as `location` or `sublocation` continues to be supported. This is used to consume the default location and sublocation the Siemens Web Framework offers.

## Changes to specific UI elements

The following pages list all the changes to specific UI elements applicable for Siemens Web Framework v5 and v6:

- **Changes to aw-repeat**
- **Changes to aw-source-editor**
- **Changes to aw-file-upload**
- **Changes to aw-chart**
- **Changes to aw-checkbox and aw-radiobutton**
- **Removal of aw-master-details-panel**
- **Changes to visible-when**
- **Removal of aw-init**
- **Changes to aw-panel-section and aw-command-panel-section**
- **Changes to aw-command-sub-panel**
- **Changes to aw-navigate-breadcrumb**


Unpublished work. © 2025 Siemens

## Changes to aw-repeat

The `aw-repeat` custom attribute was re-implemented, leveraging the new view compiler to convert the syntax to new constructs automatically.

### track by $index

In previous versions, it was possible to specify `track by $index` to associate the objects in the collection with the DOM elements or associate elements by identity. As of Siemens Web Framework 5, this is typically no longer needed.

More specifically, consider the following code:

```
<aw-filter-category
    search-action="selectHierarchyFilter"
    category="item"
    aw-repeat="item:data.categories.navigateCategories">
</aw-filter-category>
```

...will internally be converted by the view Compiler to the following code, which tracks items automatically by `$index` by default:

```
{ Object.entries( AwParseService.instance( "data.categories.navigateCategori
es" )
    ( {props, data, ctx, i18n, subPanelContext, conditions, formProp,
dispatch, selectionModels} ) || [] ).map( ( [ $index, item ] ) => (
        <div key={$index}>
            <AwFilterCategory {...getFields(fields)}></AwFilterCategory>
        </div>
    )
}
```

**Note**

It is not necessary to manually remove `track by $index` from your code. Additionally, you can still use `track by` specifying other variables, and the view compiler will perform the conversion as expected.

### Avoiding complex aw-repeat usage

In previous versions, `aw-repeat` was the only way to iterate over a UI Element in a view. While `aw-repeat` is still supported and recommended for simple scenarios, you should not use `aw-repeat` anymore when:

- You have nested aw-repeat usages
- You are iterating over property widgets

- You have to dynamically display several widgets based on conditions

In these cases, the view compiler may not perform a correct conversion. Therefore it is recommended to convert your declarative view to a render function leveraging the full JSX syntax, as in the previous section's example.

## Changes to aw-source-editor

The `aw-source-editor` UI Element contract was changed from version 5 of Siemens Web Framework in terms of:

- The props needed to configure the component
- How to be notified when the editor content changes

### Interface Change

In previous versions of Siemens Web Framework, it was possible to use `aw-source-editor` in the following way:

```
<aw-source-editor name="testSourceEditor" content="data.editor.data"
config="data.config"></aw-source-editor>
```

Where `data.editor` and `data.config` could be defined as follows:

```
{
    "data": {
        "editor": {
            "data": "console.log('Hello, World!');"
        },
        "config": {
            "language": "javascript",
            "theme": "vs-dark",
        }
    }
}
```

As of Siemens Web Framework 5, the `aw-source-editor` must be instantiated as follows (in a view template):

```
<aw-source-editor name="testSourceEditor" content="data.editor"
config="data.config"></aw-source-editor>
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

...or like this (in a render function):

```
<AwSourceEditor name="testSourceEditor" {...props.fields.editor}
config={props.viewModel.data.config}></AwSourceEditor>
```

Where `editor` and `config` are defined as follows:

```
{
    "data": {
        "editor": {
            "dbValue": "console.log('Hello, World!');"
        },
        "config": {
            "language": "javascript",
            "theme": "vs-dark"
        }
    }
}
```

In particular, note that the `content` prop must now be a **field**.

### Executing actions when content change

To execute an action when the editor content is updated, you can now use the `onUpdate` hook in the component containing the editor and observing the `editor` property as follows:

```
{
    "actions": {
        "editorContentUpdated": {
            "actionType": "JSFunction",
            "method": "contentUpdated",
            "inputData": {
                "data": "{{data.editor}}"
            },
            "deps": "js/myTestService.js"
        }
    },
    "lifecycleHooks": {
        "onUpdate": {
            "action": "editorContentUpdated",
            "observers": ["viewModel.data.editor"]
        }
    }
}
```


Unpublished work. © 2025 Siemens

## Changes to aw-file-upload

In previous versions of the framework, the following UI Elements were available to provide a widget to upload files from the file system:

- aw-file-upload
- aw-file-uploader
- aw-multifile-upload

As of Siemens Web Framework version 5, all these implementations have been consolidated into a single UI Element called `aw-file-upload` which provides support for uploding single files or multiple files.

### Contract for `aw-file-upload`

The new `aw-file-upload` UI Element exposes the following props:

- `type-filter` Specify which file types can be uploaded.
- `multiple` If set to `true`, it is possible to upload multiple files (default: `false`)
- `file-change-action` Handler action that gets invoked when a file is uploaded. When this handler action gets invoked, it gets the following uploaded file information:
  - `fileName` Name of the file including extension
  - `fileNameNoExt` File name without extension
  - `validFile` If the uploaded file is valid or not
  - `formData` Formdata of the uploaded file
- `remove-tooltip` Tooltip to display the button provided to remove an uploaded file.

### Example

In previous versions of the framework, the `aw-file-upload` UI Element updated `fileName`, `fileNameNoExt`, `validFile`, and `formData` variables directly on the consuming view model.

With the new contract, it is now necessary to provide an action to execute when a file is uploaded. Consider the following example:

```
<aw-file-upload type-filter=".json" file-change-
action="updateFormData"></aw-file-upload>
```

In this case, the `updateFormData` action can be configured as follows:

```
{
    "actions": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "updateFormData": {
            "actionType": "JSFunction",
            "method": "getSelectedFileData",
            "inputData": {
                "parameters": "{{parameters}}"
            },
            "outputData": {
                "fileData": "fileData"
            },
            "deps": "js/myFileUploadService"
        }
    }
}
```

If you want to save the file data received when the file is uploaded, implement a simple pass-through method, like the following:

```
export const getSelectedFileData = (parameters) => {
    // Optionally add your logic here to process the uploaded file
    return {
        fileData: parameters.formData
    };
};
```

## Changes to aw-chart

In versions of Siemens Web Framework prior to 5, `aw-chart` only needed a single `chart-provider` prop containing all its configuration:

```
<aw-chart chart-provider="data.myChartProvider"></aw-chart>
```

As of Siemens Web Framework version 5, a new `chart-point` attribute has been added to specify the chart points of the chart, which were previously managed via the chart provider instead:

This means that chart points must be retrieved first via a dedicated action and then passed to the `aw-chart` component, which will no longer have the responsibility to update the chart points.

In the previous example, `data.myChartPoints` is treated as an internal state of the container component, which uses `aw-chart` and is therefore stored on `data`. When `data.myChartPoints` is updated via an action, the chart will be re-rendered automatically.


Unpublished work. © 2025 Siemens

## Changes to aw-checkbox and aw-radiobutton

Some changes were introduced in Siemens Web Framework version 5 that modify the behavior of these components when setting certain flags.

In previous versions of Siemens Web Framework:

- Setting the `isEditable` property of the `prop` attribute passed to `aw-checkbox` or `aw-radiobutton` has no effect.
- Setting the `isEnabled` property of the `prop` attribute passed to `aw-checkbox` or `aw-radiobutton` enables/disables the control.

From Siemens Web Framework 5 onwards:

- Setting the `isEditable` property of the `prop` attribute passed to `aw-checkbox` or `aw-radiobutton` enables/disables the control.
- Setting the `isEnabled` property of the `prop` attribute passed to `aw-checkbox` or `aw-radiobutton` has no effect.

Note, however, that the behavior of `aw-widget` hasn't changed and is consistent regardless of the underlying widget displayed, i.e.:

- If the `isEditable` property is set to `true`, an editing widget is displayed. Otherwise a label is displayed.
- If the `isEnabled` property is set to `false` and:
  - The `isEditable` property is set to `true`, and the editing widget is displayed, but it is disabled
  - The `isEditable` property is set to `false`, and a label is displayed (the `isEnabled` flag has no effect).

## Removal of aw-master-details-panel

The directive `aw-master-details-panel` has been removed. The following two alternate methods can be used to achieve the standard master details pattern:

- **Sync Strategy**
- **Atomic Data**


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Changes to visible-when

The behavior of the **visible-when** custom attribute was changed to behave like **exist-when**. In other words, if the condition specified in the **visible-when** attribute is false, the component will not be rendered at all, and it will not be present in the DOM.

This change was done to ensure better performance, as typically it is faster to avoid rendering a component rather than simply hiding it and keeping it in the DOM, especially when the component with the **visible-when** attribute has children.

> **Tip**
>
> If, for some reason, you need to hide a component instead of not rendering it at all, you can use **show-when** instead of **visible-when**.

## Removal of aw-init

The `aw-init` custom attribute is no longer supported from version 5 of Siemens Web Framework. Instead, you should perform variable initialization in the View Model.

Consider the following `aw-init` usage:

```
<aw-command-panel caption="i18n.Classify" aw-
init="data.activeView='ClassificationSub'">
    <aw-command-sub-panel panel-id="ClassificationSub" visible-
when="data.activeView==='ClassificationSub'">
</aw-command-panel>
```

In this case, `aw-init` must be removed from the view:

```
<aw-command-panel caption="i18n.Classify">
    <aw-command-sub-panel panel-id="ClassificationSub" visible-
when="data.activeView==='ClassificationSub'">
</aw-command-panel>
```

...and the relevant variable must be initialized in the corresponding view model instead:

```
{
    "data": {
        "activeView": "ClassificationSub"
    }
}
```


Unpublished work. © 2025 Siemens

## Changes to aw-panel-section and aw-command-panel-section

Unlike in previous versions, as of version 5 aw-panel-section and aw-command-panel-section are *always* collapsible by default.

To prevent sections from being collapsible and restore the old default behavior, set the collapsed prop to disabled:

```
<aw-command-panel-section caption="Test Section"
anchor="data.testSectionAnchor" collapsed="disabled" name="TestSection"
context="data">
<!-- Contents go here -->
</aw-command-panel-section>
```

## Changes to aw-command-sub-panel

In versions of Siemens Web Framework prior to 5, it was possible to access the state of the parent component (aw-command-panel) from child components (aw-command-sub-panels), and also the state of any sibling component from another sibling component.

This violates the encapsulation principles and has been eliminated from Siemens Web Framework 5 onwards.

Instead:

- If you need to access the child component state from the parent component, you need to pull the state up in the parent component and pass it to the child component via props
- If you need to access the parent component state from the child component, you need to pass the state of the parent component to the child components via props
- If you need to access the state of a sibling component, you need to move the shared state up to the parent and pass it to the children's components via props

**Tip**

For more information on how to use aw-command-sub-panel and implement navigation following these new best practices, see **Command Sub Panels**.

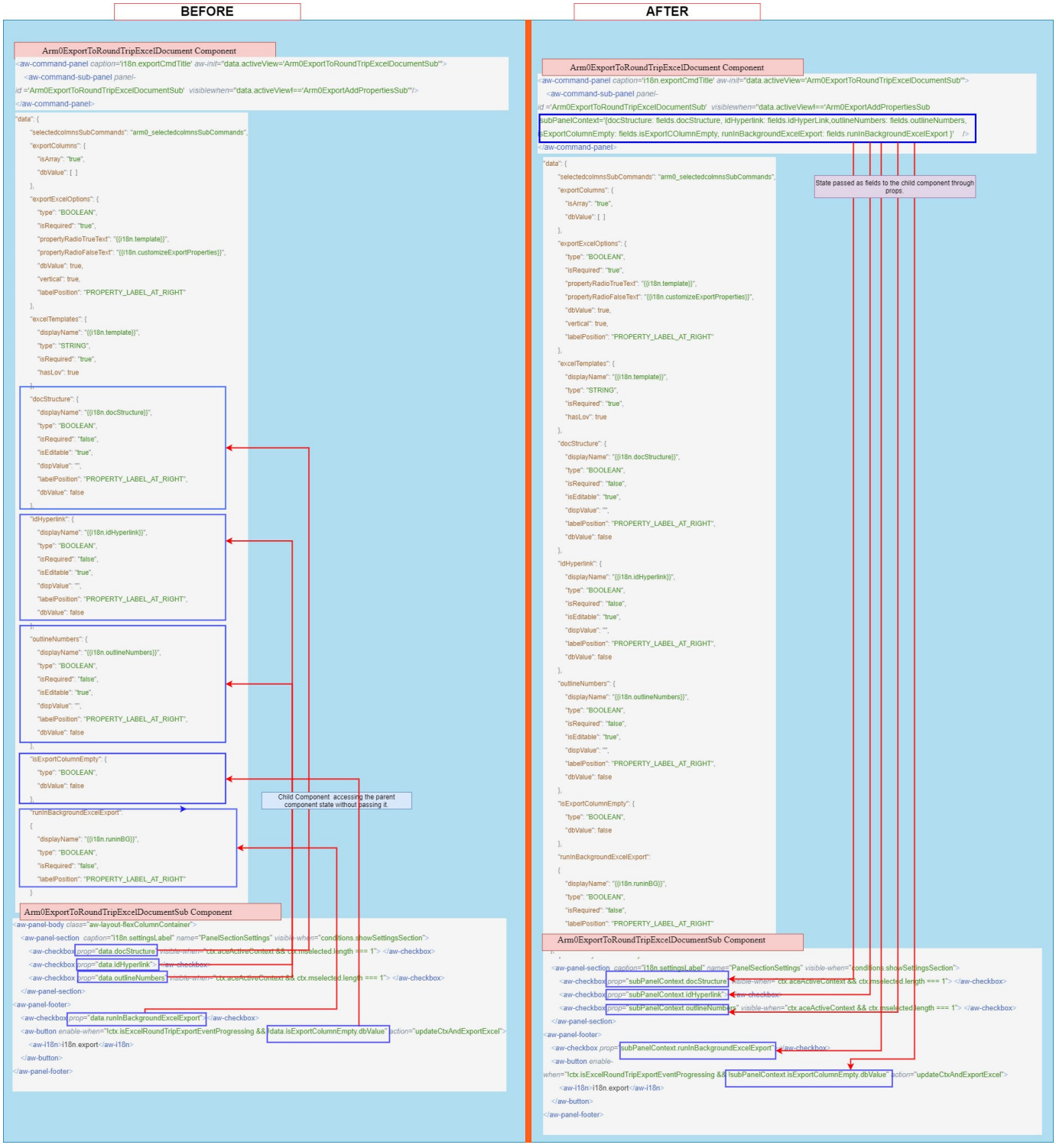
### Basic Example

The following picture shows how to refactor a basic usage of aw-command-sub-panel:


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Advanced Example

The following picture shows how to refactor an advanced usage of `aw-command-sub-panel`:




## Changes to aw-navigate-breadcrumb

As of Siemens Web Framework version 5, the `aw-navigate-breadcrumb` contract was reviewed and improved to manage cross-component state more appropriately and features an overall simplified design.


Unpublished work. © 2025 Siemens

In prior versions, a breadcrumb was instantiated as follows:

```
<aw-navigate-breadcrumb breadcrumb-
config="data.carBOMbreadcrumbConfig"></aw-navigate-breadcrumb>
```

In this case, `carBOMbreadcrumbConfig` was an object defined in the view model, such as the following:

```
{
    "data": {
        "carBOMbreadcrumbConfig": {
             "vm": "carBOMBreadcrumb",
             "crumbDataProvider": "carBOMcrumbDataProvider",
             "chevronDataProvider":
"navigationBreadcrumbchevronDataProvider",
             "popuplist": "/html/navigateBreadcrumblistView.html"
        }
    }
}
```

As of Siemens Web Framework version 5, the contract was changed as follows:

More specifically:

- The `crumbs` prop must be set to a list of crumb configuration objects, provided as an array that can be constructed with any ordinary action. There is, therefore, no need for a dataProvider for crumbs.
- The `popuplist` prop must be set to the name of a component which will be loaded in a popup when the chevron is clicked. There is, therefore, no need for a chevron dataProvider at the breadcrumb level, as the contents displayed in popups are delegated to the component specified via the `popuplist` prop.

Additionally:

- The `<breadcrumbViewModel>.selectCrumb` event is no longer supported. You can define an `onCrumbClick` function to be called in every crumb object definition, which would be invoked when a particular crumb is clicked.
- The `<breadCrumbId>.chevronClicked` event is no longer supported. You can add logic to be performed when the chevron is clicked using the `onMount` and `onUpdate` **lifecycle hooks** of the component specified via the `popuplist` prop.
- The following props will be passed to the component specified via the `popuplist` prop:
  - `selectedCrumb`, the currently-selected crumb object
  - `chevronPopup`, a reference to the popup containing the component displayed

For more details on how to configure breadcrumbs, see **this page**.

# Loading SVG images

Prior to Siemens Web Framework 6.0, all the SVG images were grouped into a single bundle and downloaded on the initial load of the application. Regardless of the image usage, about 5.5MB of data needed to be downloaded, with a significant factor being redundant images. Furthermore, developers could only access an image in string format through the `iconService`.

This approach had a few downsides:

- All SVG images were downloaded during startup
- Unused SVG images were also getting downloaded
- `iconService` was needed to access the images
- Only string format was supported
- ES6 import was not supported for SVGs
- Runtime image stitching was degrading performance

## Changes introduced in Siemens Web Framework 6.0

As of Siemens Web Framework 6.0:

- Any SVG image placed in an **src/image** module folder that *does not* start with **type** (typically type icons) will be automatically converted into a React component at build time, and it will be automatically bundled along with JavaScript source code only if it is actually used (i.e. imported in JavaScript code or used in common configuration points like commandsViewModel.json) See the following section for more details.
- Any image (not only SVG) placed in an **swf/images** module folder will be copied to the **/assets** output folder, like in previous versions. Typically, you should use this folder for any image that is not an icon or frequently-referenced SVG file.
- Any SVG image placed in an **src/image** module folder that starts with **type** (typically type icons) will be still copied to the **/assets** output folder, like in previous versions.

The benefits of this approach are the following:

- Only SVG images used in the code will be downloaded.
- The browser requests SVG images as required and not all at once, which reduces the initial data download.
- SVG images can be imported in JavaScript as ES6 import.
- SVG images can be imported as a *Component* or *String*.
- All image processing happens at build time, so no extra stitching is needed during run time.


Unpublished work. © 2025 Siemens

**Note**

There are no changes in the way SVG icons are used when an icon ID is specified in configuration ponts like `commandsViewModel.json` and similar. Still, there are changes in the way SVG images are accessed from JavaScript code, as explained in the following section.

## Restoring the old behavior

If you want to restore the old behavior and pre-load *all* icons at startup like in previous versions of the framework, add the following `imageBundles` configuration to the `solutionDef` section of your solution **kit.json** file:

```
{
    "solutionDef": {
        "solutionId": "YourSolutionName",
        "imageBundles": {
            "bundle": ["*"]
        }
    }
}
```

## Importing SVG images in JavaScript code

All SVG images can be imported into JavaScript code, including type icons.

To import an SVG image in JavaScript code, for example for using an icon as component in a render function, you can import `svgComponent` from the specified image reference, like this:

```
import AwWidget from "viewmodel/AwWidgetViewModel";
import { SvgComponent as CmdUp } from "image/cmdUp24.svg";

export const awNumericRenderFunction = (props) => {
    return (
        <div>
            <AwWidget {...prop}></AwWidget>
            <CmdUp />
        </div>
    );
};
```

Alternatively, you can also retrieve the raw SVG string by importing `svgString` from the specified image reference, like this:

```
import { svgString as cmdSettings } from "image/cmdSettings24.svg";
self.addGridMenu = function () {
    const localeGridMenuText = _splmTableMessages.gridMenu;
    const html =
        "" +
```

```
        '<div class="aw-commands-svg">' +
        '<button type="button" class="aw-commands-commandIconButton icon-
override" tabindex="-1" title="' +
        localeGridMenuText +
        '" aria-label="' +
        localeGridMenuText +
        '" >' +
        cmdSettings +
        "</button>" +
        "</div>";
    let menu = _t.util.createElement("div", "aw-splm-tableMenuButton");
    menu.innerHTML = html;
    let btn = menu.querySelector("button");
    _keyboardService.setOnFocusAndBlur(btn);
    _table.insertBefore(menu, _table.children[0]);
    loadGridMenuItems();
};
```

## Alternatives to referencing SVG image paths

In general, it is no longer possible to reference the path to an SVG image file simply because that file no longer exists as SVG images are now converted components at build time.

For example, consider the following code used in cell template definition:

```
const cell = document.createElement("div");
cell.className = "aw-splm-tableCellTop";
var cellImg = document.createElement("img");
cellImg.src = getBaseUrlPath() + "image/miscExpandedTree12.svg";
cellImg.className = "aw-base-icon aw-type-icon aw-splm-tableIcon";
cellImg.style.cursor = "pointer";
cell.appendChild(cellImg);
```

Instead of specifying the image to include via a path, you must import it as shown in the following example and include the imported SVG markup directly in the DOM:

```
import { svgString as miscInProcessIndicator } from 'image/
miscInProcessIndicator16.svg';

...
var cell = document.createElement( 'div' );
cell.className = 'aw-splm-tableCellTop';

var cellImg = document.createElement( 'span' );
cellImg.className = 'aw-base-icon aw-type-icon aw-splm-tableIcon';
cellImg.style.cursor = 'pointer';
cellImg.innerHTML = miscInProcessIndicator;
cell.appendChild( cellImg );
```

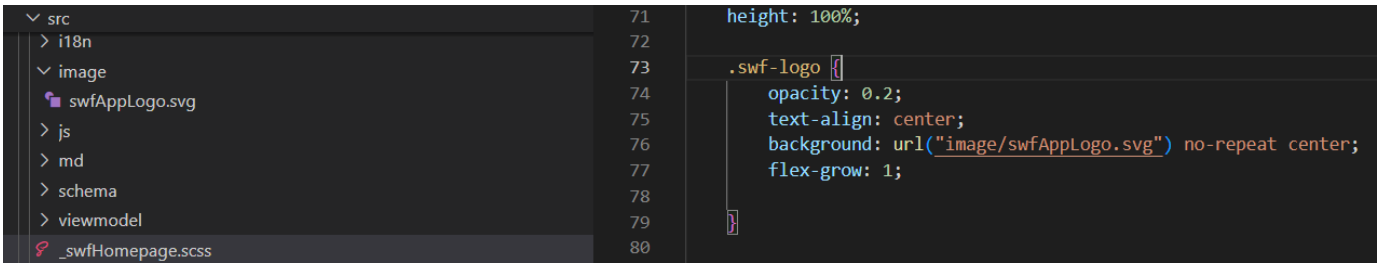

Siemens Web Framework
Unpublished work. © 2025 Siemens

**Tip**

SVG images starting with `indicator` and `type` prefixes can still be consumed via `img` tag. As a best practice, however, you should refactor your code to use imports instead.

## Using SVG images in CSS/SCSS file

To use a SVG image in CSS/SCSS code, for example for using an image as background url, you can use reltive path of that SVG icon in url, this path will be relative to css file in which you are using it, like this:




Also in this case you have to put this in your package.json

```
"homepage": ".",
```

This will make sure that all the asset paths are relative to index.html

## Using iconService

The `iconService` should only be used for functionalities related to type icons. Other icons should be directly imported into JS Files.

Below functions from iconService are made async, which will also return images in string format.

The following `iconService` methods are now asynchronous and return the raw SVG image code as a string:

- getTileIcon
- getCmdIcon
- getAwIcon
- getIcon
- getIndicatorIcon

### Build-time impact and optimizations

Since SVG images are processed at build time, build time will increase to process and bundle images. The additional time is needed to convert SVG strings into React components, and the framework is caching this transformation, so subsequent builds will use cached components instead.

To improve first-time build performance, a new **@swf/images** package containing precompiled images present inside **@swf/core** can be added as a dependency to any SWF project. This will improve first-time build timing as it will not compile framework-owned images. This optimization is enabled by default if you add **@swf/images** as a dependency.

For debugging purposes, you can set the `FORCE_IMAGE_OPTIMIZATION` to `false` to turn off this optimization before building your application.

### Runtime impact and optimizations

With this new approach, SVG icons will be downloaded one by one by default. As a result, some applications that use a lot of SVG images can increase network chattiness.

To mitigate this, define can define image bundles in the **kit.json** `solutionDef` configuration. For example, the images which are required by a page can be bundled together to reduce chattiness.

For usage examples related to specific use cases, see the following sections.

### Example: bundle only a few frequently-used images

```
{
    "solutionDef": {
        "solutionId": "showcase",
        "imageBundles": {
            "bundleA": [
                "image/cmdUser24.svg",
                "image/cmdUp24.svg",
                "image/cmdOne12.svg",
                "image/miscHead24.svg",
                "image/cmdSettings24.svg",
                "image/cmdHeader24.svg"
            ],
            "bundleB": [
                "image/cmdMask24.svg",
                "image/cmdDown24.svg",
                "image/cmdTwo12.svg",
                "image/miscUiEnd24.svg",
                "image/MiscFilter24.svg",
                "image/MiscExpan24.svg"
            ]
        }
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

In this scenario, when the browser requests any image from bundleA or bundleB, the whole bundle will be downloaded. The browser doesn't need to perform individual requests to get other images from the same bundle. The images which are not mentioned in bundleA or bundleB will be downloaded one by one.

**Note**

TA bundle name can be anything, and it will be used within the bundle file name.

### Example: Bundle all images in different bundles

```
{
    "solutionDef": {
        "solutionId": "showcase",
        "imageBundles": {
            "bundleA": [
                "image/cmdUser24.svg",
                "image/cmdUp24.svg",
                "image/cmdOne12.svg",
                "image/miscHead24.svg",
                "image/cmdSettings24.svg",
                "image/cmdHeader24.svg"
            ],
            "bundleB": [
                "image/cmdMask24.svg",
                "image/cmdDown24.svg",
                "image/cmdTwo12.svg",
                "image/miscUiEnd24.svg",
                "image/MiscFilter24.svg",
                "image/MiscExpan24.svg"
            ],
            "bundleC": ["*"]
        }
    }
}
```

In this scenario, all images not already present in bundleA and bundleB will be bundled into bundleC, and bundleC will be loaded when the browser requests any of the images from bundleC.

**Note**

In this scenario, bundleC will also contain images that are present in the application but are not used.

### Example: Bundle all images in the same bundle

```
{
    "solutionDef": {
        "solutionId": "showcase",
        "imageBundles": {
            "bundleA": ["*"]
        }
```

```
    }
}
```

In this scenario, all images will be bundled into bundleA even if they are not actually used within the application. The browser will end up downloading all images as used to happen in previous versions.

## Migrating unit tests to Jest

As of Siemens Web Framework version 5, **Jest** must be used for unit testing instead of **Jasmine**, that was used in previous version.

### Why did Siemens Web Framework migrate to Jest?

- It is still possible to use most of the Jasmine APIs in the Jest Framework. Now, Jest has all functionality available like spies, mocks, etc. therefore those should be used instead of Jasmine APIs.
- With react-scripts v4.0.3, the Jest framework was upgraded to v26. This does not support Jasmine's global `spyOn` method.
- Jest is fast and has **good documentation**.

### How to migrate tests from Jasmine to Jest

Jest provides the **jest-codemods** utility that can be used to migrate jasmine tests to jest. You can use this utility to migrate tests from other frameworks like mocha, chai, AVA, etc. The **Jest migration guide** also talks about this utility.

### How to run jest-codemods

This is explained in the `jest-codemods` **README.md file**.

- If you have `npx` then you can use the following command to execute this utility:

```
npx jest-codemods src/`/*.test.js
```

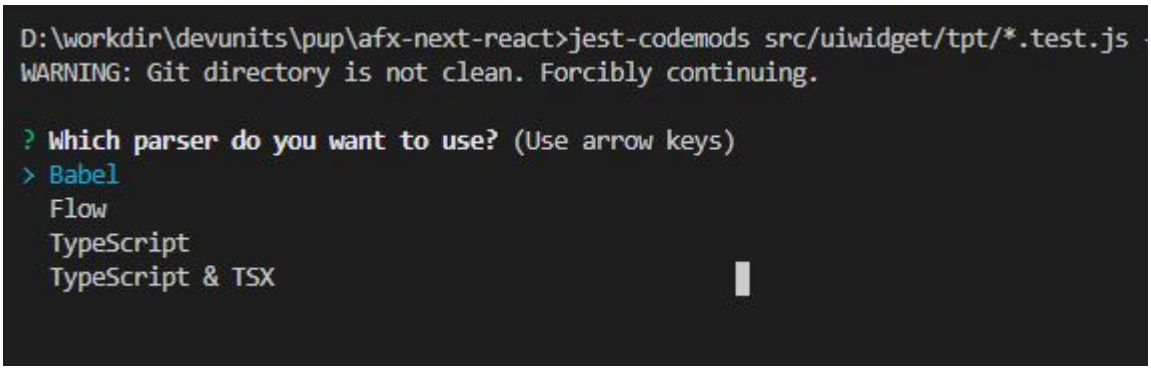
- Another option is to install `jest-codemods` globally using npm and then run it:

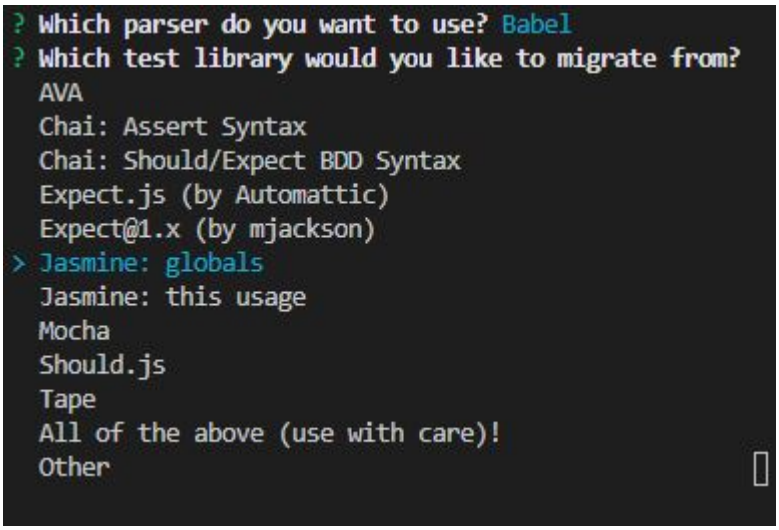
```
npm install -g jest-codemods
jest-codemods src/`/*.test.js
```

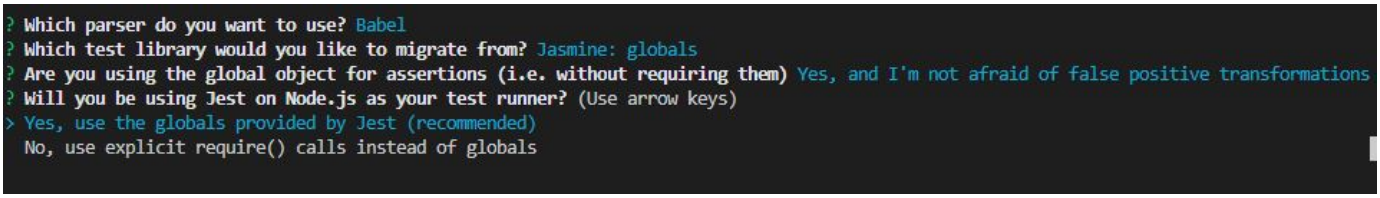
```
Note jest-codemods supports can convert a limited number of files in one
pass. If you have a large number of test files you need to process them in
chunks, e.g. folder by folder.
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

Once you run `jest-codemods`, it will ask for a few inputs; select the following options:




## Tips on how to fix tests after the migration

Although `jest-codemods` helps during the migration from Jasmine, it is often necessary to manually fix tests after the migration as not everything can be easily migrated automatically.

## Restore All Mocks

Jasmine automatically resets the spies once the test is completed and before another test start. Jest spies do not support that. This causes failures after migration. To fix this you need to do the following:

- Create a `setupTests.js` file under src folder, if you already have it then you need to use the same.

- Add the following code snippet in `setupTests.js`:

```
/* eslint-env jest */
beforeEach( () => {
    jest.restoreAllMocks();
} );
```

The previous code will be executed before each test and will restore all Jest mocks. Following Jest APIs are available:

- **jest.clearAllMocks**,
- **jest.resetAllMocks**,
- **jest.restoreAllMocks**

Within Siemens Web Framework code, `restoreAllMocks` is used, as it restores the spies so that the next call invokes the original implementation instead of mocked.

### Jasmine.spyOn Vs Jest.spyOn

When you use jasmine.spyOn, it by default blocks the original code invocation. If you want to spy in jasmine and still want to invoke the actual code then you use the following syntax:

```
spyOn(appCtxServicem, 'getCtx').and.callThrough();
```

But, Jest handles it differently. When you use jest spy for eg `jest.spyOn(appCtxService, 'getCtx')`, it invokes the original implementation by default. In order to stop that you need to use following function:

```
jest.spyOn(appCtxService, 'getCtx').mockImplementation(() =>{});
```

### Mock localStorage and sessionStorage in Jest

Jasmine supports spying localeStorage and sessionStorage objects. This is not supported by Jest. In Jest, you need to use **this workaround**

### spyOn().and.returnValues(x,y,z)

Jasmine supports multiple return values for spied functions using `and.returnValues`. When the spied function is invoked for the first time it returns x, the following time it returns y, and so on.


Siemens Web Framework
Unpublished work. © 2025 Siemens

The same thing is not supported by default in Jest. You need to write your own function to handle this.

```
const returnValues= [-1, -1, 1, -1,-1];
const lodashSpy = jest.spyOn(_, 'indexOf');
lodashSpy.mockImplementation(() => {
   return returnValues[ lodashSpy.mock.calls.length - 1 ];
} );
```

### obj.mock.calls.mostRecent().args vs. obj.mock.calls[obj.mock.calls.length-1].args

Using `jest-codemods` will replace the `mostRecent()` function with `calls[obj.mock.calls.length-1]`. Note that it won't touch the `.args` part, and this will lead to test failure since now, the converted code returns the args itself.

Hence make sure to remove this part. The final code will look like this:

```
// Before Migration
let args = obj.mock.calls.mostRecent().args;
// After Migration
let args = obj.mock.calls.[obj.mock.calls.length-1].args;
// Expected
let args = obj.mock.calls.[obj.mock.calls.length-1];
```

### jest.spyOn.calls.all() vs. jest.spyOn.mock.calls

The migration process removes the use of `.args` , but that is not sufficient as jest still doesn't support `calls.all()`, hence they need to be converted to mock.calls as it returns the same 2D array.

```
let demoSpy = jest.spyOn(...);

// Before Migration
expect(demoSpy.calls.all()[0].args[0]).toEqual(...)
// After Migration
expect(demoSpy.calls.all()[0][0]).toEqual(...)
// Expected
expect(demoSpy.mock.calls[0][0]).toEqual(...)
```

### .and.callFake(...) vs. .mockImplementation(...)

It has been observed that the migration tool does not convert the `.and.callFake(...)` function when it is called on a variable that holds `jest.spyOn(...)`. Hence manual conversion is required in this case:

```
let demoSpy = jest.spyOn(...);

// Before Migration
demoSpy.and.callFake(() => {...});
// After Migration
```

```
demoSpy.and.callFake(() => {...});
// Expected
demoSpy.mockImplenetation(() => {...});
```

### mockFn.calls.reset() vs mockFn.mockReset()

The migration does not take care of the `reset()` function. In Jest, you have to use `mockReset()`.

```
let demoSpy = jest.spyOn(...);

// Before Migration
demoSpy.calls.reset();
// After Migration
demoSpy.calls.reset();
// Expected
demoSpy.mockReset();
```

### Error: process.exit called with "1"

`jest-codemods` does not handle recursive conversions like the following:

```
// Before Migration
obj.and.callThrough().and.callFake(...);
// After Migration
obj.and.callFake(...);
```

Note that `.and.callFake(...)` is also invalid in Jest. Hence this has to be converted to `mockImplementation(...)` (refer to the previous case).

## Upgrading to version 6.1

### Jest v7 upgrade

In version 6.1 of Siemens Web Framework, Jest was upgraded to version 7. This may have some impact on your unit tests. For more information see **Jest Release Notes**.

### Changes to the aw-list contract

The `context-menu-command-context` property has been renamed to `command-context`. Its usage is unaltered.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Upgrading to version 7.x

This section explains how to upgrade to version 7.x of Siemens Web Framework.

**Warning**

It is assumed that your project has already been upgraded to the latest minor or patch release of version 6 of Siemens Web Framework. If it has not, complete previous upgrade steps before continuing the upgrade.

The following sections explain more in detail the steps to perform to upgrade to each specific 7.x release. Follow them in sequence depending on your upgrade scenario. If a version is not listed, no upgrade steps are necessary for that particular version.

## Updating your package.json file

Some changes introduced in version 7.0 of Siemens Web Framework require updates to the `packages.json` file of your project.

### Adding @swf/tooling as a development dependency

In previous versions, the `@swf/tooling` package was automatically installed as a dependency of the `@swf/core` packages. Because this package is essentially used at development time, it is now necessary to include it as a development dependency in your `package.json` file:

```
{
    "dependencies": {
        "@swf/core": "^7.0.0"
    },
    "devDependencies": {
        "@swf/tooling": "^7.0.0"
    }
}
```

This change must be made before installing Siemens Web Framework version 7.0 or higher.

**Tip**

If you have the **Siemens Web Framework CLI** installed on your system, you can run `aw upgrade 7.0.0` to perform this change automatically.

### Updating the package.json scripts

As of version 7.0, Some internal build dependency have been reviewed. As a result, the default scripts used to build, start, and test a Siemens Web Framework project must be updated.


Unpublished work. © 2025 Siemens

Verify that the scripts in your project do not call `react-app-rewire` directly and refer to the **default NPM scripts** for the most up-to-date script definitions.

> **Tip**
>
> If you have the **Siemens Web Framework CLI** installed on your system, you can run `aw start`, `aw build`, and `aw test` instead of the corresponding NPM scripts to avoid any problem: the CLI will automatically execute the correct script based on the version of Siemens Web Framework that you are using in your project.

## Migrating from aw-sidenav to aw-dialog

As of Siemens Web Framework v7.0, the **aw-sidenav** component is deprecated in favor of **aw-dialog**. This page provides information on how to migrate from `aw-sidenav` to `aw-dialog` based on specific usage patterns.

### Rationale

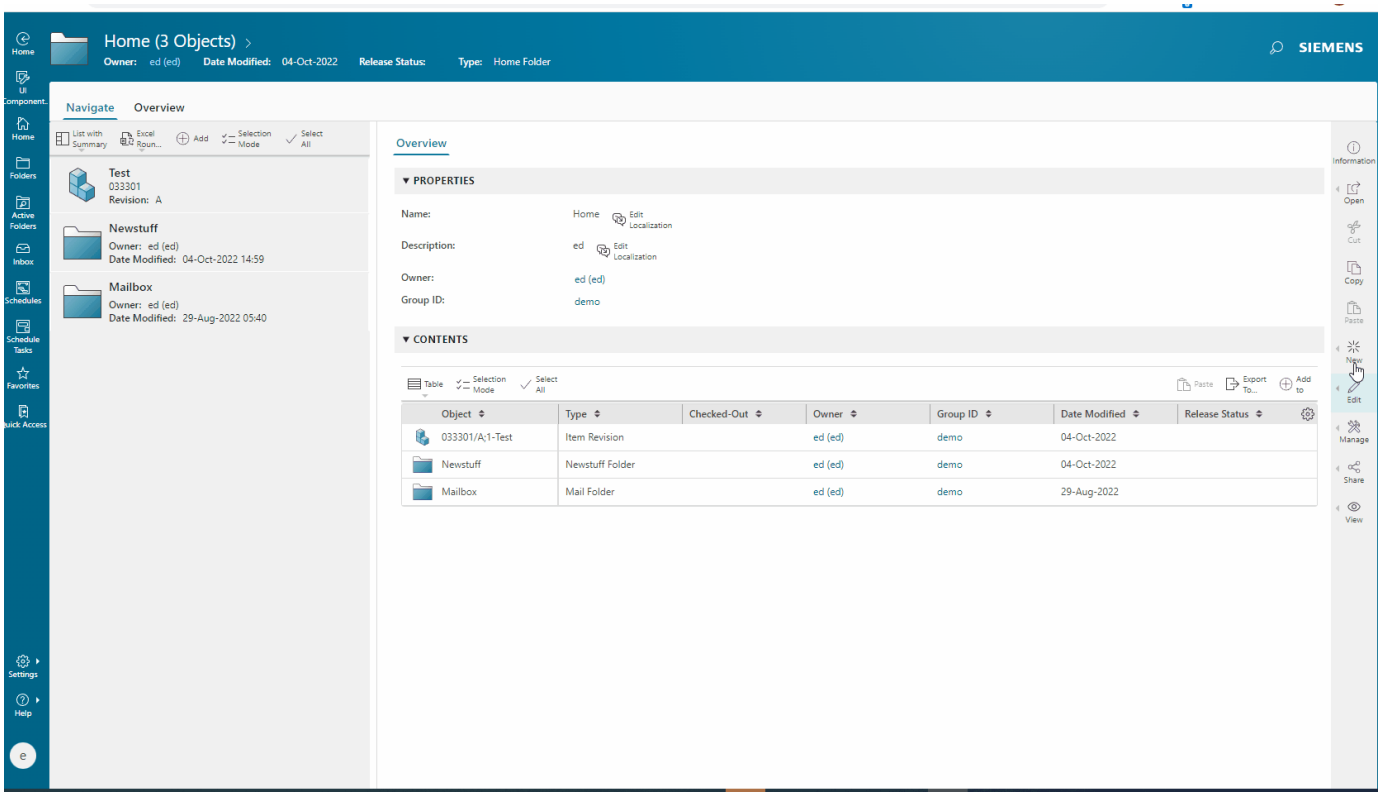
The old `AwSideNav` has some inherent shortcomings. `AwSideNav` is embedded inside a sublocation. This embedded nature makes sharing data between commands and the sidenav difficult. The global CTX object is used when sharing data between commands and the sidenav. Using the global CTX object is not recommended. Even when not used, the `AwSideNav` component exists in the DOM. Moving commands using the sidenav is also impossible as they are tightly coupled and operate through publishing events.

`AwDialog` replaces the `AwSideNav` component to overcome the shortcomings above while providing additional features. `AwDialog` provides the ability to move, resize, and dock/undock panels inside a page, including:

- New `AwDialog` component
- Dock and undock a push dialog and make it moveable
- Dock and undock applies to push-type dialogs only
- Once undocked, the user can move around and resize the dialog
- The user can dock the dialog back to its original position and size
- Move and resize the dialog by default
- A user, by default, can move around and position floating dialogs
- A user can reset the dialog to its original position and size
- Remembering the last position and size of a dialog
- Both push and float dialogs can remember their last position and size
- If there are ten dialogs in the application, each dialog can be uniquely sized and positioned
- The system will remember unique location and size per dialog


Siemens Web Framework
Unpublished work. © 2025 Siemens

This is an example of the dock behavior of a dialog, along with remembering the last position and size of a dialog.




Different usages of dialog can be found in **Storybook** by searching for `aw-dialog`.

## Design

`AwDialog` is a wrapper around `AwPopup`, and its configuration involves a **DialogOptons** object that is similar to the **PopupOptions** object.

The configuration takes 'parent' as an option. The parent acts as a CSS locator for the element, and the dialog will exist as a child of that CSS locator in the DOM. The dialog width and height are calculated with respect to the 'parent' element's dimensions. We are supporting two types of dialogs:

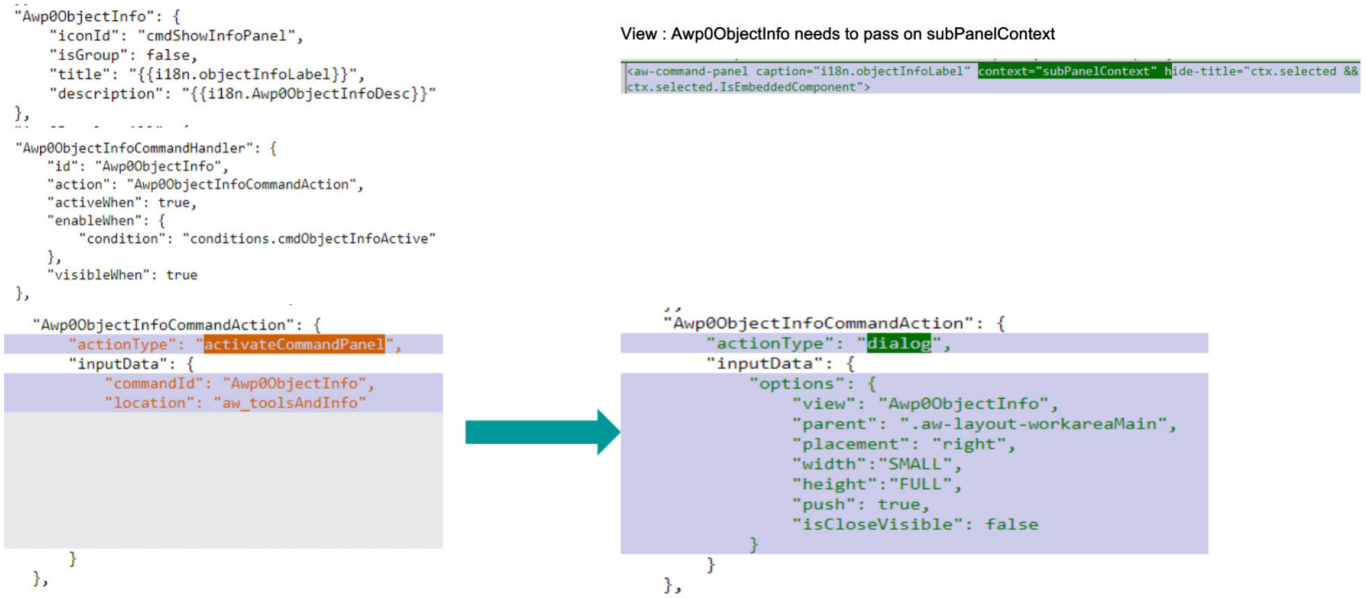
- **Float**
  - It resides as a child of an element specified by the parent locator.
  - It will not push the content of sibling
  - Its height and width are calculated with respect to the parent
  - Parent should have the position as relative for this type of dialog to work
  - This will be draggable by default
  - User can reposition or resize and remember it per dialog

- **Push**
  - It resides as a child of an element specified by parent locator.
  - It will push the content of sibling
  - parent should be flex container for this type of dialog to work

## Common patterns for replacing AwSidenav usage with AwDialog

### Pattern 1: Declarative Direct Activation

Convert usages of actions where `actionType: 'activateCommandPanel'` or `method: 'activateCommandPanel'` to `actionType: 'dialog'` with relevant options. Make sure the commandContext is passed to the underlying view.



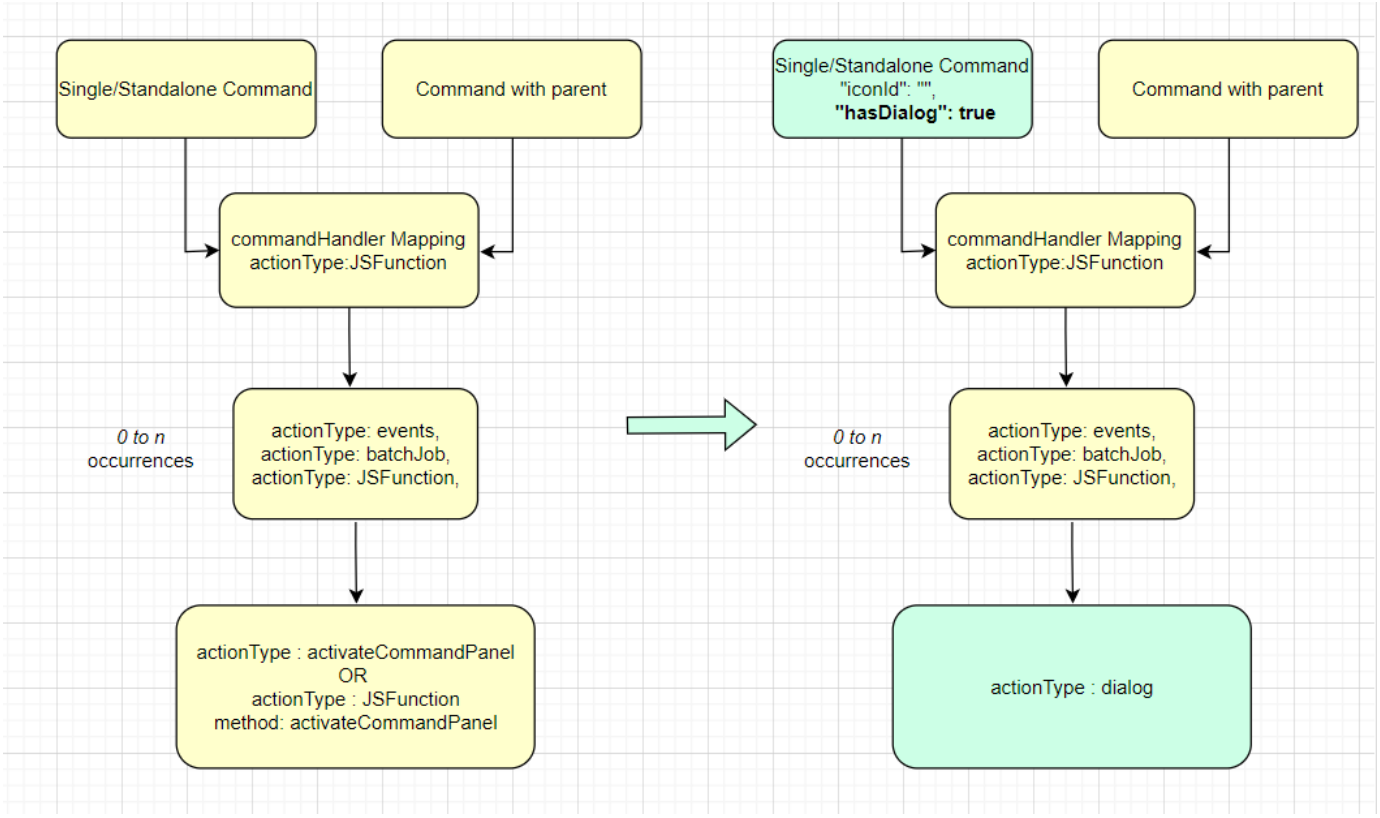

**Tip**

If you have the **Siemens Web Framework CLI** installed on your system, you can run `aw upgrade 7.0.0` to perform this change automatically.

### Pattern 2: Declarative Indirect Activation: Standalone Command

This pattern applies when no direct `activateCommandPanel` action is mapped to your commandHandler action. Instead, there are 'n' number of JSFunctions, events, and batch jobs involved before eventually having `actionType: activateCommandPanel`. Suppose there is any dialog associated during the chain of actions. In that case, the `hasDialog` has to be set to `true` in the command definition to let the SWF identify it. Notice how the batch is mapped to the commandHandler in the below snapshot, which also has an activateCommandPanel action.


Unpublished work. © 2025 Siemens




```
"Awp0NameValueCreate": {
    "iconId": "cmdAdd",
    "title": "{{i18n.ADD_BUTTON_TITLE}}"
},

"Awp0NameValueCreateCommandHandler": {
    "id": "Awp0NameValueCreate",
    "action": "createNameValuePropertyBatch",
    "activeWhen": {
        "condition": "conditions.cmdNameValuePropertyCreateActive"
    },
    "visibleWhen": true
},

"createNameValueProperty": {
    "actionType": "activateCommandPanel",
    "inputData": {
        "commandId": "Awp0NameValueCreate",
        "location": "aw_toolsAndInfo"
    }
},
```

View : Awp0ReplaceDataSet needs to pass on subPanelContext

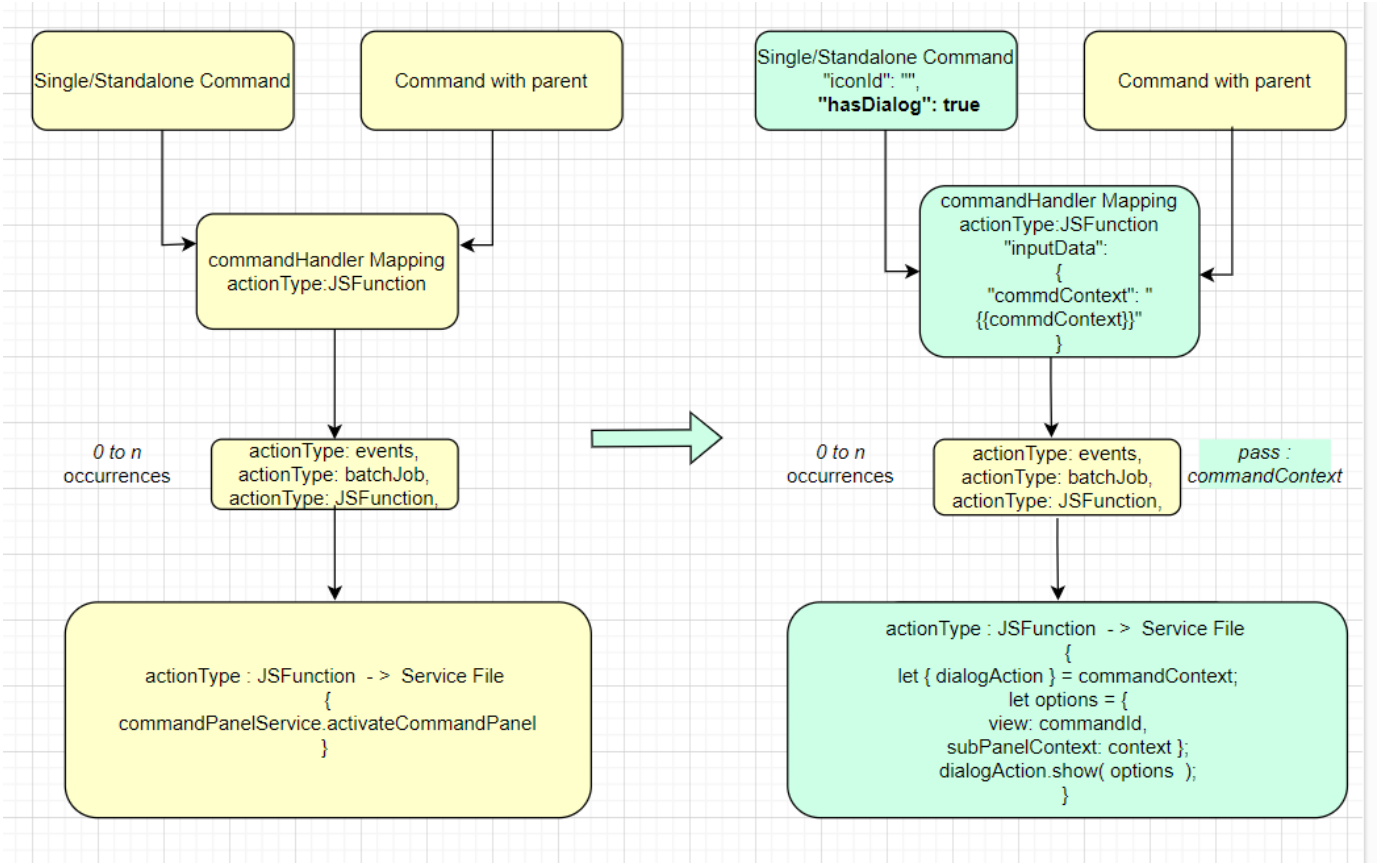
```
<aw-command-panel caption="i18n.NameValue_Create" context="subPanelContext">

"Awp0NameValueCreate": {
    "iconId": "cmdAdd",
    "title": "{{i18n.ADD_BUTTON_TITLE}}",
    "hasDialog": true
},

"createNameValueProperty": {
    "actionType": "dialog",
    "inputData": {
        "options": {
            "view": "Awp0NameValueCreate",
            "placement": "right",
            "parent": ".aw-layout-workarea",
            "width": "MEDIUM",
            "height": "FULL",
            "isCloseVisible": false,
            "subPanelContext": "{{commandContext}}"
        }
    }
},
```

## Pattern 3: Activation Via Service

There are instances when you may need to activate a command panel from a service file. The `commandContext` will have a `dialogAction` to show the dialog. There could be 'n' number of services involved provided the `commandContext` is correctly shared.



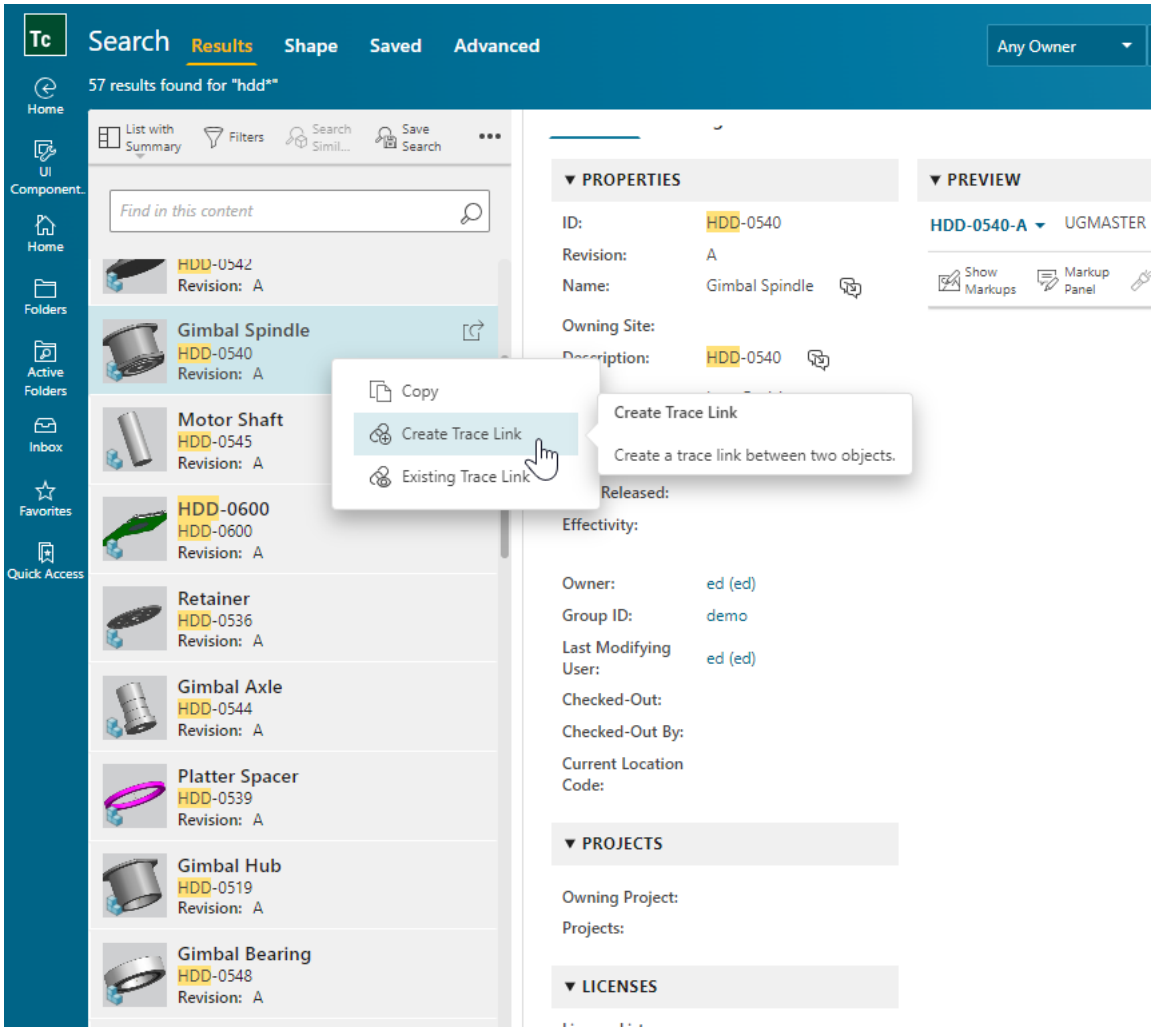

```
"Awp0ShowSaveAs": {
    "iconId": "cmdSaveAs",
    "title": "{{i18n.Awp0ShowSaveAsTitle}}",
    "description": "{{i18n.Awp0ShowSaveAsDesc}}"
},
"Awp0ShowRevise": {
    "iconId": "cmdRevise",
    "title": "{{i18n.Awp0ShowReviseTitle}}",
    "description": "{{i18n.Awp0ShowReviseDesc}}"
},
```

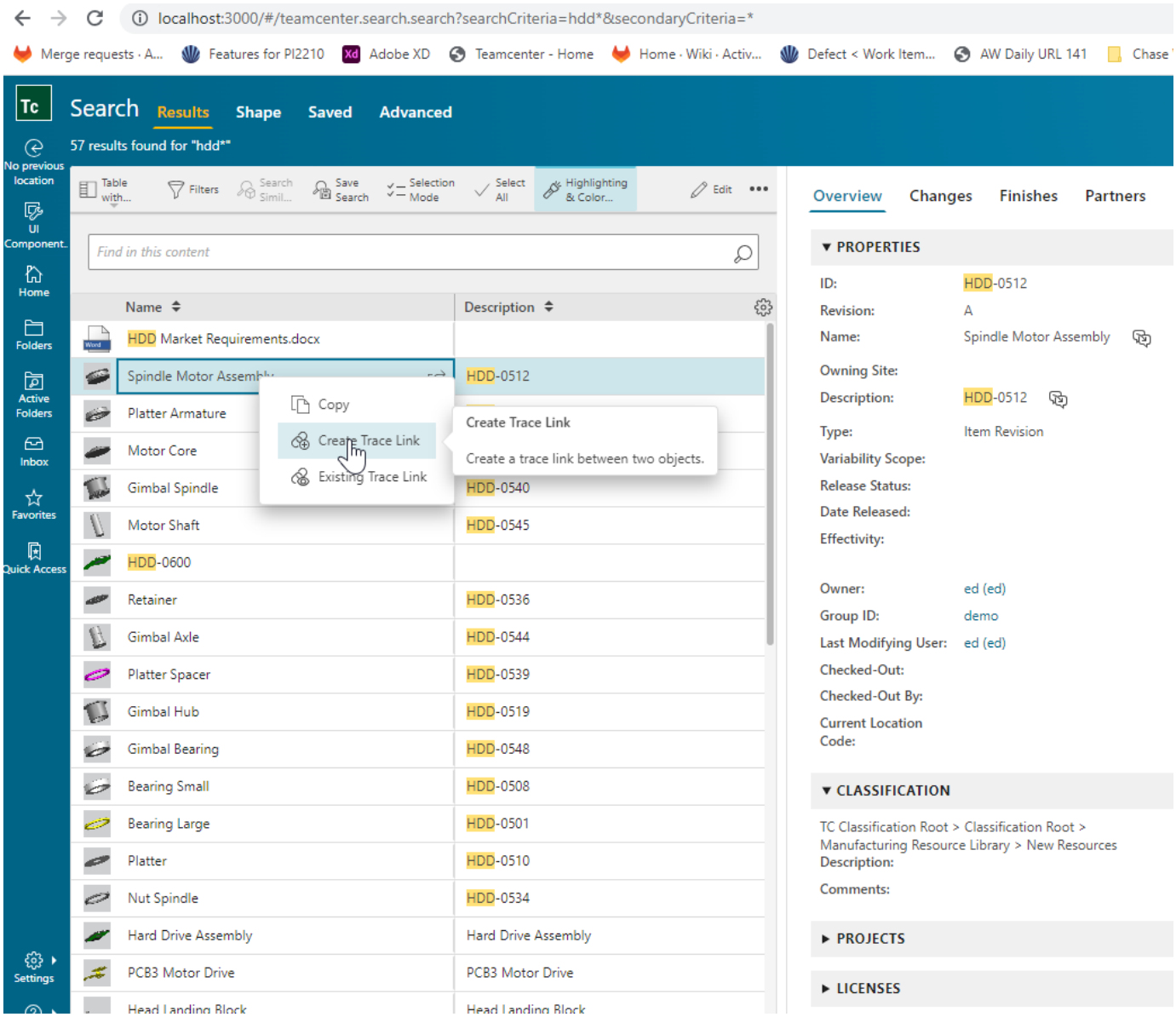
```
"Awp0ShowSaveAs": {
    "iconId": "cmdSaveAs",
    "title": "{{i18n.Awp0ShowSaveAsTitle}}",
    "description": "{{i18n.Awp0ShowSaveAsDesc}}",
    "hasDialog": true
},
"Awp0ShowRevise": {
    "iconId": "cmdRevise",
    "title": "{{i18n.Awp0ShowReviseTitle}}",
    "description": "{{i18n.Awp0ShowReviseDesc}}",
    "hasDialog": true
},
```

## Pattern 4: Custom Dialog component

Sometimes, a dialog needs to be opened from a right-click context menu command on a list, a table, or anywhere on the page. In this case, the command gets destroyed when it is clicked on. A custom dialog is required in this circumstance.


Unpublished work. © 2025 Siemens


Search
Results
Shape
Saved
Advanced
Any Owner
57 results found for "hdd*"
Home
UI Component..
Home
Folders
Active Folders
Inbox
Favorites
Quick Access
List with Summary
Filters
Search Simil...
Save Search
Find in this content
Revision: A
Gimbal Spindle
HDD-0540
Revision: A
Motor Shaft
HDD-0545
Revision: A
HDD-0600
HDD-0600
Revision: A
Retainer
HDD-0536
Revision: A
Gimbal Axle
HDD-0544
Revision: A
Platter Spacer
HDD-0539
Revision: A
Gimbal Hub
HDD-0519
Revision: A
Gimbal Bearing
HDD-0548
Revision: A
Copy
Create Trace Link
Existing Trace Link
Create Trace Link
Create a trace link between two objects.
PROPERTIES
ID: HDD-0540
Revision: A
Name: Gimbal Spindle
Owning Site:
HDD-0540
Released:
Effectivity:
Owner: ed (ed)
Group ID: demo
Last Modifying User: ed (ed)
Checked-Out:
Checked-Out By:
Current Location Code:
PROJECTS
Owning Project:
Projects:
LICENSES
PREVIEW
HDD-0540-A
UGMASTER
Show Markups
Markup Panel




There may be a use case where a custom dialog and its lifecycle are needed. An example would be to open a dialog onMount or no command involved. The `aw-dialog` page in **Storybook** has an example of this.

The `aw-dialog` must be consumed at the appropriate hierarchy in your view.html/jsx. Define the respective dialog action in the view model with basic options. Sometimes, passing this `dialogAction` to the respective component in the hierarchy is required. An example would be using the `subPanelContext` or context props.

```
<aw-full-screen include-component-name='AwUniversalViewer' header-
     title='{{subPanelContext.caption}}' header-
path='{{subPanelContext.activeTab.name ?
     subPanelContext.activeTab.name :
subPanelContext.showObjectContext.activeTab.name}}' header-
     context-
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
title='{{subPanelContext.selected.props.object_string.uiValues[0]}}'
     context='{...props.subPanelContext, markUpDialogAction:
actions.markupDialogAction}'>
</aw-full-screen>
<aw-dialog action="markupDialogAction"></aw-dialog>
```

```
{
  "schemaVersion": "1.0.0",
  "actions": {
    "markupDialogAction": {
      "actionType": "dialog",
      "inputData": {
        "options": {
          "parent": ".sw-showcase-container",
          "placement": "right",
          "width": "MEDIUM",
          "height": "FULL",
          "push": true
        }
      }
    }
  },
  "onEvent": [],
  "data": {},
  "functions": {},
  "messages": {},
  "conditions": {},
  "lifecycleHooks": {},
  "i18n": {},
  "props": {
    "sub-panel-context": {
      "type": "object"
    }
  }
}
```

```
export const awUniversalViewerRenderFn =
function( { viewModel, subPanelContext, fields,    elementRefList,
enableMultipleFilesPerDataset } ) {
    // Build the viewer in pieces to account for exist-whens
    // Get the uv ref
    let uvRef = elementRefList.get( 'uvRef' );
    const viewerData = {
        ...viewModel.viewerData,
        markUpDialogAction: subPanelContext.markUpDialogAction
    };
}
```

The Universal Viewer ensures the parent's dialogAction is available to its children(html, pdf, image). Which in turn passes to AwViewerHeader and all the commands within. Configuration of the


Unpublished work. © 2025 Siemens

commandsViewModel may be needed, and access the custom dialogAction within your JS service. Make sure to invoke the show API with prepared options.

```
export let activateMarkupPanel = function( viewerCtx ) {
    let markupCtx = exports.getMarkupContext();
    const { markUpDialogAction } = viewerCtx;
    .....
    //commandPanelSvc.activateCommandPanel('Awp0MarkupMain',
'aw_toolsAndInfo', viewerCtx, true );
    //prepare your options
    // this will override the default options defined in your ViewModel
[ Awp0ViewerGalleryViewModel.json ]
    let markUpOptions = {
        view: 'Awp0MarkupMain',
        parent: '.aw-layout-workareaMain',
        placement: 'right',
        width: 'MEDIUM',
        height: 'FULL',
        push: true,
        subPanelContext: viewerCtx
    };
    markUpDialogAction.show( markUpOptions );
};
```

## closeDialog API

The closeDialog API accepts a unique popup ID available in your subpanelContext via popupOptions. Passing the commandContext to the view is essential if it handles the closure. The view controls the dialog's closure with the API's help.

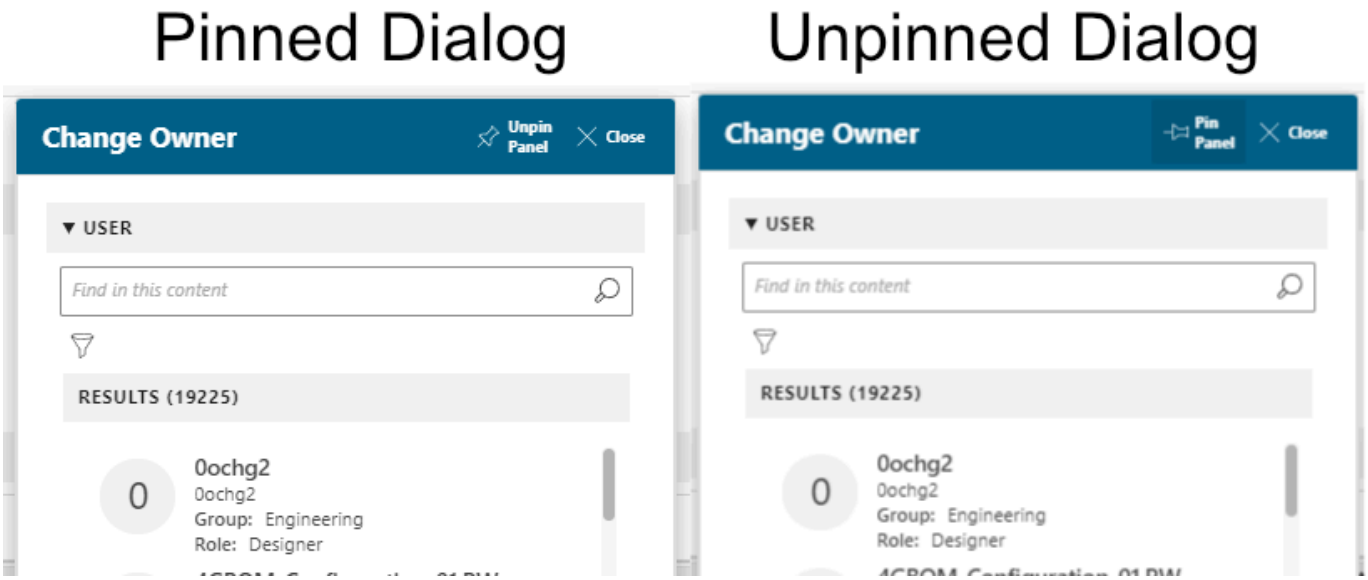
Siemens Web Framework
Unpublished work. © 2025 Siemens

The dialog is devoid of traditional events that worked with the sidenav. Any such dependencies can be handled independently by invoking the closeDialog API or overriding the 'Close' command handler conditionally without impacting OOTB behavior. During the conversion, ensure there are no side nav-specific events left over. An example of this would be having the "complete" event and "closeDialog" together doesn't make sense. As the closeDialog API is adapted, the "complete" event specific to sidenav should be removed.

## Pin/Unpin Dialog

Like the sidenav, a dialog can be easily pinned/unpinned by providing options highlighted in the code snippet below. isPinUnpinEnabled is the option supporting pinning/unpinning. The panelPinned option sets the initial state to pinned.

```
{
    "openChangeOwnerPanel": {
        "actionType": "dialog",
        "inputData": {
            "options": {
                "view": "Awp0ChangeOwner",
                "placement": "right",
                "parent": ".aw-layout-workarea",
                "width": "MEDIUM",
                "height": "FULL",
                "isCloseVisible": false,
                "subPanelContext": "{{commandContext}}",
                "isPinUnpinEnabled": true,
                "panelPinned": true
            }
        }
    }
}
```




## FAQ

### How do I access the panelContext?

When the sidenav opened, the "context" object was passed to activateCommandPanel, which was registered on the global CTX object that could be used in a JsFunction later. The context is not available in the commandContext, so data that resided on 'panelContext' for sidenav is now available on commandContext. Make sure the subPanelContext passed to your view is also made available at the right place in the component hierarchy. One of the purposes of passing subPanelContext to the view was to make available information to the command panel. However, the same needs to be passed to child components as needed via context or sub-panel-context.

```
<aw-command-panel caption="i18n.diagramsPanelTitle"
context="subPanelContext">
    <aw-command-sub-panel sub-panel-context="subPanelContext" >
    ...
</aw-command-panel>
```

### How do I debug what is passed in the commandContext?

The dialog is embedded inside `AwCommand`, so in the ReactJS debugger tool, locate the command and see what is passed to it. If needed, go up in the hierarchy to see who provides context to `AwCommand`.

### Where is the dialog component?

The dialog needs a placeholder component or command. `AwCommand` automatically hosts a dialog if it has a popup or command with an action type of dialog.


Unpublished work. © 2025 Siemens

### Why don't I see my dialog, even though everything is wired correctly?

The dialog lifecycle is bound to the container's lifecycle. An example would be if the command is destroyed, dialog within is also destroyed.

### What if no direct command is involved with my activateCommandPanel?

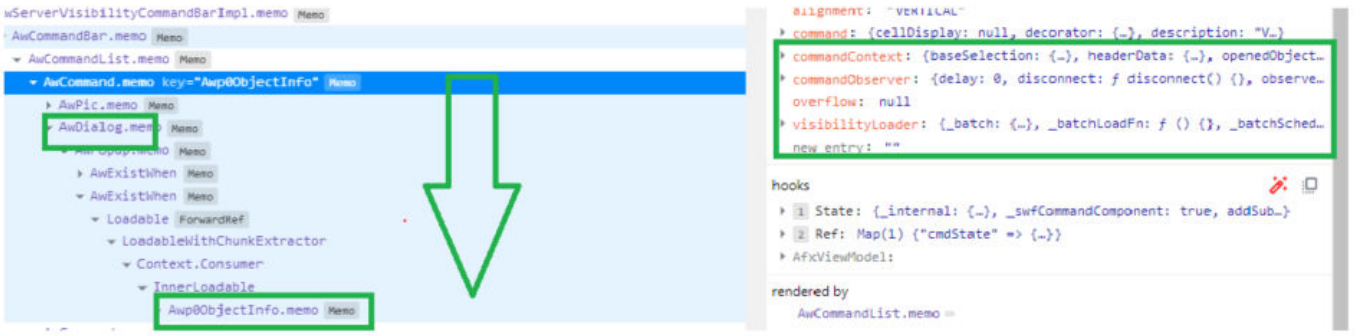
If no command exists, you can use AwDialog inside their component as needed.

### When to use `hasDialog`?

`hasDialog` becomes important when a command handler has no direct action mapped with an actionType dialog but will eventually have one via a set of batch/events/jsFunctions.

### My close command is not working, or I don't see my commands on the dialog header. What could be wrong?

Make sure to share command context via options to the dialog. Also, ensure your underlying view is rightfully passed on to display your custom commands. Here is an example:




## Changes to user session controls

As of version 7.0 of Siemens Web Framework, the user avatar and `aw_userSessionbar`` command bar (collectively referred to as *user session controls*) have been moved from the left-hand navigation toolbar to to the top-right corner of the page, next to the Siemens logo.

If you are using one of the out-of-the-box **location** components like `common-location` or `aw-default-location` or one of the out-of-the-box header components like `common-header` or `aw-header2` that are provided by Siemens Web Framework, you do not have to make any changes to your code, and the user session controls will be automatically moved to the header, next to the Siemens logo.

If you are using custom location that does not use one of the out-of-the-box header components or a custom view, you must include the **aw-session-controls** component in your view, like this:

```
<aw-session-controls></aw-session-controls>
```

## Change in right wall command placement

The `aw_rightWall` command bar anchor has been removed from all default view templates. If you were leveraging this anchor for your commands, you should review your command placement based on UX guidelines. A new anchor called `aw_workarea_toolbar`has been added to add commands to the new workarea toolbar (page-level toolbar) which should be used instead of the right wall in most cases.

## Upgrading to version 7.1

The following pages list all the steps to perform to upgrade to Siemens Web Framework v7.1:

- **Migrating from aw-default-sublocation to aw-standard-sublocation**
- **Migrating build.json srcPaths configuration**
- **Migrating JSON configuration files**
- **Highcharts library upgraded to version 11.3.0**

**Tip**

Except for some edge cases related to the **aw-default-sublocation migration**, the whole upgrade can be automated by running the following command using the **SWF CLI**:

```
aw upgrade 7.1.0
```

### Migrating from aw-default-sublocation to aw-standard-sublocation

As of Siemens Web Framework v7.1, the **aw-default-sublocation** component is deprecated in favor of **aw-standard-sublocation**. This page provides information on how to migrate from `aw-default-sublocation` to `aw-standard-sublocation` based on specific usage patterns.

**Note**

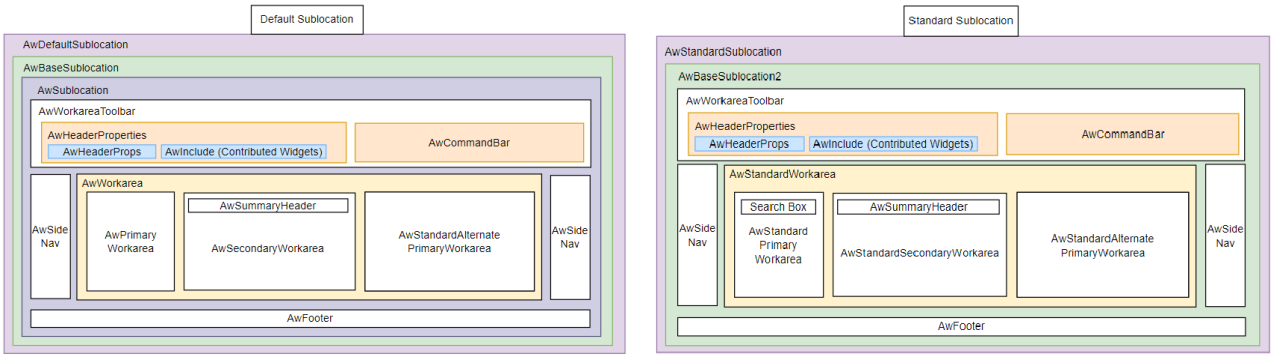
Note that this component is just being deprecated in this release, but it will continue to work at least until the next major release. However, we recommend that you start migrating now as described in the following sections.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Rationale

Framework is currently providing two types of **high-level-sublocations**, and because of legacy reasons both had to be maintained.

At present, AwStandardSublocation offers more functionalities compared to AwDefaultSublcoation, and both share the same component hierarchy. Therefore, AwStandardSublocation can seamlessly replace AwDefaultSublcoation within the existing system.




### Migration using CLI

AwDefaultSublocation consumers can migrate automatically with the help of the SWF **CLI** by running the following command:

```
aw migrate default2standardsublocation
```

**Tip**

This command will automatically be run if you run the CLI to upgrade to version 7.1.0 of Siemens Web Framework (`aw upgrade 7.1.0`).

**Note**

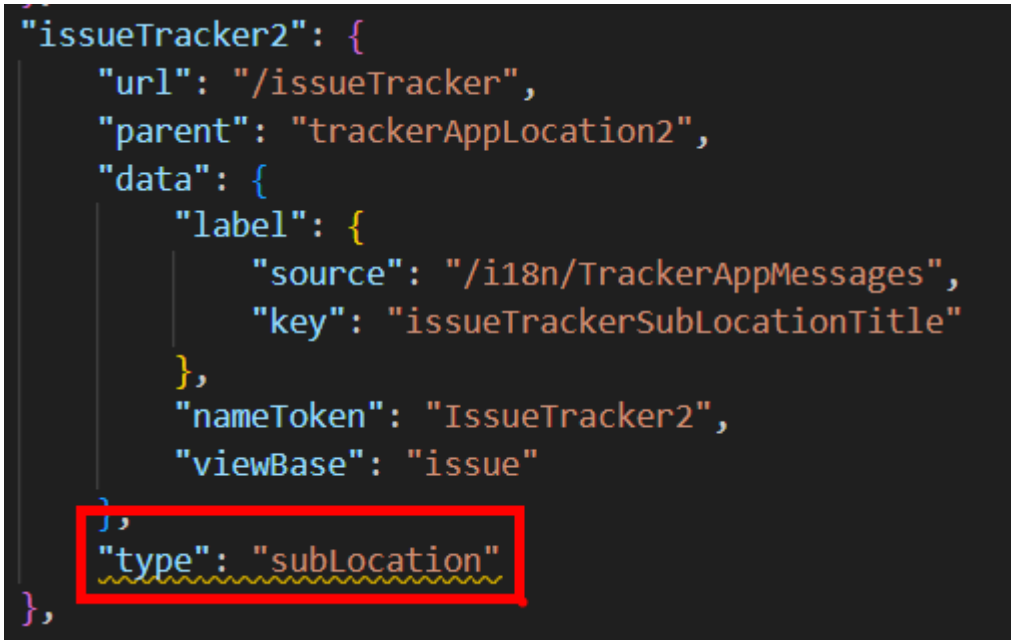
JSX usages will not be detected or migrated automatically.

### Common patterns for replacing AwDefaultSublocation usage with AwStandardSublocation

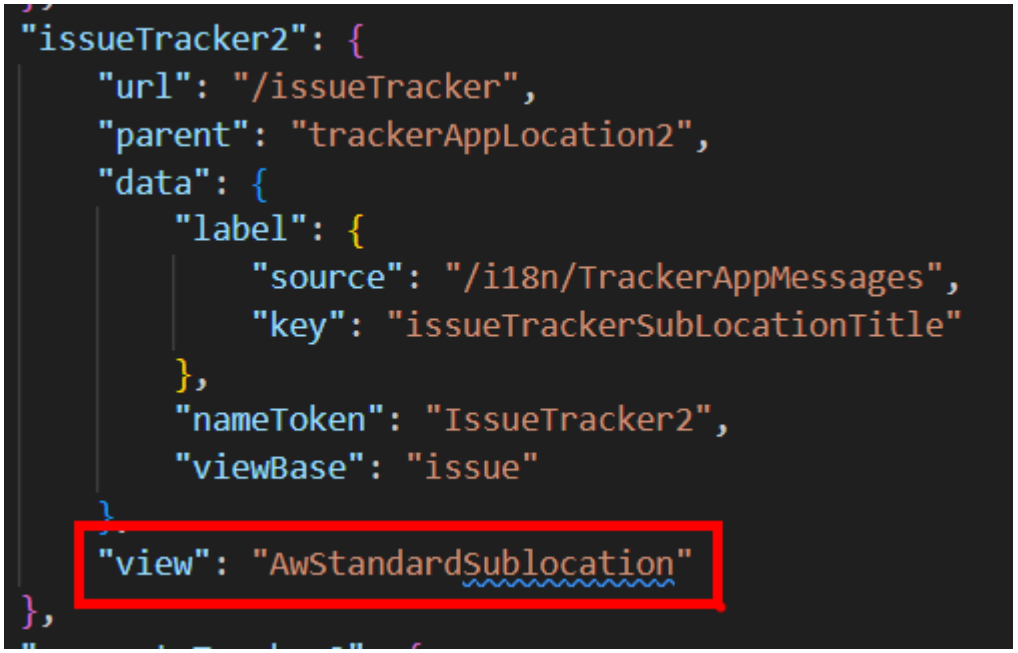
#### Pattern 1: Declarative usage through states.json

- Replace `"type": "sublocation"` with `"view": "AwStandardSublocation"`

- e.g. consider the following state



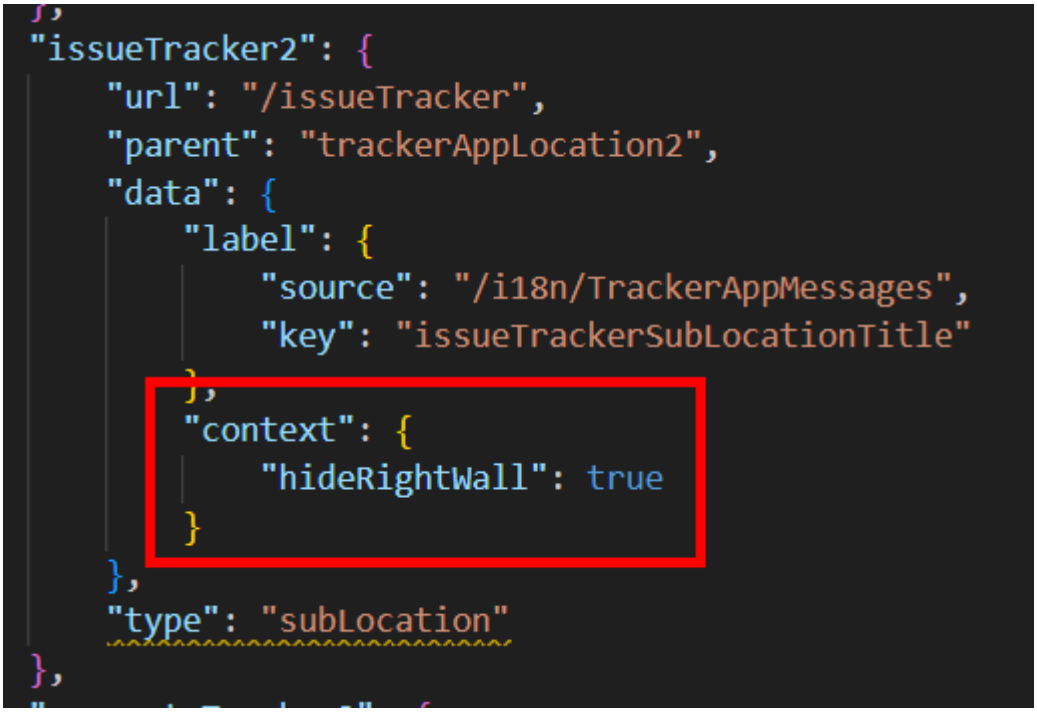

can be updated like following



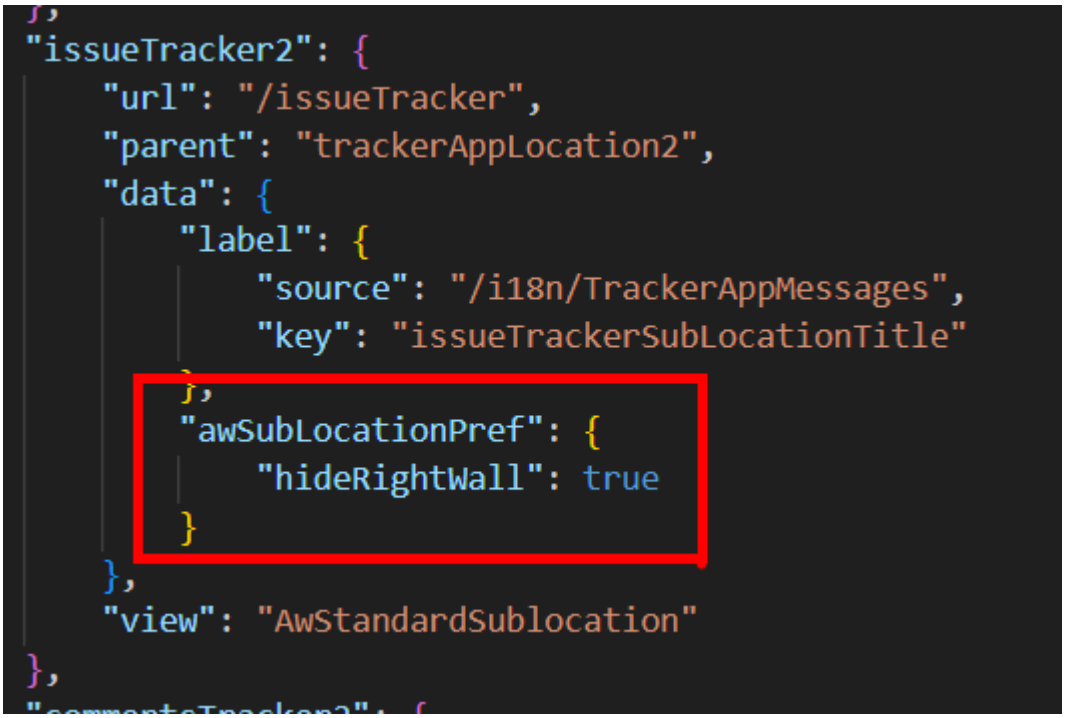

- If `hideRightWall` is provided on `context` then it needs to be moved to `awSubLocationPref`


Unpublished work. © 2025 Siemens

- e.g.




This instance can be converted as




**Pattern 2: HTML usages**

- Replace `<aw-default-sublocation>` with `<aw-standard-sublocation>`
- Pass provider to `<aw-standard-sublcoation>`

- This provider is usually available on `data` section of view-model as `data._sublocation`
- e.g.

```
<aw-default-sublocation sub-panel-context="data.provider"></aw-default-sublocation>
```

This instance can be converted as

```
<aw-standard-sublocation provider="data.provider" sub-panel-context="data.provider" ></aw-standard-sublocation>
```

- If class is provided on `<aw-default-sublocation>` need to move it up in the hierarchy.
  - e.g.

```
<aw-default-sublocation class="test-class" sub-panel-context="data.provider" ></aw-default-sublocation>
```

This instance can be converted as

```
<aw-flex-row class='test-class'>
    <aw-standard-sublocation provider="data.provider" sub-panel-context="data.provider"></aw-standard-sublocation>
</aw-flex-row>
```

**Pattern 3: JSX usage**

- Similar to html migration, replace `AwDefaultSublocation` with `AwStandardSublocation`.
- Pass provider to `AwStandardSublocation`

## Upgrading build.json srcPaths configuration

As of Siemens Web Framework v7.1, the @swf/core package now internally depends on @swf/table, @swf/elements, @swf/essentials and @swf/engine. All these packages contain SWF kits and modules. If your application is migrated to v7.1 and `build.json` srcPaths configuration does not contain the paths to these packages, then the build will break. Make sure the `build.json` srcPaths property contains following entries along with applications-specific entries.

```
{
    "srcPaths":[
        "node_modules/@swf/core/@(repo|src|test)",
        "node_modules/@swf/table/@(repo|src|test)",
        "node_modules/@swf/elements/@(repo|src|test)",
        "node_modules/@swf/essentials/@(repo|src|test)",
        "node_modules/@swf/engine/@(repo|src|test)",
        "src"
    ]
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Note**

You can run `aw migrate updatebuildjsonconfig` to migrate build.json srcPaths using the SWF CLI. This script will check the SWF packages that application is using in `node_modules` and add entries appropriately. This script is also executed automatically when running `aw upgrade 7.1.0`. The following packages will automatically update build.json srcPaths when updated via the `npm install`, `npm ci` or `npm update` commands.

- @swf/engine
- @swf/table
- @swf/essentials
- @swf/elements

## Migrating JSON configuration files

If you have created any of the following files before version 7.1 of Siemens Web Framework (Teamcenter 2406), you need to migrate them to the new format introduced from this version onwards:

- secondaryWorkAreaTabs.json
- indicators.json
- typeIconsRegistry.json

To perform the migration, you can run the following command using the SWF CLI:

```
aw migrate configurations
```

**Tip**

This command will automatically be run if you run the CLI to upgrade to version 7.1.0 of Siemens Web Framework (`aw upgrade 7.1.0`).

**Note**

The migration of configurations is also part of the build process. This means that if you build an application with configurations written in the old format, it will be converted to the new format at runtime to create the stitched configuration. To migrate the configurations to the new format at the source, you need to explicitly use the CLI.


Unpublished work. © 2025 Siemens

## Highcharts library upgraded to version 11.3.0

In Siemens Web Framework v7.1, the **aw-chart** component has been updated to use Highcharts 11.3.0 (was 9.0.1).

There a few internal differences with the newer Highcharts library that can affect `aw-chart` users:

- Issues with charts with a very large number of data points (changes below are demonstrated with the "Scatter Chart with million points" example in Storybook)
  - Previously, data labels were automatically disabled above a certain threshold to avoid overlapping. In some cases it may now be desirable to set isDataLabelOnChartEnabled to false to avoid useless labels.
  - In latest Highcharts examples with many thousands of data points, axis min and max are set to improve performance. We follow the same pattern with our million point example now to preserve performance.
- Issues where charts are now less tolerant of bad practices.
  - Resize/render timing altered. Cases with improper layout (e.g., h-12 with sibling) might be more problematic now.
  - Brackets/illegal characters in hc variable names no longer work.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# 10. Best Practices

The following pages contain some best practices to follow in any Siemens Web Framework project:

- **Implementation best practices**
- **Troubleshooting best practices**
- **Security best practices**
- **Performance best practices**

## Implementation best practices

The following guidelines will help you get the full potential out of Siemens Web Framework.

### Elements

- Reuse **elements** as building blocks for your appliation.
- Elements should be responsible for one clear purpose.
- Elements naming must be semantically meaningful and consistent in usage.
- Elements should be codeful internally but always used declaratively (codeless).
- If an element is being contibuted, the right abstraction is necessary.
- Elements should not leak internal implementation.
- Elements should not leak CSS classes.
- Elements should not expose internal choice of technology

### View

- Keep **views** clean
- **Views** are for specifying **element** hierarchy only
- Each view should refer to **data**, **actions**, **localizations** and **conditions** from the corresponding **view model**
- The *visible-when*, *enable-when*, *exist-when* should refer to **conditions** in the **view model**
- Use **actions** and **events** to declaratively map **view model** state
- Do not leak internal implementation (**elements**, **views** and **view models**)
- You should avoid CSS classes in the **views**
- Do not leak internal choice of technology in **views** and **view models**)


Unpublished work. © 2025 Siemens

### Services

- Create fit-for-purpose services
- Tailor services to minimize network trips and reduce latency costs
- Push logic to the server
- Minimize client side code and processing - Less code to maintain, greater reliability, reduced complexity, improved performance and reduced footprint
- **Actions** defined in declarative **view models** should be able to directly and simply map input and output JSON data for the service; there should not be lots of client-side transformation
- Have at most one service call per user action
- Have at most one service call when the **view model** is loaded

### JavaScript Code Extensions

- JavaScript code extensions and usage of *glue code* should be kept to a minimum; it should be an exception, not the rule

## Troubleshooting best practices

The following sections provide some basic troubleshooting tips for common problems that may arise while developing your web application.

### Siemens Web Framework tooling is not working as expected

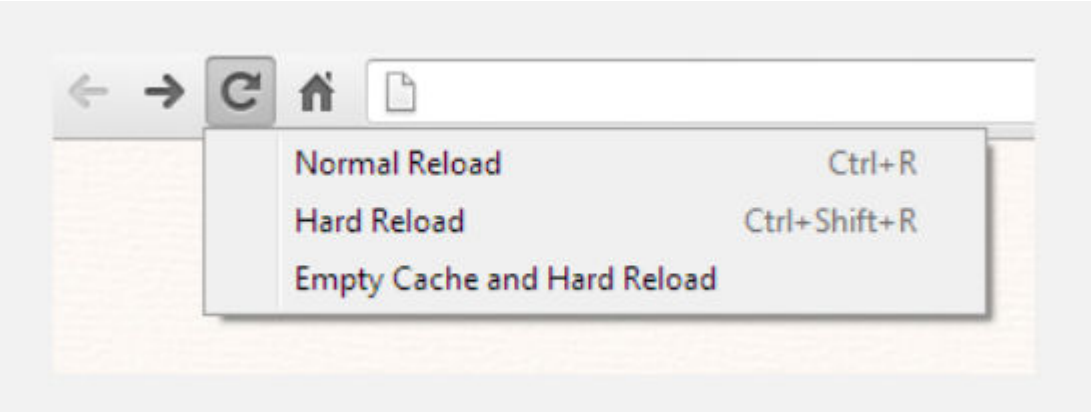
- Check that your version of NodeJS is supported by Siemens Web Framework tooling. This should be specified in the `engines` section of the `packages.json` file of the `node_modules/@swf/tooling` package within your project.
- Make sure you are running the latest version of the framework by checking the versions specified in your project `package.json` file. To force a re-installation of the @swf packages:
  1. Delete your `packages-lock.json` file
  2. Run `npm install`

### Changes are not reflected in the UI

- Clear the browser cache and perform a hard-reload in your web browser. For example, in Google Chrome:
  1. Press `F12` to open the Chrome Developer Tools.


Siemens Web Framework
Unpublished work. © 2025 Siemens

2. Right-click the browser refresh button and select **Empty Cache and Hard Reload**.




- Restart your development build:
  1. Verify that all the **default NPM scripts** are available in your project.
  2. If your web application is running via `aw start`, stop it by pressing `CTRL+C`.
  3. Run `aw clean`.
  4. Run `aw start` again.

## Unable to run aw or swf CLI commands

In case you face an error along the lines of `'aw' is not recognized as an internal or external command`, it is since SWF CLI is not properly installed for system.

## Deprecation warning of Punycode in console log

The console logs of any command of SWF may result in the following warning:

```
D:\work\swf\app\sampleApp>npm run start
npm warn config ignoring workspace config at D:\work\swf\app\sampleApp/.npmrc

> @swf/sampleApp@7.2.0-0 start
> node node_modules/@swf/tooling/js/wrapper.js start

[16:26:51] info: + config-overrides: generated paths (102 ms)
[16:26:51] error: + (node:40352) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland al
ternative instead.
[16:26:51] error: + (Use `node --trace-deprecation ...` to show where the warning was created)
[16:26:52] info: + PRE_BUILD_CHECKS: 25350 MB available / 67869 MB total
[16:26:52] info: + Default SOURCE_MAP_QUALITY is set to 'MEDIUM'. For faster build timing and HMR set the SOURCE_MAP_QUALITY
 to 'LOW'.
[16:26:52] info: + Building site: tutorial
[16:26:52] info: + Included kits:
```

This warning is caused by the Node.js 22 update, which some dependencies have not yet addressed. It does not impact the functionality of your application development.

If you still wish to supress this warning, kindly use the `--disable-warning` option in the script defined in the package.json.

# Security best practices

This page contains information on security best practices that you should be following when implementing a web application using Siemens Web Fraework, and what Siemens Web Framework provides and supports for mitigating attacks and address security vulnerabilities.

## Enabling Content Security Policy (CSP)

**Content Security Policy**, a W3C Working Draft, is a declarative policy that sets content restrictions for a web resource. It is implemented via the `Content-Security-Policy` HTTP header and can be used to instruct browsers to restrict loading of resources only from trusted sources which can be done by whitelisting specific origins, using nonces or hashes.

**Note**

Although it is possible to specify CSP directives using the `<meta>` tag on the client side, Siemens Web Framework does not rely on this method, as it does not enforce specific CSP policies. Instead, you should configure your web server to include the appropriate CSP header depending on your needs and use cases.

CSP can prevent cross-site scripting (XSS) vulnerabilities, click-jacking, mixed content security issues and code injection. It can provide a second layer of defense an protect the user's application. Using a strict CSP policy, combined with other security best-practices such as adopting template systems with strict contextual auto-escaping, vulnerability scanning, and a manual security review can significantly decrease the risk of introducing an XSS bug which can be exploited against users of modern browsers.

**Note**

It is important to remember that CSP alone does not guarantee protection against cross-site scripting. As mentioned in the **OWASP Cheat Sheet**: *"CSP should not be relied upon as the only defensive mechanism against XSS. You must still follow good development practices such as the ones described in **Cross-Site Scripting Prevention Cheat Sheet**, and then deploy CSP on top of that as a bonus security layer."*.

## Structure of the CSP header

A CSP header looks like the following:

```
Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; frame-ancestors 'self'; form-action 'self'
```

The value of the header is a semicolon-separated list of *directives* specified in the following way: ***directive*** **'*value*'**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

By default, CSP directives are wide open. If you don't set a specific value for a directive, (for example `font-src`) then that directive behaves by default as though you'd specified `\*` as the valid source (for example, you could load fonts from anywhere, without restrictions). You can override this default behavior by specifying a `default-src` directive. This directive defines the defaults for most directives that you leave unspecified. Generally, this applies to any directive that ends with `-src`.

## Recommended CSP policies

For a simple but comprehensive guide on CSP best practices and the recommended CSP policies, see the **OWASP Chear Sheet**.

Consider for example the following header, which is quite restrictive and recommended by OWASP:

```
Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; frame-ancestors 'self'; form-action 'self'
```

Basically, it sets the `default-src` directive to `'none'`, thereby blocking all external requests by default, and then:

- It restricts loading scripts (`script-src`), images (`img-src`), and stylesheets (`style-src`) only from the same origin as the application (`'self'`).
- It restricts connections and requests (`connext-src`) only to the same origin as the application (`'self'`).
- It specifies that the application can be loaded in an iFrame (`frame-ancestors`) only within the same origin as the application (`'self'`).
- It restricts forms (`form-action`) to be submitted only to the same origin as the application (`'self'`).

Note that in this case, a lot of `-src` directives (like `font-src` to load external fonts) are omitted and will default to `'none'`.

In general, the recommended approach is to test your application with a fairly restrictive CSP header like this one, and then progressively enable specific directives depending on the use case. For example, by whitelisting specific origins to which resources can be safely requested.

## CSP required by Siemens Web Framework

Some functionality provided by Siemens Web Framework require certain directives to be set to values which allow evaluation in certain contexts.

More specifically, the following directive configurations are required:

| Directive | Value | Notes |
|---|---|---|
| script-src | 'self' 'unsafe-inline' 'unsafe-eval' | External scripts are normally loaded from the same origin. In certain contexts, inline scripts and some forms of evaluation are used (see the dedicaded section further on). |
| font-src | 'self' | External fonts are normally loaded from the same origin. |
| connect-src | 'self' | Some HTTP requests to external assets on the same origin are performed. |
| img-src | 'self' data: | Images are normally loaded from the same origin, and in some case they may be included via data URIs. |
| style-src | 'self' 'unsafe-inline' | External styles are normally loaded from the same origin. In certain contexts, inline styles are used (see the dedicated section further on). |

The following is an example header with the `default-src` set to `'self'` and the specific directive values mentioned in the previous table:

```
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; img-src 'self' data:; style-src 'self' 'unsafe-inline'
```

## Why are `unsafe-inline` and `unsafe-eval` needed for some directives?

Siemens Web Framework has to evaluate some strings as JavaScript expressions, for example when processing View and ViewModel expressions and conditions. Because of this, it is necessary to specify `unsafe-eval` for the `script-src` directives.

Additionally, `unsafe-inline` must be specified for the `script-src` and the `style-src` directive if you are using any of the following third-party libraries:

- HighCharts (used by the `aw-chart` component)
- Monaco (used by the `aw-source-editor` component)
- CKEditor (used for rich-text editing)

**Tip**

If you are using additional third-party software, you may need to enable additional CSP directives if they rely on functionality that would be blocked by CSP directives.

## Additional information on CSP

- For more information on special directive sources, and allowing inline resources in conjunction with hashes and nonces, see the **Special Directive Sources** section of the OWASP Content Security Policy Cheat Sheet.


Siemens Web Framework
Unpublished work. © 2025 Siemens

- For some additional guidelines on implementing a strict Content Security Policy using nonces, see **Mitigate cross-site scripting (XSS) with a strict Content Security Policy (CSP)**
- For a quick reference on CSP directives and their support in different browsers, see the **Content Security Policy Reference**.
- For the latest version of the CSP W3C Working Draft, see **Content Security Policy Level 3**

## Preventing Cross-Site Scripting (XSS) attacks

Cross-Site Scripting (XSS) is a style of attack where the front of the website acts as a launching point for attacks on other users visiting the website. This happens when developers don't properly test their code for the possibility of allowing scripts to be injected.

**Tip**

For more information on Cross-Site Scripting see the **Cross-site Scripting Page** on the OWASP web site, and their **Cross-Site Scripting prevention cheat sheet**.

Siemens Web Framework provides some built-in ways to mitigate the risk of cross-site scripting.

## Input sanitization

By default, input sanitization is applied to every input in a view file. Sanitization of ViewModelProperties in a declarative view model is carried out as well.

If a custom component is created out of HTML tags(like `<input>`) and is directly invoked in the DOM using javascript code/glue code, there is a possibility that the user input is not sanitized. In this case, the attacker might inject JavaScript code which if rendered will cause a security threat.

In this case, you can sanitize the value that is going to be passed to the element using the `htmlEscapeAllowEntities` method of the `js/sanitizer` service. This method contains 3 parameters:

- `value` the HTML String that needs to be escaped.
- `replaceAnchors` Set this to `true` if any external reference should be wrapped in an HTML, for example the `href` attribute of an `<a>` tag.
- `replaceNewLines` Set this to `true` if any newline characters should be replaced by an HTML `<br>` tag.

## Preventing Cross-Site Request Forgery (CSRF) attacks

CSRF is a server attack that occurs on the functionality of web applications which use form-based submissions like POST requests and cookie-based authentication. In CSRF, an attacker tricks the user into visiting a different web page with malignant code that sends a malicious request to the application's web server. The attackers can change the integrity of data by changing user's email, password or profile image.


Unpublished work. © 2025 Siemens

Sometimes the attacker can access the sensitive data by switching user to attacker's account(if there is no validation).

CSRF attacks are typically mitigated via server-side code and web server configuration, regardless of what web framework is used on the client side.

**Tip**

For more information on Cross-Site Request Forgery see the **Cross-site Request Forgery Page** on the OWASP web site, and their **Cross-Site Request Forgery prevention cheat sheet**.

# Performance best practices

This guide documents the performance factors a web client Solution built on Siemens Web Framework should take into account to achieve optimal performance. Given your web client will be operating over a WAN you need to manage both latency and bandwidth. In a global deployment latency is a tax on every request made over the WAN in the order of ~300ms per request for cross-continental cases. For best user perceived performance we must consider how to minimize chats over the network to reduce latency costs. Techniques such as bundling help us to “get more together” and cache control, “don’t re-fetch what you have in the browser already”, help us to reduce the impact of these performance costs. To reduce bandwidth demands a combination of demand loading, instead of downloading all code and data at start-up, in addition to minification and other compression techniques help us to reduce the size of data transported over the WAN.

Some of the capabilities to optimize the performance of your web client Solution are provided with Siemens Web Framework. Other aspects will relate to how the components and infrastructure serving the content are configured which is beyond the domain of what is provided in Siemens Web Framework, but for which we can highlight some best practice to follow and implement.

## Build for Development vs Test and Production

You will have slightly different priorities when developing your web client as to when you are testing or deploying for production. During development iteration, legibility of the code for debugging and speed of iteration are important factors. When deploying for production, runtime efficiency over higher latency WAN becomes critically important such as optimizations for caching and minimizing latency and bandwidth costs.

By default Siemens Web Framework will build your Solution optimized for Development mode. This is controlled by the “DRAFT” environment variable. It is true by default which means the minification and bundling steps are skipped. Setting this to false will configure ApoSiemens Web Frameworkllo build tooling to optimize for Test and Production. Features such as bundling & minification will then be used.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Minification

One means of reducing bandwidth demands is to make artifacts smaller for transport by minifying. Minification is the process of removing unnecessary characters such as white space without materially changing the contents (such as variable names etc). The Siemens Web Framework build tooling when in Production mode (DRAFT is false), will minify the JS artifacts using the tool specified in the kit.json

Minification and concatenation of CSS and JSON files (such as localization) is also provided by the Siemens Web Framework build tooling when in Production mode (DRAFT is false).

## Cache control

When in Production mode the build system creates the assets for the site in an “assets” folder which has a build stamp suffix for its name (e.g. “assets71829313”). This ensures the site files deployed have unique URLs and are therefore considered as different files by the browser cache. As a result this allows the setting of the HTTP header “Cache-Control” to have a very large “max-age”. The browser will then cache these resources once it has retrieved them and will not re-fetch safe in the knowledge they will not change unless redeployed (at which point they are considered as different files given they will have different URLs).

Setting of the Cache-Control max-age in the HTTP header is controlled by the configuration of your app server. (See Additional Resources).

## Compression

In order to reduce the bandwidth demands and performance cost to deliver your site content you should look to compress the site artifacts. Browsers can accept content that has been compressed. Each app server has a different strategy for how to implement. Configuring gzip compression is highly recommended to gain approximately an order of magnitude reduction in payload. (See Additional Resources).


Unpublished work. © 2025 Siemens

Unpublished work. © 2025 Siemens

# 11. Reference

This section contains the reference documentation of Siemens Web Framework.

It is organized into the following areas:

- **Components** Reference documentation on public UI components and widgets.
- **Attributes** Reference documentation on all the public attributes.
- **Types** Reference documentation on custom types and objects used in conjunction with components and services.
- **Configuration points** Reference documentation on public JSON configuration files.
- **Services** Reference documentation on public JavaScript services.

## Components

This section contains the reference documentation of all the public Siemens Web Framework components.

> **Tip**
> For examples on how to use components refer to the **Siemens Web Framework Storybook**.

### aw-advanced-bar

Package: **@swf/composites**

A component used to represent a advanced bar.

#### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| type | AdvancedBarTypeEnum | string | No | | Represents the title of a popup. |
| orientation | AdvancedBarOrientationEnum | string | No | `"HORIZONTAL"` | Specifies the orientation of the bar. The vertical orientation is only applicable to TOOLBAR type. |


Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | TASKBAR and FOOTER are always horizontal. |
| negative | boolean | boolean | No | `"false"` | If set to `true`, background-color will be transparent. This attribute can only be applied to TOOLBAR. |

## aw-avatar

Package: **@swf/elements**

An element representing a user avatar.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of an action to execute on click. |
| source | string | object | No | | Path to an image to display, representing the user. |
| size | AvatarSizeEnum | string | No | | The size of the avatar. |
| name | ViewModelProperty | field | No | | The name of the user to display in the avatar if the `full-layout` attribute is set to `true`. This attribute will also be used to determine the initial of the user if the `initials` attribute is not set. |
| initials | string | string | No | | The two initials of the user to display instead of an image. This attribute must be set to a string of one or two characters, or it will be ignored. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| full-layout | boolean | string | No | `false` | If set to `true`, the full name of the user will be displayed. |
| text-position | TextPositionEnum | string | No | | If `full-layout` is set to `true`, this attribute is required and it is used to specify where the name of the user will be displayed, |

**Tip**

Place images you want to display in avatars in any `src/images` folder under your module, and refer to them in the `source` attribute specifying their path from the output access folder. For example: `src/images/myImage.jpg` → `assets/images/myImage.jpg`.

If you want to add images in your solution/module follow the steps below :

1. Create **image** folder inside **src** folder of module which will be parallel to **html, i18n, js** folders
2. Keep images or icons inside image folder which you want to use in the view.
3. Use path to specify **source** attribute as following: **"dbValue": "assets/image/nameOfImage.jpg"**

Framework adds all the images to **assets/image** folder in the war after the solution is built.

## aw-back-button

Package: **@swf/elements**

A component used to represent back button with command defined in the commandsViewModel. If pre-panel-id is provided, it will navigate to the panel when clicked. Otherwise it act as aw-button.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| title | string | string | No | | Name of button, displayed on it's right. |
| action | string | action | No | | Represents name of action to be executed when button is clicked. |
| pre-panel-id | string | string | No | | Id of tab to navigate when back button is clicked. |


Unpublished work. © 2025 Siemens

## aw-balloon-popup-panel

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-popup** instead

A component used to represent a balloon popup.

### Props

This component has no props.

> **Note**
> aw-balloon-popup can be configured by using **popup-options** as attributes.

## aw-break

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0.
> Use
> instead

A component used to represent a line break.

### Props

This component has no props.

## aw-button

Package: **@swf/essentials**

A component used to represent a button.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of the action to execute when the button is clicked. |
| button-type | ButtonTypeEnum | string | No | `"accent-high-contrast"` | The style of the button. |
| size | ButtonWidthEnum | string | No | `"default"` | Determines the width of the button. |

## Button placement

Buttons typically are used inside the aw-panel-footer section of **aw-panel**.

Wrapping multiple buttons in a **div** element specifying the **sw-button-group** class allows more flexibility. **sw-column** can be added to stack vertically or **sw-row** class can be used to lay buttons out horizontally.

## Button type

By default, **aw-button** defaults to a button of type **button**, i.e. a button with no default behavior for what concerns form submission. You can use the standard **type** attribute to change this:

```
<aw-button type="submit" action="someAction">Click me!</aw-button>
```

## Styling buttons

You can use the default `class` attribute to style the button *instead* of `button-type` (to specify one of the predefined **button types**) and to also specify its **size**.

By default, buttons are of size `medium`, unless a different size class is specified.

**Note:** By default *aw-button* defaults to the button type **Submit**, which is the default HTML5 button type. So when input element and aw-button element are used together in the same form and RETURN key is pressed, click action will be automatically triggered. To override the button type, please add a type attribute to aw-button like

```
    <aw-button type="button" action="someAction"> button name </aw-button>
```


Unpublished work. © 2025 Siemens

## aw-cell-header

Package: **@swf/pagecomponents**

Component to show the concise header information about the input viewModelObject.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| anchor | string | string | No | | The first anchor name for which the command bar will be displayed in the summary header. |
| header-vmo | ViewModelObject | object | Yes | | The ViewModelObject for which the summary header is supposed to be shown. The props.object_string property on VMO will be used to display the title. |
| sub-panel-context | object | object | No | | This is context object which will be passed to command bar. This can be used to write conditions for commands. |

## aw-chart

Package: **@swf/charts**

An element representing a chart.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| chart-provider | ChartProvidersConfiguration | object | Yes | | Represents an object to configure the chart. |
| chart-points | ChartPointsConfig | object | No | | Represents an object to configure the properties chart points. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Warning**

This component is part of the **@swf/charts** NPM package and requires a valid **HighCharts** license to be used. Also, you must include **afx-charts** in the `kitDeps` array of your solution `kit.json` file to use this component in your modules.

### Events

This component publishes following events:

| Event Name | EventData | EventData Description |
|---|---|---|
| *chartProviderName*.selected | { "selectedChartEntity": **ViewModelObject** } | Fired when an item of the chart is selecteds |
| *chartProviderName*.unselected | { "selectedChartEntity": **ViewModelObject** } | Fired when an item of the chart is deselected. |

## aw-cheat-sheet

Package: **@swf/essentials**

A cheat sheet is display element intended to enable the user to view keyboard shortcuts configured for commands. Cheat sheet are often used on pages that require input from a user.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| shortcut-list | object | object | No | | Additional data to be used to render shortcuts from consumer component. The format is {"shortcutKey": "Command title or Shortcut description"}. |

## aw-checkbox

Package: **@swf/elements**

An element representing a checkbox.


Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of an action to execute on click. |
| shiftSelectAction | string | action | No | | The name of an action to execute on shift+click. |
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the checkbox. |
| is-negated | boolean | object | No | `false` | If set to `true`, a different icon will be used to indicate negated selection. |

## aw-checkbox-list

Package: **@swf/elements**

**Note**
This component is deprecated since version 7.0.0. Use aw-widget with hint 'checkboxoptionlov' instead

A component used to represent list of checkbox

### Props

This component has no props.

## aw-check-list

Package: **@swf/elements**

An element representing a check list.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the check list. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Check list item configuration

Check list item can be configured as shown here in check list configuration.

## aw-chip

Package: **@swf/elements**

A component used to represent a chip or a tag. It can be configured to behave like a simple label, a button, or a selectable control.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| chip | ChipConfiguration | object | Yes | | The configuration used to configure the properties of the chip. |
| button-type | ButtonTypeEnum | string | No | `"base"` | The type of the chip. |
| action | string | action | No | | For SELECTION or BUTTON chips, it represents the name of the action to execute when the chip is clicked (for BUTTON chips) or when its selection changes (for SELECTION chips). |
| ui-icon-action | string | action | No | | For BUTTON chips, this can be set to the name of an action to execute when the UI icon is clicked. |

## aw-chip-list

Package: **@swf/elements**

A component used to represent a chip or a tag inside chip list.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| dataprovider | DataProviderConfiguration | object | Yes | | Represents a placeholder where user can configure a |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | way to fetch data and pass the same to underlying component. |
| button-type | ButtonTypeEnum | string | No | `"base"` | The type of the chip in chip list. It allows user to add different chip styles in one chip list. |
| action | string | action | No | | For SELECTION or BUTTON chips, it represents the name of the action to execute when one of the chip is clicked (for BUTTON chips) or when one of the chip selection changes (for SELECTION chips). |
| ui-icon-action | string | action | No | | For BUTTON chips, this can be set to the name of an action to execute when the UI icon is clicked. |

**Tip**

The response of the data provider must be set to an array of ChipConfiguration objects.

## aw-column

Package: **@swf/elements**

**Note**

This component is deprecated since version 6.3.0. Use aw-flex-column instead.

This component provides a container that has the height of its parent container and a width specified by the width attribute.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| width | LayoutGridSize | string | No | | Specifies the width of the column. |
| justify | JustifyContentEnum | string | No | `"top"` | Specifies column justification. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| offset | LayoutGridSize | string | No | | Defines an offset to the left of the column. |
| offset-right | LayoutGridSize | string | No | | Defines an offset to the right of the column. |
| align-content | AlignContentEnum | string | No | | Specifies alignment of content in the perpendicular direction within the column. |
| wrap-style | WrapStyleEnum | string | No | `"wrap"` | Defines a wrap style mechanism for the column. |

## aw-column-chart

Package: **@swf/charts**

**Note**

This component is deprecated since version 7.0.0. Use aw-chart with chartType 'column' instead. There is also an open-source (EChart) version of this component using the same interface called aw-column-chart2. This component will also be removed in a future release.

An element representing a column chart.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| chart-provider | ColumnChartConfiguration | object | Yes | | Represents an object to configure column chart. |

## aw-command-bar

Package: **@swf/essentials**

A container for commands, which can be aligned vertically or horizontally.

## Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| alignment | BarOrientationEnum | string | No | `"HORIZONTAL"` | The alignment of the nested commands |
| anchor | string | string | Yes | | A comma-separated list of identifiers used to identify the placement of commands, to be referenced in commandsViewModel.json files. |
| reverse | boolean | object | No | `false` | If set to `true`, then the command order is reversed. |
| show-command-labels | boolean | object | No | | If set to `true`, labels are displayed for all commands. |
| context | object | object | No | | Additional arbitrary data to include in the commandsViewModel.json to be evaluated later, when commands are executed, as *commandContext*. |
| hide-more | boolean | object | No | | If set to `true`, then the **More** button in the horizontal or vertical overflow (as applicable) is hidden even if overflow occurs. |
| overflow | boolean | object | No | | When command bar doesn't have enough space to accommodate all visible commands, this attribute will be used to control whether commands overflow in the overflow popup, or wrap on multiple lines. If set to `true`, commands will overflow, otherwise it will wrap on multiple lines. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| label-position | string | string | No | | This attribute will be used to control the command label position. |
| command-icon-size | string | string | No | | Command icon size for all commands in the command bar. |

### Configuring commands

This component represents a container for **commands**, but note that it does not contain any child component representing commands. Instead, you can contribute commands to a particular command bar (identified via its `anchor` property) using **commandsViewModel.json** files.

in Siemens Web Framework applications, `commandsViewModel.json` files can be placed in any module, and they are peers of `module.json` files. At build time, the command definitions present in all `commandsViewModel.json` in all different modules are merged together in a single file, so it is important to make sure that all identifiers used when defining commands are unique.

## aw-command-panel

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.2.0. Use **atomic data** for panel data exchange

Command panel is content container element for other UI elements such as aw-command-panel-section, aw-panel-footer, aw-panel-body etc.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| caption | string | string | No | | Display string for the command panel title. |
| hide-title | boolean | object | No | `"false"` | If set to `true`, it hides the `command-panel`. |
| anchor | string | object | No | | A comma-separated list of identifiers used to identify the placement of commands, to be referenced in `commandsViewModel.json` files. |
| context | object | object | No | | Additional arbitrary data to include in the `commandsViewModel.json` to be evaluated |


Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | later, when commands are executed, as *commandContext*. |

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-command-panel-section

Package: **@swf/elements**

**Note**

This component is deprecated since version 7.0.0. Use **AwPanelSection** instead

A logical section which can contain related UI elements which can have commands. This element should be used within an aw-command-panel.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| caption | string | string | No | | Display string for the command panel section title. |
| name | string | string | No | | Unique name of the section. This property is needed for component to publish `awCommandPanelSection.collapse` global event. |
| anchor | string | object | No | | A comma-separated list of identifiers used to identify the placement of commands, to be referenced in `commandsViewModel.json` files. |
| context | object | object | No | | Additional arbitrary data to include in the `commandsViewModel.json` to be evaluated later, when commands are executed, as *commandContext*. |
| collapsed | PanelCollapseEnum | string | No | `"false"` | When this attribute is set to true, the `command-panel-section` is displayed collapsed and when set to false, panel |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | section is displayed expanded. If this attribute is set to disabled, section will be expanded always. Default value is false. |

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-command-sub-panel

Package: **@swf/elements**

**Note**

This component is deprecated since version 7.2.0. Use **atomic data** for panel data exchange

Component used for displaying a sub panel. Used when we want to have multiple toggle enabled command sub-panels.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| panel-id | string | string | Yes | | Unique id to identify the sub-panel. Used while switching among the `sub-panels`. |
| sub-panel-context | object | object | No | | Additional arbitrary data to include in the `commandsViewModel.json` to be evaluated later, when commands are executed, as *commandContext*. |

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.


Unpublished work. © 2025 Siemens

## aw-compare2

Package: **@swf/table**

This component represents a reusable compare component which wraps a aw-splm-table.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| gridid | string | grid | Yes | | The identifier to the Grid object stored in the **grids** section of the view model and containing the table settings. |

## aw-compile

Package: **@swf/jit-view-compiler**

This component can be used to compile and render an SWF component at run time by specifying its view and view model as strings.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| comp-view | string | string | Yes | | The view representation of the component. |
| comp-viewmodel | string | string | Yes | | The view-model representation of the component. |
| comp-props | object | object | No | | A dictionary containing the props to be passed to the component bundled together in an object. |
| error-comp | string | string | No | | The fallback component that will be rendered in case of runtime exceptions |

**Warning**

This component is part of the **@swf/jit-view-compiler** NPM package. Please add an explicit dependency on this package in your project to consume this component. Also, you must include **jitViewCompilerKit** in the `kitDeps` array of your solution `kit.json` file to use this component in your modules.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Configuration tips

The `comp-props` prop expects an object containing the props that are supposed to be passed to the compiled component.

For example, consider the follogin `data` section of a view model:

```
{
"data": {
    "compProps": {
        "prop1": "Property #1",
        "prop2": {
            "enabled": true,
            "description": "..."
        }
    },
    "compView": "...the entire view of the component goes here...",
    "compViewModel": "...the entire view model of the component goes
here..."
}
}
```

When this data is passed to `aw-compile` like this:

```
<aw-compile comp-view="{{data.compView}}" comp-
viewmodel="{{data.compViewModel}}" comp-props="data.compProps"></aw-compile>
```

The resulting component instantiation (expressed in JSX syntax) will be something similar this:

```
<MyComponent prop1='Property #1' prop2={enabled: true, description:
"..."} ...></MyComponent>
```

## aw-date

Package: **@swf/elements**

An element representing a date widget.


Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the date widget. |
| quick-nav | boolean | string | No | `true` | If set to `false`, quick navigation between month and year will be disabled. |

### Date Format

You can specify the value of the date via `prop.dbValue` by using one of the following formats:

| Date Format | Date Input | Date Output | Note |
|---|---|---|---|
| DD-MMM-YYYY | 03-Mar-2018 | 03-Mar-2018 | Default format |
| DD/MMM/YYYY | 03/Mar/2018 | 03-Mar-2018 | It takes 3 types of delimeter in date(" / ", " . ", " - " eg: 03.Mar.2018), however, it will always display in default format |
| MM/DD/YYYY | 03/02/2018 | 02-Mar-2018 | |
| DD.MMM.YY | 03.Mar.18 | 03-Mar-2018 | |
| YYYY/MM/DD | 2018/09/03 | 03-Sep-2018 | |

## aw-datetime

Package: **@swf/elements**

An element representing a date-time widget.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the date-time widget. |
| quick-nav | boolean | string | No | `true` | If set to `false`, quick navigation between month and year will be disabled. |


Unpublished work. © 2025 Siemens

**Tip**

You can also configure **custom locales** to work with this component.

## aw-default-cell

Package: **@swf/elements**

This element is used in a list aw-list to display each cell. It can display an image with relevant properties alongside.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| vmo | ViewModelObject | object | Yes | | An object containing the data that will be represented by the cell. |
| hideoverlay | boolean | string | No | | If supplied `vmo` has a valid thumbnail and if the value of this property is set to `false`, then the thumbnail and typeIcon both are displayed. If the value of this property is set to `true` , then the thumbnail alone is displayed. |
| useTypeIconOnly | boolean | string | No | `false` | If the supplied `vmo` has a valid `typeIconUrl` and if the value of this property is set to `true`, then the `typeIconUrlv of the supplied vmo alone is displayed. By default, if no value is supplied, if the supplied` vmo `has a valid thumbnail and` hideoverlay `is set to` false, `then both thumbNail and` typeIconUrl` are displayed. |

## aw-default-group-cell

Package: **@swf/composites**

Default group cell has the capability of logically grouping the cells with a header. The grouping is done on the basis of the properties provided to the groupBy attribute.There can be single or multiple properties.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| item | ViewModelObject | object | Yes | | This is an object the cell accepts through `data-provider` and displays its properties. `data-provider` must return properties to be used in `group-by` prop. |
| group-by | string | string | Yes | | Properties depending on which the logical grouping of cells will be done. Can be single or multiple comma separated values. |

### Child elements

The following child elements must be present (in the same order) inside an `aw-default-group-cell` element:

- `group-header-cell`, used to define the contents of the header of each group.
- `cell`: used to define the cell to render each item in a group.

## aw-default-sublocation

Package: **@swf/pagecomponents**

**Note**
This component is deprecated since version 7.1.0. Use aw-standard-sublocation instead.

A high-level sublocation component which sets layout of page.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| sub-panel-context | object | object | No | | It allows user to share context specific information between view models. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| base-selection | object | object | No | | The default or implicit selection if the sublocation supports base selection |
| selection-model | object | object | No | | The **selection model** object passed in from the parent |
| selection-data | object | object | No | | The **selection data** (additional object required for managing the selections in PWA and SWA) which is passed down to child components. This is required only when Applications wants to override default selection management. |

## aw-dialog

Package: **@swf/essentials**

A component used to represent a dialog.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | Yes | | Represents an action of type dialog to execute on click. |

**Note**

aw-dialog can be use explicitly to have a popup like structure with more functionality such as ability to move, resize and dock/undock panel inside a page. This may be useful in many different cases, one of them being when you need to save space on screen and don't have to display sidebar at all times.

**Tip**

If you need to use configure either the width or height of a dialog, you must configure *both* properties. Also, make sure you add a scrollable container in the dialog view like **aw-scrollpanel**.

### Dialog Action Configuration

You should use a **dialog** action to open and close the dialog. This action needs to be executed on trigger of commands or links.

The dialog action allows you to specify **dialog-options** for the dialog configuration.


Unpublished work. © 2025 Siemens

# aw-echart

Package: **@swf/echarts**

An element representing a chart. Uses Apache ECharts to display a limited set of chart types.

## Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| chart-provider | ChartProvidersConfiguration | object | Yes | | Represents an object to configure the chart. |
| chart-points | ChartPointsConfig | object | No | | An object to configure the property's chart points. |

**Warning**

You must include **afx-echarts** in the `kitDeps` array of your solution `kit.json` file to use this component in your modules.

**Note**

Apache ECharts is a free open-source library. Unlike aw-chart, aw-echart does not require a 3rd-party license.

**Note**

Apache ECharts do not yet support keyboard navigation. This is expected in a future release.

**Tip**

Migration from aw-chart to aw-echart is straightforward since the chart providers follow the same pattern.

**Tip**

This component supports a limited set of chart types. To take advantage of additional ECharts capabilities, aw-echart-advanced can be used instead.

## Events

This component publishes following events:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Event Name | EventData | EventData Description |
|---|---|---|
| *chartProviderName*.selected | { "selectedChartEntity": ViewModelObject } | Fired when an item of the chart is selecteds |
| *chartProviderName*.unselected | { "selectedChartEntity": ViewModelObject } | Fired when an item of the chart is deselected. |

## aw-echart-advanced

Package: **@swf/echarts**

An element representing a native EChart.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| echart-provider | EchartProvidersConfiguration | object | Yes | | Represents an object to configure the echart. |

**Warning**

You must include **afx-echarts** in the `kitDeps` array of your solution `kit.json` file to use this component in your modules.

**Note**

Apache ECharts is a free open-source library. Unlike aw-chart, aw-echart does not require a 3rd-party license.

**Note**

Apache ECharts do not yet support keyboard navigation. This is expected in a future release.

### Events

This component publishes following events:

| Event Name | EventData | EventData Description |
|---|---|---|
| *chartProviderName*.selected | { "selectedChartEntity": ViewModelObject } | Fired when an item of the chart is selecteds |
| *chartProviderName*.unselected | { "selectedChartEntity": ViewModelObject } | Fired when an item of the chart is deselected. |

## aw-empty-workarea

Package: **@swf/composites**

An element used as a placeholder for content, when there is no data to display in a panel or work area.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| image | string | string | Yes | | The name of the icon to display in the empty work area. |
| title | string | string | No | | The title of the empty work area. |
| hint | string | string | No | | Optional hint text to display underneath the title. |
| command-id | string | string | No | | The command name for the action to be executed when the link is clicked (preferred over action) |
| action | string | object | No | | The action object to execute when the link is clicked (if `link-visibility` and `link-text` are specified). |
| link-text | string | string | No | | The text of the link to execute an action. |
| link-visibility | boolean | boolean | No | | If set to `true`, a link (configured via `link-text` and `action`) is displayed to execute an action. |

### Usage

Here is an example showing how to use `AwEmptyWorkarea` when a panel is empty. In this case, the action is displayed conditionally depending of the value of the `linkCondition` condition.

```
<div exist-when="conditions.panelIsEmptyCondition">
    <aw-empty-workarea image="imageName" title="i18n.title" hint="i18n.hint"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        action="actions.linkAction"
        link-visibility="{{conditions.linkCondition}}"
        link-text="i18n.linkText"></aw-empty-workarea>
</div>
```

## aw-expandable-row

Package: **@swf/composites**

An element representing an expandable row.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| row-height | string | string | No | | Represents height of an expandable row. |
| overflow-type | string | radio | No | | Represents overflow elements rendering strategy |

## aw-i18n

Package: **@swf/elements**

A component used to render localized text.

### Props

This component has no props.

### Contents

The content of this component must be set to a **i18n** expression pointing to a localized string defined in the view model of the container component.

### Example

Suppose that the `i18n` section of the view model of a component contains the following:

```
{
    "i18n": {
        "role": [
            "TestComponentMessages"
        ]
```

```
    }
}
```

You can display the localized text of the `role` string anywhere in the component view, using the `aw-i18n` component as follows:

```
<aw-i18n>i18n.role</aw-i18n>
```

The localized string will be fetched from the appropriate locale file based on the language of the interface (`TestComponentMessages.json` for the default locale in this example, or with the appropriate locale suffix for other locales).

## aw-file-upload

Package: **@swf/elements**

A component which allows user to browse and upload file.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| type-filter | string | string | No | | A comma-separated list of file extension names (e.g. `.gif, .png, .jpg`)used to filter the type of files do display in the file browse dialog. |
| file-change-action | string | action | No | | The name of the action to execute when a file is uploaded. |
| multiple | boolean | boolean | No | `false` | If set to true, it enables to select multiple files to upload. |
| remove-tooltip | string | string | No | | Message to be displayed when hovering over the remove icon of an uploaded file. |
| upload-model | UploadModel | object | No | | An object containing the model(s). It can be used to configure widget with different upload models. A typical model comprise of name, icon, isFolder, etc. Refer filePickerConfiguration. |
| selection-data | UploadedFiles | object | No | | An atomic data object containing the File objects and metadata on uploaded files. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | It can be used to access uploaded files programmatically. |

## Handling uploaded files

The action specified in the `file-change-action` will be invoked with a **UploadedFileMetadata** object that will be accessible in expressions via `parameters`, as follows:

```
{
    "handleUploadedFile": {
        "actionType": "JSFunction",
        "method": "getSelectedFileInfo",
        "inputData": {
            "fileName": "{{parameters.fileName}}",
            "fileNameNoExt": "{{parameters.fileNameNoExt}}",
            "fileExt": "{{parameters.fileExt}}",
            "formData": "{{parameters.formData}}",
            "data": "{{data}}"
        },
        "outputData": {},
        "deps": "js/myExampleService"
    }
}
```

## Events

This component publishes the events described in the following sections.

### invalidFileSelected

This event is published with an empty eventData object when the selected file type is not valid.

## Typical configuration for defining upload-models : filePickerConfiguration.json

```
{
    //key 1
    "add": {
        // model definition 1
        "localfile":
        {
            "name": "Upload File",
            "priority": 1,
            "iconName": "cmdFileUpload16"
        },
        // model definition 2
        "localfolder":
        {
            "name": "Upload Folder",
            "priority": 2,
```

```
            "iconName": "cmdFolderUpload16",
            "isFolder": true
        }
    },
    //key 2
    "replace": {
        //model definition
        "localfile":
        {
            "name": "",
            "priority": 1,
            "iconName": "typeSearchFolder48",
            "popupViewId": "ABCView",
        }
    }
}
```

## aw-flex-column

Package: **@swf/elements**

This component provides a container that has the height of its parent container and a width specified by the width attribute.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| align-content | AlignContentEnum | string | No | | Specifies alignment of content in the perpendicular direction within the column. |
| offset | LayoutGridSize | LayoutGridSize | No | | Defines an offset to the left of the column. |
| offset-right | LayoutGridSize | string | No | | Defines an offset to the right of the column. |
| wrap-style | WrapStyleEnum | string | No | `"wrap"` | Defines a wrap style mechanism for the column. |
| width | LayoutGridSize | string | No | | Specifies the width of the column. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| justify | JustifyContentEnum | string | No | "top" | Specifies column justification. |
| when | ResponsiveLayoutGridSizes | string | No | | Used to specify the size of a column depending on device size. |

## aw-flex-row

Package: **@swf/elements**

This component provides a container that has the width of its parent container and a height specified by the height attribute.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| align-content | AlignContentEnum | string | No | | Specifies alignment of content in the perpendicular direction within the row. |
| offset | LayoutGridSize | string | No | | Defines an offset to the top of the column. |
| offset-bottom | LayoutGridSize | string | No | | Defines an offset to the bottom of the column. |
| height | LayoutGridSize | string | No | | Specifies the height of the row. |
| wrap-style | WrapStyleEnum | string | No | "wrap" | Defines a wrap style mechanism for the row. |
| justify | JustifyContentEnum | string | No | "top" | Specifies row justification. |

### Differences with aw-row and the previous grid system

- Previously, offsets used to create new empty elements. The offsets could be given proportional or fixed demical and integer values. With the new grid system, the offsets use margin and do not create any new empty elements.
- Previously, offsets could be proportional in size or have a fixed value from 12 predefined fixed values. The fixed values have a size of 1em, 2em, 3em....upto 12em which use the fixed value units. These are specified using the attribute value of 1f, 2f, 3f...12f. Fixed decimal offsets are now deprecated in favor of proportional and fixed integer offsets.
- The justification and wrap style properties do not touch the parent containers. They get applied to the children of current container only.


Unpublished work. © 2025 Siemens

- Offsets use margin and do not create new empty elements.
- Justify use flex-based justification and do not create new empty elements.

## aw-frame

Package: **@swf/elements**

A component used to represent a frame.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| url | string | string | Yes | | URL of the content to display in a frame. |
| frame-title | string | string | Yes | | Specifies the title of the frame. |
| height | string | string | No | | Specifies the height of the frame. |
| width | string | string | No | | Specifies the width of the frame. |

## aw-full-screen

Package: **@swf/composites**

A component used to represent a full screen mode.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| header-title | string | string | No | | Represents the title of content being full-screened. Usually, it is a descriptive label, like the label of a tab or section. This is used to build the default full screen header. |
| header-path | string | string | No | | The path the user takes to navigate to the full-screened content. It is usually based on the information architecture. This is used to build the default full screen header. |
| header-context-title | string | string | No | | Represents the contextual object that is opened or selected, or data scope. This is used to build the default full screen header. |


Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| include-component-name | string | string | Yes | | This represents the name of the component which needs to be shown in the full screen. The included component will receive 'fullScreenState' on 'props.subPanelContext' which needs to be passed to AwCommandBar with 'aw_fullScreen' anchor so that Enter Full Screen command is shown. |
| context | object | object | No | | Represents the context object which will be passed to included component and header. This can be used to write conditions for header slot or commands. |

## Adding the Full Screen command to your view

To display the **Full Screen** command, you need to add an **aw-command-bar** component to your view, and set its `anchor` prop to `aw_fullScreen`, like this:

```
<aw-command-bar
    anchor='aw_fullScreen'
    alignment='HORIZONTAL'
    context='props.subPanelContext'>
</aw-command-bar>`
```

If you want to add the **Full Screen** command to an existing command bar, simply pass `aw_fullScreen` as an additional anchor, like this:

```
<aw-command-bar
    anchor='my_anchor,aw_fullScreen'
    alignment='HORIZONTAL'
    context='props.subPanelContext'>
</aw-command-bar>`
```

## Configuring the full screen header

The default full screen header can be configured via the `aw_fullScreenHeader` **layout slot**. By default, the full screen header will display the full screen props in the following order:

1. header-title
2. header-path
3. header-context-title

Additionally, the header will also include the **Exit Full Screen** command by default.

If you decide to override the default full screen header via the `aw_fullScreenHeader` layout slot, you must configure the appropriate `activeWhen` condition so that the slot is activated properly. Also, you should include a command bar in your view specifying the `aw_fullScreenHeader` anchor so that the **Exit Full Screen** command is displayed, like this:

```
<aw-command-bar
    anchor='aw_fullScreenHeader'
    alignment='HORIZONTAL'
    context='props.subPanelContext'>
</aw-command-bar>`
```

## aw-guidance-message

Package: **@swf/elements**

A component used to represent a guidance-message.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| message | MessageConfigurationObject | message | Yes | | Represnts message type, message text and message text links(if necessary) defined in message section. |
| closable | boolean | boolean | No | `false` | If set to `true`, user can see the close button. |
| banner-style | boolean | boolean | No | `true` | If set to `false`, guidance message will not be shown in banner style. |
| show-icon | boolean | boolean | No | `true` | If set to `false`, icon in message will not appear. |
| show-type | boolean | boolean | No | `true` | If set to `false`, type of guidance message will not be shown. |
| icon | string | string | No | | It displays the custom icon and override the default icon used for a particular type of guidance message. This custom icon will be |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | shown when only show-icon is `true`. |
| close-action | string | action | No | | The action to execute when guidance message is dismissed. This action is not executed if closable is set to `false`. |

**Note: **

**Note: Only aw-button can be used in the transclusion**

### Usage notes

- `{@msgTxtLink0}` will be converted into an **aw-link**.
- Only **aw-button** can be used in transclusion.

## aw-header-context

Package: **@swf/pagecomponents**

A component used to represent a header context.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| alignment | HeaderContextAlignmentTypeEnum | string | No | | Represents the alignment for header context. |
| anchor | object | object | No | | Specifies an anchor which is used for pulling context contribution from the context contribution model. |

## aw-header-props

Package: **@swf/pagecomponents**

An element representing a header props.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| header-properties | Array.HeaderPropertiesConfiguration | object | Yes | | Represents an array of object header props configuration. |
| headerproperties | Array.HeaderPropertiesConfiguration | object | No | | Represents an array of object header props configuration. |

## aw-icon-button

Package: **@swf/essentials**

A component used to represent an icon image button with command defined in the commandsViewModel. If destPanelId is given in command, it will navigate to the panel by clicking it. Otherwise it act as aw-button.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| command | IconButtonConfiguration | field | Yes | | Represents an object to configure the behavior of IconButton |
| size | ButtonSizeEnum | string | No | `"default"` | Determines the size of the icon button. |

**Note**

This component may publish or subscribe to some events to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of Showcase.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# aw-image

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-pic** instead

An element representing an image.

## Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| source | string | object | Yes | | Path to the image to display. |
| is-icon | boolean | object | No | `false` | If set to `true`, it renders the image as an icon. |

> **Tip**
> Place images you want to display in any `src/image` folder under your module, and refer to them in the `source` attribute specifying their path from the output access folder. For example: `src/image/myImage.jpg` → `assets/image/myImage.jpg`.

If you want to add images in your solution/module follow the steps below :

1. Create **image** folder inside **src** folder of module which will be parallel to **html, i18n, js** folders.
2. Keep images or icons inside image folder which you want to use in the view.
3. Use path to specify **source** attribute as following: **"dbValue": "assets/image/nameOfImage.jpg"**

Framework adds all the images to **assets/image** folder in the war after the solution is built.

# aw-image-cell

Package: **@swf/elements**

This element is used in a list aw-list to display each image cell. It can display an image with relevant description on the bottom.


Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| vmo | ViewModelObject | object | Yes | | An object containing the data that will be represented by the cell. |

## aw-include

Package: **@swf/engine**

A component which allows the user to load declarative View and View model on demand

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| name | string | string | Yes | | Represents key name for the View and View Model to demand load. |
| view-id | string | string | No | | Represents the Id which is used to uniquely identify the rendered view. |
| sub-panel-context | object | object | No | | It allows user to share context specific information between ViewModels. |

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-label

Package: **@swf/elements**

An element representing a label.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the label. |

## aw-layout-container

Package: **@swf/composites**

An element representing Layouts.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| name | string | string | Yes | | It is used to identify the the layout configuration |
| context | object | object | No | | It allows user to share layout specific information between view models. |

## aw-layout-slot

Package: **@swf/engine**

A component used to represent a layout slot.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| name | string | string | Yes | | Represents key name for slot name which is defined in layoutSlots.json. It is recommended that user should use the name format 'viewname:uniqueID'. This will make sure that anchor cannot be repeated. |
| context | object | object | No | | Specifies the information which needs to be passed on to the child layout file. |


Unpublished work. © 2025 Siemens

**Note**

This element can be used to load declarative view and view model on demand with provided conditions. It is recommended to use it for overriding any existing view/viewmodel with another view/viewmodel.

## aw-link

Package: **@swf/essentials**

A component used to represent a link.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the link. |
| dataprovider | DataProviderConfiguration | object | Yes | | Represents a placeholder where user can configure a way to fetch data and pass the same to underlying component. |
| action | string | action | Yes | | The name of an action to execute on click. |
| link-options | LinkOptionsConfiguration | object | No | | Represents an object used to configure the link options. |
| command-id | string | string | No | | If the command-id is given, it will execute the command handler on click of the link. |

## aw-link-with-popup

Package: **@swf/essentials**

**Note**

This component is deprecated since version 7.0.0. Use **aw-link** instead

A component used to display a popup on click of a link.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

This component has no props.

## aw-link-with-popup-and-icon

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-link** instead

An element representing AwLink with popup and icon.

### Props

This component has no props.

## aw-link-with-popup-menu

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-link** instead

A component used to display a popup-menu on click of a link.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the link with popup-menu. |
| dataprovider | DataProviderConfiguration | object | Yes | | Represents a placeholder where user can configure a way to fetch data and pass the same to underlying component. |

| | | | | | |
|---|---|---|---|---|---|
| disable | boolean | object | No | `false` | If set to `true`, it will disable the link with popup menu. |
| id | string | string | No | | Represents the id of the element. |
| link-popup-id | string | string | No | | Represent the id of link popup. |
| link-popup-menu-id | string | string | No | | Represent the id of link popup menu. |
| use-icon | boolean | string | No | | If set to true, the component will show an icon instead of the link text. |

**Tip**

The response of the data provider must be set to an array of **LinkPopupResponsePropertyValue** objects.

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-list

Package: **@swf/elements**

A component used to represent data is paged in from the specified data provider when the user scrolls.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| dataprovider | **DataProviderConfiguration** | object | Yes | | Represents a placeholder where user can configure a way to fetch data and pass the same to underlying component. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| show-check-box | boolean | string | No | `false` | If set to `true`, the checkbox is appear in front of list item. |
| item-options | object | object | No | | Item options is an object with predefined actions, that get called on click of the list command. |
| list-options | object | object | No | | List options is an object with predefined actions, that are used with entire list like selection. |
| show-context-menu | boolean | string | No | `false` | If set to `true`, then the right click context menu is available, else it doesn't honor right click. |
| fixed-cell-height | string | string | No | | If the height is given, list item will take fixed height. |
| show-drop-area | boolean | boolean | No | `false` | If set to `true`, drop area will be visible for the dragged element. |
| is-group-list | boolean | boolean | No | `false` | If set to `true`, then list items are arranged in logical group. |
| has-floating-cell-commands | boolean | boolean | No | `true` | If set to `false`, then the cell commands will always appear. |
| show-decorators | boolean | boolean | No | `false` | If set to `true`, then the decorators styling is applied on the list items. |
| command-context | object | object | No | | This represents an object consumed by commands inside the list items. |
| double-click-action | string | action | No | | The name of the action to be executed when a list item is double clicked. |
| loading-style | string | string | Yes | | Used to provide style for default image skeleton. |


Unpublished work. © 2025 Siemens

**Tip**

The response of the data provider must be set to an array of objects. The object can be of any type.

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-listbox

Package: **@swf/elements**

A component used to represent a listbox.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the listbox. |
| list | Array.PropertyValue | object | No | | Represents the array of items in the listbox. |
| action | string | action | No | | The name of the action to execute whenever there is a change in selection from drop down menu. |
| is-select-only | boolean | string | No | `false` | If set to 'true', the user can only select from the list items and no search box is provided. |

## aw-list-filter

Package: **@swf/elements**

A component used to filter data inside the list.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the list filter. |
| dataprovider | DataProviderConfiguration | object | Yes | | Represents a placeholder where user can configure a way to fetch data and pass the same to underlying component. |
| item-options | object | object | No | | Item options is an object with predefined actions, that get called on click of the list command. |
| is-group-list | boolean | object | No | `false` | If set to `true`, then list items are arranged in logical group. |
| double-click-action | string | action | No | | The name of the action to be executed when a list item is double clicked. |
| command-context | object | object | No | | This represents an object consumed by commands inside the list items. |

**Tip**

The response of the data provider must be set to an array of objects. The object can be of any type.

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-navigate-breadcrumb

Package: **@swf/elements**

A component used to represent a breadcrumb.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| crumbs | CrumbsConfiguration | object | Yes | | Represents an array of crumbs. |
| compact | boolean | string | No | `false` | If set to `true`, root crumb acts like a title for the breadcrumb. |
| popuplist | string | string | No | | The user defined component to be shown when a chevron is clicked. |
| is-last-crumb-editable | boolean | string | No | `false` | If set to `true`, last crumb is displayed as textbox |

## aw-navigate-panel

Package: **@swf/elements**

**Note**
This component is deprecated since version 7.0.0. Use **atomic data** for panel data exchange

A component used to navigate panels

### Props

This component has no props.

## aw-notification-cell

Package: **@swf/composites**

A component used to represent a notification cell.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| vmo | object | object | Yes | | An object containing the data that will be represented by the notification cell. |
| item | object | object | No | | An object containing the data that will be represented by the notification cell. |

## aw-numeric

Package: **@swf/elements**

**Note**

This component is deprecated since version 7.0.0. Use **aw-widget** with hint 'integer' instead

This element is used to represent a numeric input control that allows you to enter a numeric values (floating point or integer).

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the component. |

### Configuration tips

You can set `prop.type` to `INTEGERARRAY` or `DOUBLEARRAY` and `prop.isArray` to `true` to use this control to edit an array of numeric values.

## aw-panel

Package: **@swf/elements**

This element is used as a container to hold other elements like aw-panel-header, aw-panel-body, aw-panel-section, aw-panel-footer etc.


Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| caption | string | string | No | | Represents the name of the `aw-panel` container. |
| attributes | string | object | No | | Represents the set of valid `html-form` attributes as `aw-panel` essentially returns a `html-form` component. |
| reveal-action | string | action | No | | Represents the name of function to be executed when the component renders. |

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.

## aw-panel-section

Package: **@swf/elements**

This element is a container level element and used inside an aw-panel component.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| caption | string | string | No | | Represents the name displayed on the panel section. |
| anchor-position | string | string | No | `"LEFT"` | This is a flag which will make sure that if set to 'RIGHT' anchors will stay in panel title even when labels are on |
| name | string | string | No | | Unique name to identify panel section. |
| collapsed | **PanelCollapseEnum** | string | No | `"false"` | When this attribute is set to true, the panel section is displayed collapsed and when set to false, panel section is displayed expanded. If this |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | attribute is set to disabled, section will be expanded always. Default value is false. If caption is not set, collapsed will be ignored and section will not be collapsible. |
| toggle-action | string | action | No | | Specifies name of function to be executed every time panel section is toggled. |
| include-component-name | string | string | No | | When include component name is provided then corresponding component will be loaded and displayed in the panel section. |
| hide-commands-when-collapsed | boolean | boolean | No | `"false"` | When this attribute is set to true, the commands will be hidden when the panel section is collapsed. |

## aw-password

Package: **@swf/elements**

This element is used to represent a password input control that allows you to enter a password securely by obscuring inputted text.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of the action to execute when the ENTER key is pressed. |
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the password. |

## aw-pic

Package: **@swf/essentials**

This is a low-level element for inserting an image from a given source URL or icon name. By default, it is auto-sized based on its container with no padding or margin. Spacing is typically added on this element

based on the container context. Sizing is controlled by either constraining the container or by including an explicit class.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| source | string | object | No | | Represents URL to the image. |
| icon-id | string | object | No | | ID of icon in icon library. |
| alt | string | object | No | | Alternate text for the image if it can't be displayed. Used with source, not with icon-id. |
| class | PicClassEnum | string | No | | Used to set a picture size. Default is to omit class and pic will auto-size. |

### Adding custom images to your projects

If you want to add images in your solution/module follow the steps below :

1. Create **image** folder inside **src** folder of module which will be parallel to **html, i18n, js** folders
2. Keep images or icons inside image folder which you want to use in the view.
3. Use path to specify **source** attribute as following: **"dbValue": "assets/image/nameOfImage.jpg"**

Framework adds all the images to **assets/image** folder in the war after after the solution is built.

## aw-popup

Package: **@swf/essentials**

A component used to represent a popup.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| options | PopupOptions | popup | Yes | | Represents an object to configure a popup. |
| caption | string | string | No | | Represents the title of a popup. |
| commands | object | object | No | | Additional arbitrary data to include in the `commandsViewModel.json` to be |


Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | evaluated later, when commands are executed, as *commandContext*. |
| anchor | string | string | No | | A comma-separated list of identifiers used to identify the placement of commands, to be referenced in commandsViewModel.json files. |
| is-modal | boolean | string | No | `true` | If set to `false`, popup will not be shown in modal view. |
| context | object | object | No | | Specifies the information which needs to be passed on to the child layout file. |

**Tip**
This component is typically used in conjunction with a **popup action**.

## aw-popup-close-button

Package: **@swf/essentials**

**Note**
This component is deprecated since version 7.0.0. none, not used anymore

A component used to represent a popup services

### Props

This component has no props.

## aw-popup-draggable

Package: **@swf/essentials**

**Note**
This component is deprecated since version 7.0.0. Use **aw-popup** with draggable option instead

A component used to represent a popup with draggable option

### Props

This component has no props.

## aw-popup-panel

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-popup** instead

A component used to represent a popup panel

### Props

This component has no props.

## aw-popup-panel2

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-popup** instead

A component used to provide popup additional services

### Props

This component has no props.

## aw-popup-primary-button

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-button** instead

A component used to represent a popup additional services


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

This component has no props.

## aw-popup2

Package: **@swf/essentials**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-popup** instead

A component used to represent nested popup.

### Props

This component has no props.

## aw-progress-indicator

Package: **@swf/elements**

The element can be used to show a non-deterministic progress (busy) indicator. It uses event mechanism to start and stop showing the indicator. When the element receives the progress.start event it will be displayed and as soon as it receives the progress.end event it will be hidden.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| name | string | string | No | | Name to uniquely identify progress indicator component. Must be provided if we have multiple `aw-progress-indicator` components in same view, to be able to subscribe to their own specific `progress.start` and `progress.end` events. |

### Events

This component listens to the events described in the following sections.

### progress.start

This event can be used to *start* showing a non-deterministic progress (busy) indication.

No eventData needed.

### progress.stop

This event can be used to *stop* showing a non-deterministic progress (busy) indication.

No eventData needed.

```
Tip If you have given a name to your progress indicator
(eg. name="myIndicator"), this element subscribes to the myIndicator-
progress.start and myIndicator-progress.end respectively.
```

## aw-property

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-widget** instead

A component representing a property.

### Props

This component has no props.

## aw-property-boolean

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-widget** with type boolean instead

A component used to represent a boolean property


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

This component has no props.

## aw-property-image

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use aw-icon instead

A component used to represent image

### Props

This component has no props.

## aw-property-numeric

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-widget** with type integer instead

A component used to represent a Numeric property

### Props

This component has no props.

## aw-property-string

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-widget** with type string instead

A component used to represent a string property


Unpublished work. © 2025 Siemens

### Props

This component has no props.

## aw-property-string-val

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-widget** with type string instead

A component used to represent a string property

### Props

This component has no props.

## aw-property-val

Package: **@swf/elements**

> **Note**
> This component is deprecated since version 7.0.0. Use **aw-widget** instead

A component representing a property value.

### Props

This component has no props.

## aw-radiobutton

Package: **@swf/elements**

A control representing a group of radio buttons. It can be aligned horizontally or vertically and supports a variety of data types, although it is typically used for Boolean values.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the radio button. |
| list | Array.PropertyValue | object | Yes | | An array of values to define all the radio button group options. Use this property if the group of radio buttons is not used to represent a Boolean value. |
| action | string | action | No | | The name of an action to execute when an option is selected. |

### Configuration tips

- You can set `prop.vertical` to `true` to use to configure alignment of radio button in a the form of vertical layout.
- You can set `prop.isEditable` to `true` to configure true/false labels for radio button.
- You can set `prop.isArray` to `true` to configure Array of boolean values for radio button.

## aw-row

Package: **@swf/elements**

**Note**

This component is deprecated since version 6.3.0. Use aw-flex-row instead.

This component provides a container that has the width of its parent container and a height specified by the height attribute.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| height | LayoutGridSize | string | No | | Specifies the height of the row. |
| justify | JustifyContentEnum | string | No | `"top"` | Specifies row justification. |
| offset | LayoutGridSize | string | No | | Defines an offset to the top of the column. |

| | | | | | |
|---|---|---|---|---|---|
| offset-bottom | string | string | No | | Defines an offset to the bottom of the column. |
| align-content | AlignContentEnum | string | No | | Specifies alignment of content in the perpendicular direction within the row. |
| wrap-style | WrapStyleEnum | string | No | `"wrap"` | Defines a wrap style mechanism for the row. |

## aw-scrollpanel

Package: **@swf/elements**

An element that provides a scrollable panel when the content overflows. Container for additional elements and content.

### Props

This component has no props.

### Usage notes

- Use scroll panel only when content overflows.
- Vertical scrollbars are always on the right.
- Horizontal scrollbars are always at the bottom.
- Scroll panel automatically appears or disappears based on how the content fits in the container.

## aw-selection-summary

Package: **@swf/pagecomponents**

this component render AwSelectionSummary .

### Props

This component has no props.

]

It also reacts to selection change & tab change and renders the corresponding view.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Defining secondaryWorkareaTabs.json

```
    {
            "schema": "1.0.0.0",
            "tabs": {
                "movieSummary":{
                    "name": "Movie Summary",
                    "priority": 100,
                    "view": "AwMovieSummary",
                    "visibleWhen": "selection.length === 1"
                },
                "movieDetails":{
                    "name": {
                        "source": "/i18n/UIMessages",
                        "key": "movieDetails"
                    },
                    "priority": 200,
                    "view": "AwMovieDetails",
                    "visibleWhen": {
                        "condition":
"conditions.shouldMovieSummaryBeVisible"
                    }
                }
            },
            "conditions": {
                "shouldMovieSummaryBeVisible": {
                    "expression": {
                        "$and": \[ {
                                "$source": "selection.length",
                                "$query": {
                                    "$gt": 0
                                }
                            },
                            {
                                "$source": "selection",
                                "$query": {
                                    "$all": {
                                        "$source":
"modelType.typeHierarchyArray",
                                        "$query": {
                                            "$in": \[ "ItemRevision",
"Item" \]
                                        }
                                    }
                                }
                            }
                        \]
                    }
                }
            }
        }
```


Unpublished work. © 2025 Siemens

In secondaryWorkareaTabs.json, tabs objects need to be defined. All such 'tabs' objects are combined from all secondaryWorkareaTabs.json and are rendered as tabs. If only one tab comes to be visible tab, then tab header is not shown. Instead its view is loaded.

**Details of the tabs object:**

| Field | Description |
|---|---|
| Key | It is a mandatory value which indicates unique identifier for a tab. |
| name | It is mandatory input which represents name of the tab that will appear on the UI. |
| priority | For all visible tabs, priority indicates the order in which the tabs will appear. |
| view | When the tab is selected, view specified here will be loaded. E.g. in above case AwMovieSummaryView.html & AwMovieSummaryViewModel.json are loaded. |
| visibleWhen | This represents condition which will be evaluated to determine if tab will be visible or not. Condition expression can use, selection, context and ctx. |

**Inteface for AwSelectionSummary Component:**

| Field | Description | Mandatory |
|---|---|---|
| selection | It represents selection object. This could be sigle object or an array of selection objects. | Yes |
| context | This represents additional context information. Setting context.pageContext.sublocationState.secondaryActiveTabId to a tab name makes that tab as selected | No |
| selection-data | It represents selection in a loaded view when tab is selected | No |

**Programmatic selection of tab:** In order to programmatically select a tab, consumer needs to set the prop context.pageContext.sublocationState.secondaryActiveTabId to tab name. This triggers the tab change & tab name speficied as a value gets activated.

## aw-search-box

Package: **@swf/elements**

A component used to represent a search box.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of the action to execute on click. |
| reset-action | string | action | No | | This action executes on click of close button. It is activated when search box contains text. |
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the search box. |
| change-action | string | action | No | | This action executes on the basis of onChange event of the search box input. |

**Note**

The search box element takes a property to bind to the search string and an action as parameters. On click of search icon, the search action is called.

## aw-separator

Package: **@swf/elements**

A component used to represent a separator.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| is-vertical | boolean | string | No | `true` | If set to `false`, the adjacent elements will get separated horizontally. |

## aw-session-controls

Package: **@swf/pagecomponents**

This component is used to display the user avatar and user session controls. It is positioned on the top-right corner of the screen (adjacent to the Siemens logo), and it contains the **aw_avatar** and **aw_userSessionbar** layout slots.


Unpublished work. © 2025 Siemens

### Props

This component has no props.

**Note**

Commands added to the **aw_userSessionbar** anchor will not overflow automatically. To replicate the overflow behavior, you can configure a group command using the **cmdMore** icon ID.

## aw-sidenav

Package: **@swf/essentials**

**Note**

This component is deprecated since version 6.3.0. Use **aw-dialog** instead.

A component used to represent a side navigation panel.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| config | SideNavConfigObject | object | Yes | | Configuration object defining the behaviour of the sidenav component. |
| id | string | string | Yes | | Represents a unique identifier associated with the sidenav. This id is used in the events that opens the sidenav from the event. |
| sub-panel-context | object | object | No | | It allows user to share context specific information between view models. |

**Note**

This component may publish or subscribe to some **events** to provide certain behaviors. Although you should typically rely on component props or alternative APIs, more information on these events can be found in the API Documentation page for this component on previous versions of **Showcase**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# aw-source-editor

Package: **@swf/source-editor**

A component used to represent a source editor.

## Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| name | string | string | Yes | | Represents the name of the editor. |
| content | ViewModelProperty | field | No | | Represents the content of the editor. |
| config | SourceEditorConfig | object | No | | Represents an object to configure the editor. |
| editor | Field<MonacoEditor> | object | No | | Represents the instance of the Monaco editor. This prop must be set to an empty Field (atomic data) object that will be updated with a reference to the Monaco editor object once it is loaded. |
| global-config | Field<MonacoGlobalApi> | object | No | | Represents the global Monaco object. This prop must be set to an empty Field (atomic data) object that will be updated with a reference to the Monaco configuration object once the editor is loaded. |

## Specifying support for a custom language

By default `aw-source-editor` supports syntax highlighting and language features specific to JavaScript. To enable one of the **many languages** supported out-of-the-box by Monaco, such as *markdown*, proceed as follows:

1. Instantiate `aw-source-editor` in your view making sure you specify the `global-config` property.

```
<aw-source-editor
    content="props.text"
    config="data.editorConfig"
    name="{{props.name}}"
```

```
    global-config="data.monacoConfig">
</aw-source-editor>
```

2. In the corresponding view model, configure `monacoConfig` in the `data` section as **atomic data**, and configure an `onUpdate` hook to be executed when the `monacoConfig` field is updated (note that the `monacoConfig` field that will be populated by Monaco editor shortly after loading):

```
{
    "schemaVersion": "1.0.0",
    "actions": {
        "init":{
            "actionType": "JSFunctionAsync",
            "inputData": {
                "config":"{{fields.monacoConfig}}"
            },
            "outputData": {
            },
            "method": "init",
            "deps": "js/markdownEditorService"
        }
    },
    "data": {
        "monacoConfig": {
            "initialValues": {},
            "meta": {}
        },
        "editorConfig": {
            "options": {
                "automaticLayout": true,
                "readOnly": false,
                "lineNumbers": "off",
                "wordBasedSuggestions": false,
                "minimap": {
                    "enabled": false
                }
            },
            "height": 500
        }
    },
    "lifecycleHooks": {
        "onUpdate": [
        {
            "action": "init",
            "observers": [
                "fields.monacoConfig"
            ]
        }
    ]
    }
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

3. In a JavaScript service file (`js/markdownEditorService` in this case), define the `init` method to load the markdown language and set it as Monaco's model language:

```
export const init = async( configField ) => {
    const monaco = configField.getValue();
    if ( monaco.editor ) {
        // Import markdown language settings
        await import( 'monaco-editor/esm/vs/basic-languages/markdown/
markdown.contribution' );
        // Set model language to markdown
        monaco.editor.setModelLanguage( monaco.editor.getModels()[0],
'markdown' );
        // Disable built-in JS support

monaco.languages.typescript.javascriptDefaults.setCompilerOptions( { noL
ib: true } );
    }
};
```

## aw-slider

Package: **@swf/elements**

A component used to allow a single selection from a range of values, shown along a horizontal or vertical bar.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the slider. |

**Note**

The ViewModelProperty used to configure this component must have a `dbValue` property set to an array of SliderValue objects and have the `isArray` property set to `true`.

## aw-splitter

Package: **@swf/essentials**

A component used to represent a splitter that can be used to separate and resize different areas of the page.


Unpublished work. © 2025 Siemens

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| min-size-1 | string | string | No | `"20px"` | Represents the height/width of top/right adjacent elements. |
| min-size-2 | string | string | No | `"20px"` | Represents the height/width of bottom/ left adjacent elements. |
| direction | string | string | No | `"VERTICAL"` | Specifies if aw-splitter is horizontal or vertical. |
| is-primary-splitter | boolean | boolean | No | `"false"` | If set to `true`, while view mode changes inside sublocation, splitter will update accordingly. It is only applicable if splitter gets used in sublocations. |
| action | string | action | No | | The name of action to execute when the splitter is resized. |
| splitter-id | string | string | No | | If `is-primary-splitter` is set to `true` and there are more than one primary-splitter on a page, then each splitter has given a particular id to get themselves updated to persist their positions while user resized any of the splitter on view mode. |
| show-on-hover | boolean | boolean | No | `"false"` | If set to `true`, splitter will be visible only on hover. |

## aw-splm-table

Package: **@swf/table**

This component can be used to represent a complex interactive table supporting advanced functionalities like pagination, sorting, filtering, editing, and a hierarchical view.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| gridid | string | grid | Yes | | The identifier to the Grid object stored in the **grids** section of the view model and containing the table settings. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| use-tree | boolean | object | No | `false` | If set to `true`, the table will be rendered as a hierarchical table. |
| show-check-box | boolean | object | No | `false` | If set to `true`, a check box will be added as a first column of the table (useful for multiple selection). |
| show-context-menu | boolean | object | No | `false` | If set to `true`, enables right-click context menu for the table. |
| show-decorators | boolean | object | No | `false` | If set to `true`, the decorators styling is applied to table rows (useful if any cell command using decorators is used; see **Command** for more information). |
| command-context | object | object | No | | Arbitrary context data that will be made available to any command triggered from the table (cell commands, settings, column menu, context menu). |
| reusable | boolean | boolean | No | `false` | If set to `true`, the table configuration specified in the **Grid** object referenced by the `gridid` attribute can be reused multiple times (requires the `table-context` and `selection-data` attributes to be configured correctly). |
| table-context | **TableContext** | object | No | | An object used to configure some settings of the table if the `reusable` attribute is set to `true`. |
| selection-model | object | object | No | | The **selection model** object passed to the table to manage selection configuration. |
| selection-data | object | object | No | | The **selection data** object passed to the table to manage selection state. |

## Events

| Name | Input Data | Description |
|---|---|---|
| *gridId*.plTable.clientRefresh | n/a | Refreshes the table with the data currently in the data provider on the client. This is necessary to reflect client side changes to properties on a view model object in the table. |

| | | |
|---|---|---|
| *gridId*.plTable.reload | n/a | Reloads the table with the data retrieved by the data provider load action. |
| *gridId*.plTable.updated | { updatedObjects: Array.<ViewModelObject> } | Updates the corresponding table rows with the latest server data retrieved for the given view model objects. Note that each ach view model object must have a uid. |

## Custom header cells and columns

aw-splm-table supports three ways to configure custom header cell and column templates which are useful if you want some custom cell display, such as icons, vertical text, etc.

To use any custom property renderer templates, you must first create a **propertyRendererTemplates.json** file. This will be a sibling to the `commandsViewModel.json` and `module.json` files.

For more information, see the **Customized Cells - Basic**, **Customized Cells - Advanced**, and **Customized Headers** examples in Storybook.

## Editing table data

Clicking on the edit command of a table will enable you to make bulk edits to all editable cells in the table. These changes must be saved or discarded by clicking the cancel or save edit command.

Alternatively, individual cell editing allows you to make changes directly to cells without clicking on the edit command first. Changes to a cell will be automatically saved if the **Autosave Tables** toggle is enabled.

**Note**

Individual table cell editing will automatically be enabled and function if you have configured the edit config in your table view model. If your table supports editing, but you want to disable individual cell editing, set the `enablePropEdit` dataProvider property to `false` or a condtion.

Instructions for Autosave On Mode:

To edit with autosave on:

1. Open the user settings panel and enable the **Autosave Tables** toggle.
2. Click on a cell to select it.
3. If the cell border is blue, it is editable (a grey border indicates the cell is not editable).
4. Click on the cell again to start editing it.
5. Click anywhere outside of the cell to automatically save your changes.


Siemens Web Framework
Unpublished work. © 2025 Siemens

To edit with autosave off:

1. Open the user settings panel and disable the **Autosave Tables** toggle.
2. Click on a cell to select it.
3. If the cell border is blue, it is editable (a grey border indicates the cell is not editable).
4. Click on the cell again to start editing it.
5. When finished editing cells, click on the **Save** or **Cancel** edit command.

### Reusable table

If the `reusable-table` prop of aw-splm-table is set to `true`, it will be possible to effectively reuse the same grid configuration specified via the `grid-id` prop multiple times. In case of a reusable table you must manage the table state via the `table-context` prop, which must be set to a valid **TableContext**.

For more information, see the **Reusable Table** example in Storybook.

## aw-summary-header

Package: **@swf/pagecomponents**

UI Element to display the selection header of the input ViewModelObject.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| header-vmo | Array.**ViewModelObject** | object | Yes | | The Array of ViewModelObjects for which the summary header is supposed to be shown. The props.object_string property on the first VMO will be used to display the title. |
| header-props | Array.**HeaderPropertiesConfiguration** | object | No | | This is the array of properties which should show in summary header. |


Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| sub-panel-context | object | object | No | | This is context object which will be passed to command bar. This can be used to write conditions for commands. |
| first-anchor | string | string | No | | The first anchor name for which the command bar will be displayed in the summary header. |
| second-anchor | string | string | No | | The second anchor name for which the command bar will be displayed in the summary header. |

## aw-step-progress

Package: **@swf/elements**

A component used to represent a step-progress component.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| current-step | StepConfiguration | object | Yes | | Represents the current active state. |
| completed-step | StepConfiguration | object | Yes | | Represents the completed step i.e user has navigated through that step. |
| steps | Array.StepConfiguration | object | Yes | | Represents the list of steps. |
| clickable | boolean | object | No | `false` | If set to `true`, then user can naviagte through clicking on any completed step. |
| step-click-action | string | action | No | | If clickable is set to `true`, it specifies the action to execute on click of any current or completed step. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## aw-tab-bar

Package: **@swf/composites**

An element representing a tab bar.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| anchor | string | string | Yes | | It is used to identify the placement of tab bar, to be referenced in tabsViewModel.json files. |
| tab-context | object | object | No | | It allows user to share tab specific information between view models. |
| tab-selection-data | object | object | No | | It contains the selected tab information. To access the selected tab information, you can provide an atomic data object with a tabId field. |
| type | string | string | No | | Pass 'unmanage-content' to enable independent content management for the selected tab. |
| show-single-tab-as-header | string | string | No | | Pass true to remove the selected tab styling and display it as header title when there is a single tab. |
| max-tab-width | string | string | No | | Represents the maximum width of a tab title. Only pixel value is supported. |

### Configuring tabs

This component represents a container for **tabs**, but note that it does not contain any child component representing tabs. Instead, you can contribute tabs to a particular tab bar (identified via its `anchor` property) using **tabsViewModel.json** files.

In Siemens Web Framework applications, `tabsViewModel.json` files can be placed in any module, and they are peers of `module.json` files. At build time, the tab definitions present in all `tabsViewModel.json` in all different modules are merged together in a single file, so it is important to make sure that all identifiers used when defining tabs are unique.

## aw-tab-set

Package: **@swf/composites**

An element representing a tab set.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| tabs | Array.TabConfiguration | object | Yes | | specifies the array of objects with tab information. |
| max-tab-width | string | string | No | | Represents the maximum width of a tab title. Only pixel value is supported. |
| tab-set-id | string | string | Yes | | Specifies the ID this tab set will use to register with to support navigation. |
| action | string | action | No | | Represents an action to execute on click. |
| sub-panel-context | object | object | No | | It allows user to share context specific information between view models. |
| close-action | string | action | No | | Specifies which close action to be performed, if the tab is closeable. |

### Events

#### awTab.setSelected

This event is subscribed by the component and can be used to programmatically select a tab by specifying a TabConfiguration object identifying the tab to select.

#### awTab.selected

This event is published by the component whenever a tab is selected. The `eventData` object is set to a TabConfiguration object identifying the selected tab.

## aw-textarea

Package: **@swf/elements**


Siemens Web Framework
Unpublished work. © 2025 Siemens

This element represents a multi-line plain-text editing control, useful when you want to allow users to enter a sizeable amount of free-form text, for example a comment on a review or feedback form.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | No | | The ViewModelProperty used to configure the widget. |

### Configuration tips

- You can set `prop.type` to `STRINGARRAY` and `prop.isArray` to `true` to use this control to edit an array of textual values.
- You can set `prop.numberOfLines` to set the initial height of the textarea input control.

## aw-textbox

Package: **@swf/elements**

A text box is a graphical control element intended to enable the user to input text information. Text boxes are often used on pages that require input from a user.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of the action to execute when the ENTER key is pressed. |
| blur-action | string | action | No | | The name of the action to execute when the textbox loses focus. |
| prop | ViewModelProperty | field | No | | The ViewModelProperty used to configure the widget. |

### Configuration tips

- You can specify the `prop.requiredText` property of the prop attribute to show a placeholder.
- You can set `prop.type` to `STRINGARRAY` and `prop.isArray` to `true` to use this control to edit an array of textual values.


Unpublished work. © 2025 Siemens

- If we want `aw-textbox` to grab the focus when the page loads, we should use the viewModelProperty's autofocus property, e.g. `data.prop.autofocus:'true'`

## aw-tile

Package: **@swf/composites**

A component used to represent a tile.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| tile | TileConfiguration | object | Yes | | An object containing the data that will be represented by the notification cell. |

### Events

#### gateway.unpinTile

This event is published whenever a tile is being unpinned.

## aw-tile-canvas

Package: **@swf/composites**

A component used to represent a tile canvas.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| tile-groups | TileGroupConfiguration | object | Yes | | Represents an array of tile groups. |

### Events


Siemens Web Framework
Unpublished work. © 2025 Siemens

### gateway.editing

This event is published when the tile canvas goes into editing mode. The `eventData` object is set to the TileConfiguration object corresponding to the tile that is being edited.

### gateway.tileClick

This event is published when a tile is clicked. The `eventData` object is set to the TileConfiguration object corresponding to the tile that is being clicked.

### gateway.updateTile

This event is published when the user finished editing the tile canvas. The `eventData` object is set to an object containing the following properties:

- tiles: an array of TileConfiguration objects corresponding to the modified tiles.
- groupNames: An array listing the groups of each modified tile (the index identifies which tile the group refers to).

## aw-tile-group

Package: **@swf/composites**

A component used to represent a tile group.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| tile-group | TileGroupConfiguration | object | Yes | | Represents an array of tile groups. |

## aw-time

Package: **@swf/elements**

This element allows you to enter a time either through text input, or by choosing from a list of predefined times in a dropdown list.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the time. |

**Note**

This component allows you to browse a specific time from a list of predefined times with a precision of 30 minutes (i.e. 00:00, 00:30, 01:30, ...). Alternatively, you can specify your own time by typing in the widget following the HH:MM:SS format.

## aw-togglebutton

Package: **@swf/elements**

This element represent an on-off toggle button. A toggle button take up less space than an ON/OFF radio button group and communicate their intended purpose better.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| action | string | action | No | | The name of the action to execute on toggle. |
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the toggle button. |

## aw-toolbar

Package: **@swf/essentials**

A container element holding one or two command bars.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| first-anchor | string | string | No | | The anchor of the first (left or top on `orientation`) command bar included in the toolbar. |
| second-anchor | string | string | No | | The anchor of the second (right or bottom, depending on `orientation`) command bar included in the toolbar. |
| orientation | BarOrientationEnum | string | No | `"HORIZONTAL"` | The orientation of the toolbar. |
| reverse | boolean | object | No | `false` | Reverses the order of the command of the first command bar. |
| reverse-second | boolean | reverse | No | `false` | Reverses the order of the command of the second command bar. |
| show-command-labels | boolean | object | No | | If set to `true`, labels are displayed for all commands. |
| context | object | object | No | | Additional arbitrary data to include in the commandsViewModel.json to be evaluated later, when commands are executed, as *commandContext*. |
| overflow | boolean | string | No | `true` | When the toolbar does not have enough space to accomodate all visible commands, this attribute is used to control whether commands overflow in overflow popup (`true`), or wrap in multiple lines (`false`). If both command bars exist in a horizontal tool bar, the left command bar will start dropping commands from the right, |


Unpublished work. © 2025 Siemens

| | | | | | |
|---|---|---|---|---|---|
| | | | | | while the right command bar will start dropping commands from the left. For vertical toolbar it works similarly to overflow from the middle. |
| label-position | string | string | No | | This attribute will be used to control the command label position. |
| command-icon-size | string | string | No | | Command icon size for all commands in the toolbar. |

## aw-tree

Package: **@swf/elements**

This component is used to show information in hierarchial way.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| tree | TreeData | object | Yes | | Provides data to `aw-tree`. |
| name | string | string | No | | A unique identifier for the tree. It is used in the `treeNodeSelected` event to identify which tree element has been selected. |
| anchor | string | string | No | | The identifier of the command anchor to use for the tree cell commands. |
| cellCommandContext | object | object | No | | Provides data for `aw-tree` cell commands. |
| expand-action | object | object | No | | An action which is to be executed on expansion of a node. |

### Events


Siemens Web Framework
Unpublished work. © 2025 Siemens

### *treeName*.treeNodeSelected

This event is published when a node is selected. The `eventData` object is set to a TreeData object corresponding to the selected node.

### awtree.updateSelection

This event is subscribed to by the `aw-tree` component and can be used to select and expand a node in the tree by specifying an object as `eventData` containing the following props:

- **name** The name of the tree.
- **selectionValue** the value or label used to select a node within the tree.
- **match** The matching algorithm to use to find the selected node. It can be set to either `exact` (the `selectionValue` must match the node label exactly) or `contains` (the `selectionValue` must be contained within the node label).

## aw-warning-label

Package: **@swf/elements**

An element representing a warning label.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| text | string | string | Yes | | Represents the text to be displayed as a warning. |

## aw-widget

Package: **@swf/elements**

This element is used to represent all widget types in one place.

### Props

| Name | Type | Binding Type | Required | Default | Description |
|---|---|---|---|---|---|
| prop | ViewModelProperty | field | Yes | | The ViewModelProperty used to configure the widget. |

| | | | | | |
|---|---|---|---|---|---|
| hint | string | string | No | | Specifies the input widget which can be used to display or edit the property. |
| options | AwWidgetOptionsConfiguration | object | No | | The options object used to provide additional configuration the widget. |

# Attributes

This section contains the reference documentation of all the public Siemens Web Framework attributes.

**Tip**

For examples on how to use attributes refer to the Siemens Web Framework Storybook.

## aw-class

An attribute used to apply specific CSS classes conditionally.

### Value

An array of object, each of which is a key/value pair with:

- *key* The CSS class to apply.
- *value* The reference to a condition that must be satisfied for the class to be applied.

### Example

```
<aw-button
    action="testButtonAction"
    aw-class="[{ 'accent-positive': conditions.isCheckboxChecked },
{ 'accent-caution': !conditions.isCheckboxChecked }]">
    Click Button
</aw-button>
```

## aw-click

An attribute to execute an action when a component is clicked.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Value

The reference to an action to execute on click.

### Example

```
<aw-icon icon-id="cmdBack" aw-click="actions.iconAction"></aw-icon>
```

## aw-enter

Executes an action when the ENTER key is pressed within the component.

### Value

A reference to the action to execute.

### Example

```
<aw-textbox prop="data.box1" aw-enter="actions.displayValue"></aw-textbox>
```

## aw-repeat

An attribute to render a component multiple times based on the specified array.

### Value

A string in the form *variableName*: *arrayOfItems*, where:

- *variableName* is the variable that will can be used within the repeated component, and that will be populated with an item of the specified array.
- *arrayOfItems* is a reference to an array of items to iterate upon.

### Example

```
<ul>
    <li aw-repeat="listItem: data.listItems">{{listItem}}</li>
</ul>
```


Unpublished work. © 2025 Siemens

## enable-when

An attribute to enable a component based on a condition.

### Value

A reference to a condition used to enable the component.

### Example

```
 <aw-button action="testButtonAction" enable-
when="conditions.isCheckboxChecked">Click Button</aw-button>
```

## exist-when

An attribute to render a component only if the specified condition is satisfied.

> **Tip**
> This attribute will add the component to the DOM only if the condition is satisfied. This is normally preferred for performance reasons, however, you can use use **show-when** to keep the component in the DOM at all times and toggle its visibility instead.

### Value

A reference to the condition used to determine if the component must be rendered or not.

### Example

```
<aw-label exist-when="conditions.isCheckboxChecked" prop="data.label"></aw-
label>
```

## extended-tooltip

An attribute that can be used to display a tooltip containing arbitrary content. The tooltip is displayed when hovering over the element where this attribute is used.

**Description:** extended-tooltip is a attribute element that can be used to add extended tootltip to any UI or Non UI element(commands).


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Attribute Name | Description | Is Mandatory |
|---|---|---|
| extended-tooltip | Provide the configuration for extended-tooltip in UI elements, that is; view, extendedTooltipContent. | No |
| extended-tooltip-context | This attribute is used to pass the reference of object which will help to update the extended tooltip view dynamically. samples: extended-tooltip-context="data" extended-tooltip-context="data.tooltips.tooltip1" | No |
| extendedTooltip | Provide the configuration for extended-tooltip in commands | No |
| extended-tooltip-options | This attribute is used to configure placement and flipBehavior for extended Tooltip, eg: {'placement': 'top', 'flipBehavior':'opposite'}. Options for positioning the Extended Tooltip relative to UI element are - \['top-start','top','top-end','right-start','right','right-end','bottom-end','bottom','bottom-start','left-end','left','left-start'\], Options for flipBehavior are \['fixed', 'opposite', 'clockwise', 'counterclockwise'\] | No |

**Event:** No event is listened or emitted.

**Special View-Model Parameters:** the extended-tooltip object has some optional parameters. See examples in the view-model.

| Parameter | Value | Description | Is Mandatory |
|---|---|---|---|
| view | any view name | Provide view name for "extended-tooltip" in UI elements and "extendedTooltip" key in command definition | No |
| extendedTooltipContent | any string | Only valid when the user wants to show single/ multi line texts or UI value using ViewModel property without **view** | No |

The value of this attribute is set to an object that can contain one of the following properties (they are mutually exclusive):

- `view`: The name of a view/view model to display inside the tooltip.
- `extendedTooltipContent`: A string to display inside the tooltip.

Additionally, the following attributes can be added to the same element where the the `extended-tooltip` attribute is used:

- `extended-tooltip-options`: An **ExtendedTooltipOptions** object used to configure the placement of the tooltip. If this attribute is not used, the tooltip will be displayed at the bottom.


Unpublished work. © 2025 Siemens

- `extended-tooltip-context`: An object that can contain any property that will be accessible within the tooltip view through `subPanelContext`.

### Note :

Extended tooltip has default max height and max width which is 360px. If the content goes beyond max limit, you can add link where you can specify rest of the content. For more details, you can visit checkout command example in commands section.
**Configuring orientation is not applicable for any commands.**

## Configuring and Consuming extended-tooltip

Extended-tooltip, as the name suggests, provides a rich content and information on simple UI elemnts, commands and cell of list.

## Configuring tooltip with only text

If an, extended tooltip with only text has to be shown on the hover of aw-link, then following changes are required.You can configure it by 2 ways.

**1.** In view, the user has to provide the value for **extended-tooltip** attribute that takes any object(For eg: tooltipWithOutView) which is present in data section of the view model.

```
    <aw-link prop="data.box1" extended-
tooltip="data.tooltipWithOutView"></aw-link>
```

In the view model, the tooltip object, that is, tooltipWithOutView should have **extendedTooltipContent** as a key which contains the text/evaluated text which will be shown in the tooltip.

```
    "data": {
            "tooltipWithOutView": {
                "extendedTooltipContent": "{{data.box2.uiValue}}"
            },
        }
```

**2.** If you do not have viewModel for your view file. You can pass object which includes extended tooltip content. tooltipContent = { extendedTooltipContent: 'Tooltip custom content' }

```
                <aw-link-with-popup prop="prop" use-icon="useIcon"
id={{id}} extended-tooltip="{{tooltipContent}}"></aw-link-with-popup prop>
```

## Configuring tooltip with rich content on Simple UI elements

If an, extended tooltip with rich content has to be shown on the hover of aw-link, then following changes are required.You can configure it by 2 ways.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**1.** The view of the element remains same as the view of the "configuring tool-tip with only text". However,in the view model the user has to configure **view** of the extended- tooltip. The view could be any view provided by the user. In this case "extendedTooltip" is the view which contains rich content for extended tooltip. The view model looks like:

```
    <aw-link prop="data.box1" extended-tooltip="data.tooltipWithOutView"></
aw-link>


        "data": {
            "tooltipWithView": {
                "view": "extendedTooltip"
            }
        }
```

**2.** If you do not have viewModel for your view file. You can directly pass view name(extendedTooltip) in UI element instead of fetching it from viewmodel like #1.

```
        <aw-link prop="data.box1" extended-tooltip="extendedTooltip"></aw-
link>
```

## Configuring tooltip on cell of aw-list

If an, extended tooltip with rich content has to be shown on the hover of cell in aw-list, then following changes are required. The user should add **extended-tooltip-context** and **extended-tooltip** attributes on the usage of custom cell element. The value of extended-tooltip-context will be the name of the object which will be present in the scope and the value of extended-tooltip is the any object(For eg: listTooltip) which is present in data section of the view model. The view will look like :

```
    <aw-list dataprovider="data.dataProviders.performSearch">
            <aw-omdb-cell vmo="item" extended-tooltip-context="item"
                extended-tooltip="data.listTooltip">
            </aw-omdb-cell>
        </aw-list>
```

Object which you are passing in extended-tooltip-context, that can be refer by subPanelContext like below :

```
    "data": {
            "poster": {
                "dbValue": "{{subPanelContext.Poster}}"
            }
        }
```


Unpublished work. © 2025 Siemens

## Configuring tooltip on commands

By default the tooltip shows the configured *title* of the command. If a *description* is also provided then it is shown along with the *title*\> as can be seen in examples. Now the command configuration is improved to show *title* and *description* in a tooltip.

```
    "commands":{
            "mailCommandExt": {
                "iconId": "cmdMail",
                "isGroup": false,
                "title": "Mail Command",
                "description": "Used to send a mail"
            }
        }
```

If, however, an extended tooltip with rich content needs to be shown on hover over a command, the following changes are required:

1. The user should add the **extendedTooltip** property in the command definition and configure the view inside it.
2. The value of the view can be any name; in this example, we are using **extendedCommand**, which represents the view and view model of the extended tooltip.
3. Additionally, extended tooltip options can be configured by passing an **options** object inside it.

The command's view model will look like this:

```
    "commands":{
            "mailCommandExt": {
                "iconId": "cmdMail",
                "isGroup": false,
                "title": "Extnded Command with priority 100",
                "extendedTooltip": {
                    "view": "extendedCommand",
                    "options": {
                        "placement": "top"
                    }
                }
            }
        }
```

## Configuring tooltip on cell of aw-splm-table

If an, extended tooltip with rich content has to be shown on the hover of cell in aw-splm-table, then following changes are required. The user should add **extended-tooltip-context** and **extended-tooltip** attributes on the usage of custom cell element. The value of extended-tooltip-context will be the name of the object (For eg: tooltipDetails) which will be present in the scope and the value of extended-tooltip is


Siemens Web Framework
Unpublished work. © 2025 Siemens

any object having view as a key which is present in data section of the view model. The JS code will look like :

```
    var cell ='<img extended-tooltip-context="tooltipDetails" extended-
tooltip= "' + dataDetails + '" />' ;
```

Object which you are passing in extended-tooltip-context, that can be refer by subPanelContext like below :

```
    "data": {
            "textField": {
                "displayName": "Country Name",
                "type": "STRING",
                "dispValue": "{{subPanelContext.countryName}}"
            }
        }
```

The user has to inject ngUtils dependency in JS file and convert HTML string to angularJS element by calling **element()** and pass HTML content, that is, cell as a parameter into it. After that it should compile an angularJS template to HTML DOMElement by calling compile()and pass parent element, angularJS element and scope data(any object to be set on the primary '$scope'), that are, tableElem, currentElement and tooltipObject respectively as parameters in compile function.

```
    var currentElement = ngUtils.element( cell );
        var compiledResult = ngUtils.compile( tableElem, currentElement,
null, null, tooltipObject );
```

### Configuring placement and flipBehavior on extended tooltip

To configure placement and flipBehavior on extended tooltip, following changes needs to add in view of the element.

extended-tooltip-options="{'placement': 'top', 'flipBehavior':'opposite'}" or extended-tooltip-options="{'placement': \['top', 'right', 'left', 'bottom'\] }"

```
<aw-button
    extended-tooltip="{extendedTooltipContent: 'Test Content...'}"
    extended-tooltip-options="{placement: 'top', flipBehavior:
'counterclockwise'}"
>Test Button</aw-button>
```

## show-when

An attribute to display a component only if the specified condition is satisfied.

**Tip**

This attribute will keep the component in the DOM at all times, and just toggle its visibility. If you want to add a component to the DOM only if a condition is satisfied (preferred for performance reasons), use **visible-when** or **exist-when** instead.

### Value

A reference to the condition used to determine if the component must be rendered or not.

### Example

```
<aw-label show-when="conditions.isCheckboxChecked" prop="data.label"></aw-
label>
```

## visible-when

An attribute to render a component only if the specified condition is satisfied.

**Tip**

This attribute will add the component to the DOM only if the condition is satisfied. This is normally preferred for performance reasons, however, you can use use **show-when** to keep the component in the DOM at all times and toggle its visibility instead.

### Value

A reference to the condition used to determine if the component must be rendered or not.

### Example

```
<aw-label visible-when="conditions.isCheckboxChecked"
prop="data.label"></aw-label>
```

# Types

This section contains the reference documentation of all the public custom object types and enumerations defined in Siemens Web Framework.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## ActiveCommandConfig

A configuration object identifying the currently active command.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| commandId | string | | The commandId of the currently active command. |

## AdvancedBarOrientationEnum

Represents the orientation of advanced bar.

### Values

- `"HORIZONTAL"` Represents horizontal orientation of bar.
- `"VERTICAL"` Represents vertical orientation of bar.

## AdvancedBarTypeEnum

Represents the type of advanced bar.

### Values

- `"TASKBAR"` This is dark blue colored bar with fixed height and can only aligned to HORIZONTAL.
- `"TOOLBAR"` This is grey colored normal toolbar with fixed height if HORIZONTAL and fixed width if VERTICAL.
- `"FOOTER"` This is dark blue colored bar with fixed different height and can only be aligned HORIZONTAL.

## AlignContentEnum

It uses flex align content options to align content in the perpendicular direction within the row/column.

### Values

- `"start"` Start
- `"center"` Center
- `"end"` End

- `"stretch"` Stretch

## AlignmentTypeEnum

Specifies the alignment of header context.

### Values

- `"HORIZONTAL"`

## ArrowOptions

Specifies the **arrow options**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| alignment | **PopupAlignmentEnum** | `'auto'` | Specifies where to align the arrow relative to reference element. |
| offset | string | | Specifies the arrow offset relative to reference element's start side, unit in px, it can be used for start / end alignment case. |
| shift | string | | Specifies the arrow offset relative to popup element's start side, unit in px. |

## AvatarSizeEnum

Defines the allowed aw-avatar sizes.

### Values

- `"xsmall"` Extra Small
- `"small"` Small
- `"medium"` Medium
- `"large"` Large
- `"xlarge"` Extra large


Siemens Web Framework
Unpublished work. © 2025 Siemens

## AwWidgetOptionsConfiguration

An object used to configure the widget options.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| showBulkSelection | boolean | `false` | If set to `true`, it enables select-all/unselect-all button in the checkbox lov widget. |
| noValuesMessage | string | | Custom Message to display when there are no values in the widget. |

## awHttpInstance

Object providing access to various http apis.

### Methods

#### get(url: *string*, config: *AxiosConfigObject*) → *Promise.HttpPromise*

Method to perform GET request.

### Parameters

| Name | Type | Description |
|---|---|---|
| url | string | Absolute or relative URL of the resource that is being requested. |
| config | AxiosConfigObject | Optional configuration object. |

### Returns

Promise.<HttpPromise> A Promise that will be resolved or rejected with a response object.

#### delete(url: *string*, config: *AxiosConfigObject*) → *Promise.HttpPromise*

Method to perform DELETE request.

### Parameters

| Name | Type | Description |
|---|---|---|
| url | string | Absolute or relative URL of the resource that is being requested. |
| config | AxiosConfigObject | Optional configuration object. |

### Returns

Promise.<HttpPromise> A Promise that will be resolved or rejected with a response object.

## put(url: *string*, data: *object*, config: *AxiosConfigObject*) → *Promise.HttpPromise*

Method to perform PUT request.

### Parameters

| Name | Type | Description |
|---|---|---|
| url | string | Relative or absolute URL specifying the destination of the request. |
| data | object | Request content. |
| config | AxiosConfigObject | Optional configuration object. |

### Returns

Promise.<HttpPromise> A Promise that will be resolved or rejected with a response object.

## patch(url: *string*, data: *object*, config: *AxiosConfigObject*) → *Promise.HttpPromise*

Method to perform PATCH request.

### Parameters

| Name | Type | Description |
|---|---|---|
| url | string | Relative or absolute URL specifying the destination of the request. |
| data | object | Request content. |
| config | AxiosConfigObject | Optional configuration object. |

### Returns

Promise.<HttpPromise> A Promise that will be resolved or rejected with a response object.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### post(url: *string*, data: *object*, config: *AxiosConfigObject*) → *Promise.HttpPromise*

Method to perform POST request.

#### Parameters

| Name | Type | Description |
|---|---|---|
| url | string | Relative or absolute URL specifying the destination of the request. |
| data | object | Request content. |
| config | AxiosConfigObject | Optional configuration object. |

#### Returns

Promise.<HttpPromise> A Promise that will be resolved or rejected with a response object.

### head(url: *string*, config: *AxiosConfigObject*) → *Promise.HttpPromise*

Method to perform HEAD request.

#### Parameters

| Name | Type | Description |
|---|---|---|
| url | string | Absolute or relative URL of the resource that is being requested. |
| config | AxiosConfigObject | Optional configuration object. |

#### Returns

Promise.<HttpPromise> A Promise that will be resolved or rejected with a response object.

## awParseServiceInstance

Function that accepts the expression to be parsed and returns a function to evaluate the expression. Invoke the returned function with the context to evaluate the expression to obtain the parsed value.

```
const expr = 'data.greeting';const context = {     data: {
greeting: 'Hello'     }}const parsedValue = AwParseService.instance( expr )
( context )parsedValue === 'Hello'
```

## AwStateInstance

Object providing access to the state routing apis.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| go | StateGo | | Method for transitioning to a new state. |
| params | StateParams | | The latest successful state parameters. |
| current | StateCurrent | | The current State. |
| href | StateHref | | Generates a URL for a state and parameters. Returns the url for the given state populated with the given params. |
| reload | StateReload | | A method that force reloads the current state, or a partial state hierarchy. |
| get | StateGet | | Returns the state declaration object for any specific state, or for all registered states. |
| transitionHooks | TransistionHooks | | This provides apis related to Transitions. |

## BannerContext

An object representing the configuration for the global application banner (see js/bannerService).

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| identifier | string | | A unique identifier for the banner. |
| text | string | | The text content of the banner. |
| startDate | number | | The start date of the banner, represented as a timestamp. |
| endDate | number | | The end date of the banner, represented as a timestamp. |
| duration | number | | The duration of the banner in hours. |
| type | string | `"INFO"` | The type of the banner. Must be one of 'SUCCESS', 'WARNING', or 'INFO'. |
| closable | boolean | `true` | Whether the banner can be closed by the user. |


Unpublished work. © 2025 Siemens

## BarOrientationEnum

Defines the orientation of an **aw-command-bar** or **aw-toolbar**.

### Values

- `"HORIZONTAL"` Horizontal
- `"VERTICAL"` Vertical

## ButtonSizeEnum

Defines the possible sizes for **aw-button**.

### Values

- `"small"` Small
- `"medium"` Medium
- `"large"` Large

## ButtonTypeEnum

Defines the possible **aw-button** types.

### Values

- `"base"` Base
- `"sole"` Sole
- `"positive"` Positive
- `"accent-positive"` Positive accent
- `"negative"` Negative
- `"accent-negative"` Negative accent
- `"caution"` Caution
- `"accent-caution"` Caution action
- `"accent-high-contrast"` High-contrast accent
- `"accent-mid-contrast"` Mid-contrast accent
- `"accent-marketing"` Marketing accent

- `"chromeless"` No border or background.

## ButtonWidthEnum

Defines the possible types of widths for **aw-button**.

### Values

- `"stretched"` Stretch the button taking all available width.
- `"auto"` Determine the width based on the button text.

## CellProperty

An object representing a property to display in a list cell.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| key | string | | The name of the property. |
| value | any | | The value of the property. |

## ChartPointsConfig

An chart point object used to configure an **aw-chart** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| seriesType | **SeriesTypeEnum** | | Represents the type of series for charts. |
| seriesColorClass | string | | Specifies class which has custom colors , can also be us |
| seriesMarker | **SeriesMarker** | | Represents the options for the point markers of series w |
| seriesTooltip | **SeriesTooltip** | | Represents the options for the tooltip of series which d |
| isDataLabelOnChartEnabled | boolean | | If set to `true`, the labels will be shown for chart. |
| showInLegend | boolean | | If set to `true`, the series name will be shown in the leg |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| yAxis | string | | Represents the number defines which yAxis the particu |
| dashStyle | string | | Represents the line style of the series for line charts. All ['Solid','ShortDash','ShortDot''ShortDashDot','ShortDash |

## ChartProvidersConfiguration

An object used to configure an **aw-chart** or **aw-echart** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| title | string | | Specifies the title of a chart. |
| chartType | ChartTypeEnum | | Specifies the types of charts supported. |
| chartColorOverrideClass | string | | Specifies the class which has custom colors. |
| loadDataAction | string | | Represents an action which holds chartPoints that has an array of `label` (string) and `value` (number) properties. |
| chartConfig | ChartsConfigObject | | The chart configuration settings. |
| renderMethod | string | | (Optional) Rendering method used by echart. Valid values are 'canvas' or 'svg'. Defaults to svg. Canvas mode can be used to improve performance for large datasets. |

## ChartsConfigObject

An object used to configure an **chart configuration** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| isChartZoomable | boolean | | If set to `true`, chart is zommed in both X and Y axis. |
| isYAxisLinearOrLogarithmic | ChartYaxisValuesEnum | `"linear"` | Represents the numbers along the Y-axis. |

| | | | |
|---|---|---|---|
| isDataLabelOnChartEnabled | boolean | `true` | If set to `true`, the stack labels is shown for the total value in each bar in a stacked column chart. |
| xAxisLabel | string | | Represents X-axis title, shown next to the axis line. |
| yAxisLabel | string | | Represents Y-axis title, shown next to the axis line. |
| size | string | | Represents the diameter of the pie chart. |

## ChartTypeEnum

Specifies the type of charts supported for **chart-provider**.

### Values

- `"column"`
- `"pie"`
- `"line"`
- `"scatter"`

## ChartYaxisValuesEnum

Specifies the numbers along the y-axis.

### Values

- `"linear"`
- `"logarithmic"`

## CheckListConfiguration

An object used to configure an **aw-check-list** component.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| displayName | string | | Represents the displyable name of check list. |
| type | string | `"BOOLEAN"` | It must be a boolean value. |
| isRequired | string | `true` | The property isRequired should be set to true. |
| isArray | string | `true` | The value of the property should be an array. |
| dbValue | Array.CheckListValue | | Represents the value of a checklist item. |

## CheckListValue

Represents the value of a checklist item, part of the **checklist configuration**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| iconName | string | | Represents the icon to be shown before check-list. |
| isChecked | string | `false` | If set to `true`, list item will be shown as checked in the initial state. |
| checkAction | string | | Specifies an action to execute on click of list item. |
| buttonAction | string | | Specifies an action to execute on click of icon. |
| tooltip | string | | Represents a tooltip to be shown on hover of icon. |

## ChipConfiguration

An object used to configure an **aw-chip** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| chipType | ChipTypeEnum | | Represents type of the chip. |
| labelDisplayName | string | | The displayable name of the chip. |
| labelInternalName | string | | Represents internal value of the chip. |
| iconId | string | | Represents icon of the chip. |

| | | | |
|---|---|---|---|
| uiIconId | string | | Represents UI icon of the chip (appears on the right side of the label). |
| showIcon | boolean | `true` | If set to `false`, the chip icon will not be visible. |
| showLabel | boolean | `true` | If set to `false`, the chip label will not be visible. |
| selected | boolean | | Represents initial selection state for selectable chip (if the chip is of type `SELECTION`). |
| enableWhen | boolean | | If evaluated expression is `false`, the chip will be disabled. |
| buttonType | **ButtonTypeEnum** | | Represents style of the chip. |

## ChipTypeEnum

Defines the possible types of **aw-chip**.

### Values

- `"STATIC"` Display a plain chip with no action associated and used to show status, like a label.
- `"BUTTON"` Display an actionable chip that is clickable and can trigger action, like a button.
- `"SELECTION"` Display a selectable chip that has a selected state, like a checkbox.

## Column

An object representing the configuration of a **aw-splm-table** column, defined within a **ColumnProviderConfiguration**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| displayName | string | | The name of the column that is displayed in the table header. If not set, a display name is generated based on `name`. |
| dataType | string | | The base type this property is represented by. (String, Double, Integer, Date, etc.). Commonly used for filtering |
| enableColumnHiding | boolean | `true` | If set to `true`, it is possible to hide the column via the column menu (if enabled). |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| enableColumnMenu | boolean | `true` | If set to `true`, a menu to perform operations on the column is enabled when clicking the column header. This property must be set to `true` to enable column freezing and hiding. |
| enableColumnMoving | boolean | `false` | if set to `true`, it is possible to rearrange the column by dragging and dropping its header. |
| enableColumnResizing | boolean | `true` | If set to `true`, the column can be resized via its header. |
| enableSorting | boolean | `true` | If set to `true`, sort widgets are added to the column header, allowing sorting of column data. |
| headerTooltip | boolean | `true` | If set to `true`, a tooltip is displayed when hovering the header cell for this column. |
| isFilteringEnabled | boolean | `true` | If set to `true`, filtering is enabled for this column (this is the default when the `isFilteringEnabled` property of the **GridOptions** object is set to `true`). |
| isTableCommand | boolean | `false` | If set to `true`, commands contributed to the command bar configured for the table (via the `commandsAnchor` **DataProviderConfiguration** property) will be displayed in the cells within this column. Only one column can have this property set to true, and is not applicable for multiple columns. |
| isTreeNavigation | boolean | `false` | If set to `true`, the column will be used for tree navigation (expanding and collapsing nodes). Also if this is set to true for a column, then isTableCommand must be set to false, and vice versa. |
| maxWidth | number | | The maximum width of the column in pixels (by default, 125% of `width`). |
| minWidth | number | `150` | The minimum width of the column in pixels. |
| name | string | | The unique identifier of the column (mandatory). |
| pinnedLeft | boolean | `false` | If set to `true`, the column is pinned (frozen) to the left side. |
| propertyName | string | | The internal name of the property. This is mandatory for editing support, and must be unique among all table columns. |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| renderingHint | RenderingHintEnum | | Determines what input widget will be used to edit cells within the column. |
| typeName | string | | The business object type name *(Teamcenter-specific)*. |
| width | number | | The width of the column in pixels (by default, 125% of `minWidth`). |
| modifiable | boolean | `true` | If set to `false`, column will be non editable even if **isPropertyModifiable** is `true` on individual cell. This applies to static columns only. |

## ColumnChartConfiguration

An object used to configure an **aw-column-chart** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| title | string | | Specifies the title of a chart. |
| columns | ColumnValue | | Specifies the value and label of property. |
| chartColorOverrideClass | string | | Specifies the class which has custom colors. |

## ColumnConfiguration

An object representing the configuration of a **aw-splm-table** column, defined within a **ColumnProviderConfiguration**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| displayName | string | | The name of the column that is displayed in the table header. If not set, a display name is generated based on `name`. |
| dataType | string | | The base type this property is represented by. (String, Double, Integer, Date, etc.). Commonly used for filtering |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| enableColumnHiding | boolean | `true` | If set to `true`, it is possible to hide the column via the column menu (if enabled). |
| enableColumnMenu | boolean | `true` | If set to `true`, a menu to perform operations on the column is enabled when clicking the column header. This property must be set to `true` to enable column freezing and hiding. |
| enableColumnMoving | boolean | `false` | if set to `true`, it is possible to rearrange the column by dragging and dropping its header. |
| enableColumnResizing | boolean | `true` | If set to `true`, the column can be resized via its header. |
| enableSorting | boolean | `true` | If set to `true`, sort widgets are added to the column header, allowing sorting of column data. |
| headerTooltip | boolean | `true` | If set to `true`, a tooltip is displayed when hovering the header cell for this column. |
| isFilteringEnabled | boolean | `true` | If set to `true`, filtering is enabled for this column (this is the default when the `isFilteringEnabled` property of the **GridOptions** object is set to `true`). |
| isTableCommand | boolean | `false` | If set to `true`, commands contributed to the command bar configured for the table (via the `commandsAnchor` **DataProviderConfiguration** property) will be displayed in the cells within this column. |
| isTreeNavigation | boolean | `false` | If set to `true`, the column will be used for tree navigation (expanding and collapsing nodes). |
| maxWidth | number | | The maximum width of the column in pixels (by default, 125% of `width`). |
| minWidth | number | `150` | The minimum width of the column in pixels. |
| name | string | | The unique identifier of the column (mandatory). |
| pinnedLeft | boolean | `false` | If set to `true`, the column is pinned (frozen) to the left side. |
| propertyName | string | | The internal name of the property. This is mandatory for editing support, and must be unique among all table columns. |
| renderingHint | **RenderingHintEnum** | | Determines what input widget will be used to edit cells within the column. |

| | | | |
|---|---|---|---|
| typeName | string | | The business object type name *(Teamcenter-specific)*. |
| width | number | | The width of the column in pixels (by default, 125% of `minWidth`). |

## ColumnProviderConfiguration

An object representing the configuration of a column provider used to define column-related settings for **aw-splm-table**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| frozenColumnIndex | number | | An integer value determining how many columns will be frozen (pinned to the left) follows zero-based numbering. If this property is not set, the first column will be frozen, if this property is not set and the first column is an icon column, the first two columns will be frozen. To have no columns frozen, this property must be set to -1 and no column must have the `pinnedLeft` property set to `true`. |
| sortCriteria | Array.**SortCriteriaConfiguration** | [ ] | Defines the initial sort criteria based on which the data needs to be loaded by data provider. The value is an array of Java script objects. Each object contains 2 attributes. "fieldName" and "sortDirection". E.g. sortCriteria: [{fieldName: "object_name", sortDirection: "DESC"}] |
| columns | Array.**Column** | | The configuration of each column of the table. |

## ColumnValue

An object used to configure **column values** component.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| label | string | | Specifies the label of column property. |
| value | string | | Specifies the value of column property. |

## CrumbsConfiguration

Represents an array of object **crumb** used to configure an **aw-navigate-breadcrumb** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| CrumbsObject | Array.**CrumbsObject** | | Represents an array of crumbs object. |

## CrumbsObject

An object used to configure a **crumb**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| clicked | boolean | | xxx. |
| displayName | string | | Represents the displayable name of the crumb. |
| selectedCrumb | boolean | | Represents the selected crumb. |
| showArrow | boolean | | If set to `true`, the chevron is shown at the right side of the crumb. |
| primaryCrumb | boolean | | Represents the first crumb. |

## CurrentLocationData

A configuration object containing data related to the current location.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| titles | LocationTitles | | The current location titles. |

## CurrentStateData

A configuration object containing data related to the current state.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| params | Record.<string, string> | | The current state parameters. |
| urlAttributes | Record.<string, string> | | The query string parameters of the page (passed via query string, not fragment). |

## CurrentSublocationData

A configuration object containing data related to the current sublocation.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| historyNameToken | string | | The name of the state corresponding to the sublocation. |
| label | LocalizedLabel | | The label of the sublocation. |
| nameToken | string | | The name of the sublocation, specified in the states.json configuration file. |

## DataProvider

Defines a data provider object that wraps access to both the 'viewModelCollection' as well as 'selectionModel'.


Unpublished work. © 2025 Siemens

## Properties

| Name | Type | Default | Description |
|---|---|---|---|
| viewModelCollection | ViewModelCollection | | An object that maintains the loaded ViewModelObjects and exposes APIs to access the information of collection |
| selectionModel | SelectionModel | | An object contains a list of selected UIDs and other internal state information such as multi-select state, selection mode, and selection status |

## Methods

### getSelectedObjects() → *Array.ViewModelObject*

Gets all of the loaded objects that are currently selected

### Returns

Array.ViewModelObject The list of selected VMOs.

### update(newVMOs: *Array.ViewModelObject*, totalFound: *number*)

Sets the currently loaded view model objects in the given ViewModelCollection

### Parameters

| Name | Type | Description |
|---|---|---|
| newVMOs | Array.ViewModelObject | Array of ViewModelObjects to set. |
| totalFound | number | Total number of objects found. |

### getViewModelCollection() → *ViewModelCollection*

Gets ViewModelCollection being managed by this dataprovider

### Returns

ViewModelCollection Reference to the ViewModelCollection being managed by this DataProvider.

### selectAll(dataProvider: *DataProvider*)

Selects all loaded objects in the dataprovider

#### Parameters

| Name | Type | Description |
|---|---|---|
| dataProvider | DataProvider | DataProvider to call the selectAll command from. |

### selectNone(dataProvider: *DataProvider*)

Deselects all loaded objects in the dataprovider

#### Parameters

| Name | Type | Description |
|---|---|---|
| dataProvider | DataProvider | DataProvider to call the deSelectAll command from. |

### dataProviderEnableMultiSelect(dataProvider: *DataProvider*)

Enables Multi Select Mode for selected dataprovider

#### Parameters

| Name | Type | Description |
|---|---|---|
| dataProvider | DataProvider | DataProvider to call the selectAll command from. |

### dataProviderDisableMultiSelect(dataProvider: *DataProvider*)

Disables Multi Select Mode for selected dataprovider

#### Parameters

| Name | Type | Description |
|---|---|---|
| dataProvider | DataProvider | DataProvider to call the deSelectAll command from. |

## DataProviderConfiguration

An object used to configure a data provider object in the `dataProviders` section of the view model.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| type | DataProviderTypeEnum | `"action"` | Specifies the type of data provider. |
| name | string | | Specifies the name of the data provider. |
| action | string | | Specifies the action to execute to load data in the data provider. |
| response | Array.any | | Represents the response array. It can be set to either an array of user-defined objects, or an expression pointing to a variable in the `data` section of the view model, typically containing the result of the data provider action. |
| totalFound | string | | Specifies the total objects managed by the data provider (not necessarily loaded on the client). Required for pagination support. |
| selectionModelMode | SelectionModelModeEnum | `"single"` | Specifies what kind of selection is supported by the data provider. |
| enablePropEdit | boolean | `true` | If the data provider is used by an **aw-splm-table** component, if set to `true` it will be possible edit table cells without having to go through the Edit command. Click twice on any editable table cell to edit the cell. The possible values of this option are `true`, `false`, or a declarative condition. A declarative condtion will allow true/false to be set based on the conditions resolved value. If the *Autosave Tables* toggle is enabled in the user settings panel, changes to the table cells will be saved instantly on blur. If the toggle is disabled, the user has to explicitly save the edits using the Save command. |

## DataProviderTypeEnum

Defines the possible data provider types.

### Values

- `"Static"` The data provider accepts a static list of values, specified directly in the `response` property of the **DataProviderConfiguration** object.
- `"Action"` The data provider loads data through a specified action defined in the view model.
- `"TcLOV"` *Teamcenter-specific*

## DeclarativeViewModel

The declarative ViewModel object. Please refer the **schema**

## DialogHeight

Specifies height of aw-dialog.

### Values

- `"LARGE"` 75%
- `"FULL"` 100% *(default)*
- `"STANDARD"` 40%
- `"WIDE"` 50%
- `"DEFAULT"` The same height as the width of the panel

## DialogOptions

An object used to configure an **aw-dialog** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| view | string | | Specifies the view that user wants to display inside aw-dialog. |
| caption | string | | Represents the title shown on the dialog. |
| parent | string | | Represents CSS parent locator. Dialog will reside as child of this locator in DOM. |
| height | **DialogHeight** | | Specifies height of aw-dialog. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| width | DialogWidth | | Specifies width of aw-dialog. |
| placement | DialogPlacement | | Specifies the postion of dialog. |
| push | boolean | | If set to `true`, dialog opens in push mode and sibling contents are pushed. Default is float which overlaps sibling contents. |
| subPanelContext | object | | Represents the context that user wants to pass to dialog. |
| isCloseVisible | boolean | | If set to `false`, command panel button is used and dialog close button is not needed. |
| isPinUnpinEnabled | boolean | | If set to `true`, will have pinning enabled and maintain a state panelPinned(default: false). |
| panelPinned | boolean | | If set to `true` along with isPinUnpinEnabled, dialog will have pinning enabled along with initial state of panelPinned to be true. |

### Detecting the size of a dialog

The `sidenavMode` property automatically set on the `subPanelContext` object that is passed to the dialog component.

`subPanelContext.sidenavMode` can be used to detect the current size of the dialog, based on specific width thresholds see SidenavModeEnum for the possible values of this property. This property can be used to implement responsive dialogs that display information in a different way depending on their available width.

## DialogPlacement

Specifies the position of a dialog.

### Values

- `"right"`
- `"left"`

## DialogWidth

Specifies width of aw-dialog.

### Values

- `"STANDARD"` 360px *(default)*
- `"SMALL"` 280px
- `"WIDE"` 480px
- `"EXTRAWIDE"` 540px
- `"DOUBLEWIDE"` 720px

## DirectionTypeEnum

This attribute will decide the direction of sidenav panel. The direction defines the shadow and rounding of corners only.

### Values

- `"RIGHT_TO_LEFT"` Represents that transition of sidenav should be from right to left.
- `"LEFT_TO_RIGHT"` Represents that transition of sidenav should be from left to right.

## DisplayedViewModes

An object used to represent the currently displayed view modes in **Workarea**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| pwaViews | Array.string | | Represents an array of visible PWA views. |
| swaViews | Array.string | | Represents an array of visible SWA views. |
| altPwaViews | Array.string | | Represents an array of visible Alt PWA views. |

## EditorConfigOptions

An object used to configure an **editor config options** component.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| automaticLayout | boolean | `false` | If set to `true`, it enables that the editor will install an interval to check if its container dom node size has changed. |
| readOnly | boolean | `false` | If set tp `true`, the editor will not be editable. |
| lineNumbers | boolean | `true` | If set to `false`, line numbers will not be shown. |
| wordWrap | boolean | `false` | If set to `true`, words on the editor can be seen in wrapped manner when it exceeds limit. |

## EditorLanguageEnum

Specifies the language written in **editor**.

### Values

- `"javascript"`
- `"json"`
- `"html"`
- `"typescript"`
- `"css"`
- `"XML"`
- `"PHP"`
- `"C#"`
- `"C++"`
- `"Razor"`
- `"Markdown"`
- `"Diff"`
- `"Java"`
- `"VB"`
- `"CoffeeScript"`
- `"Handlebars"`
- `"Batch"`
- `"Pug"`

- `"F#"`
- `"Lua"`
- `"Powershell"`
- `"Python"`
- `"Ruby"`
- `"SASS"`
- `"R"`
- `"Objective-C"`

## EditorThemeEnum

Represents theme of the **editor**.

### Values

- `"vs"`
- `"vs-dark"`
- `"hc-black"`

## EchartProvidersConfiguration

An object used to configure an **aw-echart-advanced** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| name | string | | Specifies a unique name for the provider within this view-model. |
| chartColorOverrideClass | string | | Specifies the class which has custom colors. |
| loadDataAction | string | | Represents an action which holds chartPoints that has an array of `label` (string) and `value` (number) properties. |
| renderMethod | string | | (Optional) Rendering method used by echart. Valid values are 'canvas' or 'svg'. Defaults to svg. |

echartConfig Initial static ECharts configuration settings. | | chartOptions | **EchartConfigObject** | | Dynamic ECharts configuration settings. Combined with echartConfig to render chart. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## EventBusEventCallback

The callback function invoked when an event is published

### Parameters

| Name | Type | Default | Description |
|---|---|---|---|
| eventData | object | | Data contained in the event that was published |

## ExtendedTooltipFlipBehavior

Represents the flip behavior of an **extended tooltip** when there is not enough space to display it based on its placement settings.

### Values

- `"fixed"`
- `"opposite"`
- `"clockwise"`
- `"counterclockwise"`

## ExtendedTooltipOptions

Represents the options to specify via an `extended-tooltip-options` object, used when configuring an **extended-tooltip**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| placement | ExtendedTooltipPlacement | | The placement of the tooltip relative to the UI element with the extended tooltip. |
| flipBehavior | ExtendedTooltipFlipBehavior | | The behavior determining the position of the tooltip when there is no space to display it according to the placement setting. |

## ExtendedTooltipPlacement

Represents the placement of an **extended tooltip**.

### Values

- `"top-start"`
- `"top"`
- `"top-end"`
- `"right-start"`
- `"right"`
- `"right-end"`
- `"bottom-end"`
- `"bottom"`
- `"bottom-start"`
- `"left-end"`
- `"left"`
- `"left-start"`

## FlipBehaviorOptions

Represents the behaviour options for popup.

### Values

- `"fixed"`
- `"opposite"`
- `"clockwise"`
- `"counterclockwise"`

## Grid

Represents the Grid (table) in a view model. Used to configure an **aw-splm-table** component.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| addIconColumn | boolean | `true` | If set to `true`, the first column of the table will be used to display the type icon of the object displayed in the table row. |
| columnProvider | string | | (required) The name of the **ColumnProvider** this grid will use. |
| dataProvider | string | | (required) The name fo the **Dataprovider** this grid will use. |
| propertyProvider | string | | The name of the property provider this grid can use |
| gridOptions | **GridOptions** | | The options used to more specifically configure a grid (table). |

## GridOptions

Represents the options used to configure a grid (table) in a view model. Used to configure an **aw-splm-table** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| enableCellHeaderSelection | boolean | `false` | If set to `true`, changes the way the table will show cell selection, by highlighting the cell selected, the row header (pinned columns) and the column header. This is generally used in table as matrix implementations. |
| enableColumnSelection | boolean | `false` | If set to `true`, adds a **Select Column** command in the column menu that will show the entire column as selected. If the column configuration has a **VMO** attached, then it will also update selection to that object. This is primarily used in table as matrix implementations. |
| enableExpandAndPaginationInEdit | boolean | `false` | If set to `true`, table will be scrolled and paginated while in edit mode, and new data will be put into edit if modifiable. When in tree mode, nodes can be expanded and new data will be loaded and put into edit mode if modifiable. |

| | | | |
|---|---|---|---|
| enableExpansionStateCaching | boolean | `true` | If set to `true`, tree expansion state will be saved to local storage. On reload the table will automatically restore to the previous expansion state. |
| enableDragAndDrop | boolean | `true` | If set to `true`, default or custom drag and drop handlers will be added to the table. |
| enableDynamicRowHeight | boolean | | If set to `true`, the height of each table row will be determined dynamically based on its data. |
| enableGridMenu | boolean | | If set to `true`, enables the table settings menu accessible via the gear icon on the top right corner of the table. |
| enablePinning | boolean | `true` | If set to `true`, it is possible to pin (freeze) table columns. By default, if no `pinnedLeft` property is set in any **Column** object of the table, the first column will be pinned to the left. |
| enableSorting | boolean | `true` | If set to `true`, sorting is enabled for all columns defined for the table. To override this setting in specific columns, use the `enableSorting` property of the **Column** object. |
| headerHeight | **TableRowHeightEnum** | | Used to specify a larger height for the table header. |
| isFilteringEnabled | boolean | `false` | If set to `true`, enables filtering for all columns. To disable on specific columns, set the `isFilteringEnabled` property to `false` in the **Column** object. |
| maxRowsToShow | number | | Sets the maximum height of the table to be equal to the number of rows specified. |
| rowHeight | **TableRowHeightEnum** | | Used to specify a larger height for table rows. |
| showContextMenu | boolean | | If set to `true`, a context menu will be displayed when context selecting a row. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| textBundle | object | | Used to display a custom title to the tree expand/ collapse button. For example: `{"TwistieTooltipCollapsed": "{{i18n.expand}}", "TwistieTooltipExpanded": "{{i18n.collapse}}" }`. |
| enableHeaderResizing | boolean | | If set to `true`, adds a draggable header handler to the table's header allowing the resize of header height. |

## HeaderContextAlignmentTypeEnum

Specifies the alignment of header context.

### Values

- `"HORIZONTAL"`

## HeaderPropertiesConfiguration

An object used to configure an **aw-header-props** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| property | **ViewModelProperty** | | The ViewModelProperty used to configure header properties. |
| renderingHint | **RenderingHintEnum** | | Specifies how the property values should be displayed. |
| renderingStyle | **RenderingStyleEnum** | | Specifies the rendering style of the property. |

## IconButtonConfiguration

Represents the options used to configure an **aw-icon-button** in a view model.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| action | string | | Represents name of action to be executed when button is clicked. |
| iconName | string | | Represents the name of icon to be displayed. |
| tooltip | string | | The tooltip to display on mouse hover. |

## IconOptionsEnum

Defines possible icon options for aw-link.

### Values

- `"textOnly"` Display text for link title or menu cell item.
- `"iconOnly"` Display icon for link title or menu cell item.
- `"both"` Display both text and icon for link title or menu cell item.

## Indicator

An object representing an indicator image to display in table and list cells.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| image | string | | The name of the icon to display (without extension or size suffix). |
| tooltip | string | | The tooltip to display on mouse hover. |
| propName | string | | The property this indicator relates to internal name |
| propDisplayName | string | | The property this indicator relates to display name |

## JustifyContentEnum

Defines to justify content within the row/column.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Values

- `"top"` Top
- `"center"` Center
- `"bottom"` Bottom
- `"space-between"` Space-between
- `"space-around"` Space-around
- `"space-evenly"` Space-evenly

## LabelPositionEnum

Defines the possible placements for property labels.

### Values

- `"PROPERTY_LABEL_AT_TOP"` Display the property label above the input widget.
- `"PROPERTY_LABEL_AT_SIDE"` Display the property label on the left of the input widget.
- `"PROPERTY_LABEL_AT_RIGHT"` Display the property label on the right of the input widget.
- `"NO_PROPERTY_LABEL"` Do not display the property label.

## Layout

An object used to represent the current layout in **Workarea**. Examples and further details about the layout can be found in **ViewModes and Layouts**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| row | array.<**Layout**.<**ViewTuple**>> | | Represents a horizontal section of the layout where the given view tuples will be rendered horizontally. If you need to create vertical sections within it, you can do so using the 'column' attribute. |
| column | array.<**Layout**.<**ViewTuple**>> | | Represents a vertical section of the layout where the given view tuples will be rendered vertically. If you need to create horizontal sections within it, you can do so using the 'row' attribute." |

## LayoutContext

An object used to represent the current layout context in **Workarea**. Examples and further details about the layout context can be found in **ViewModes and Layouts**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| layout | object.<**Layout**> | | The layout structure which is built using row and column where row represents horizontal section and column represents vertical section of the layout. |
| layoutId | string | | The unique key to indetify the layout. |

## LayoutGridSize

A value used to identify a size of a the width, height, or offset of a layout **row** or **column**.

This size can be expressed:

- As an integer from 1 to 12, identifying the proportion of the corresponding size of the parent element (example: 2).
- As a string in the form _X_f, where *X* is an integer number indicating a fixed size in em units (example: `2f`).

## LinkIconOptionsConfiguration

Defines possible icon options for **aw-link**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| title | **IconOptionsEnum** | `"textOnly"` | Specifies the properties of link with title. |
| cell | **IconOptionsEnum** | `"textOnly"` | Specifies the properties of link with menu item. |
| arrow | boolean | `true` | If set to `false`, dropdown icon won't show. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## LinkOptionsConfiguration

An object used to configure the link options.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| hasPopup | boolean | `false` | If set to `true`, it represents that link has a popup. |
| popupOptions | any | `"null"` | Specify the popup options. It is only valid when hasPopup is true. |
| view | string | `"AwLinkPopupMenu"` | When Popup shows up, it specifies the embed view to load. |
| iconOptions | LinkIconOptionsConfiguration | | Specifies the controls to show icon or text for link title and menu items. |

## LinkPopupResponsePropertyValue

An object representing a popup menu defined via an static display value and a list value.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| staticDisplayValue | string | | Element for displaying a popup menu on the click of the link. |
| staticElementObject | string | | The displayable name of the popup menu |
| selected | boolean | `false` | If set to `true`, the popup menu selected and displaying on the click of the link. |

## LocalizedLabel

An object representing a localized label.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| source | string | | The partial path to the i18n file containing the localized label (e.g. `/i18n/TrackerAppMessages`). |
| key | string | | The key of the localized label within the i18n file. |

## LocalStorageEventCallback

The callback function invoked when a local storage event is published

### Parameters

| Name | Type | Default | Description |
|---|---|---|---|
| eventData | object | | Data contained in the event that was published |

## LocationContext

A configuration object identifying the current location and sublocation.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| ActiveWorkspace:Location | string | | The name of the current location. |
| ActiveWorkspace:SubLocation | string | | The name of the current sublocation. |

## LocationTitles

A configuration object containing location titles.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| browserTitle | string | | The title to display in the browser tab. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| browserSubTitle | string | | The subtitle to display in the browser tab. |
| headerTitle | string | | The header title displayed in the common header. |

## MessageConfigurationObject

An object represnts message type, message text and message text links (if necessary) defined in message section.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| localizedMessage | string | | Represents message to be displayed in message section. |
| messageDefn | MessageDefn | | Represents message text and message type to be displayed in message section. |

## MessageDefn

An object representing message type, message text in message section.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| messageText | string | | The text of the message. |
| messageType | MessageTypeEnum | | The type of message. |
| messageTextParams | Array.string | | An optional list of parameters that can be referenced in the `messageText` positionally using placeholders like `{0}`, `{1}`, etc. |
| messageTextLinks | Array.MessageTextLink | | An optional list of links that can be referenced in the `messageText` positionally using placeholders like `,  `, etc. |
| navigationOptions | Array.MessageNavigationOption | | An optional list of buttons to display at the bottom of the message. |

## MessageNavigationOption

An object representing a button added to a message that can execute an action (see MessageDefn).

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| option | string | | A unique identifier for the option. |
| text | string | | The button text. |
| action | string | | An optional action to execute when the button is clicked. |

## MessageTextLink

An object representing a link to be included in the text of a message (see MessageDefn).

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| prop | ViewModelProperty | | A ViewModelProperty (containing just `displayName` and `type` set to `STRING`) used to specify the text of the link. |
| action | string | | An action to execute when the link is clicked. |

## MessageTypeEnum

Represents the type of a message displayed to the user.

### Values

- `"INFO"`
- `"WARNING"`
- `"ERROR"`

## NavigateAction

An object used to represent the declarative action of type **"Navigate"**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| actionType | string | | The action type (set to `Navigate`). |
| navigateTo | string | | The new URL to navigate to. URL either refers to state, or refers to fully-qualified URL pointing to an external web site. |
| navigationParams | object | | An object containing the navigation parameters to pass via URL. |
| navigateIn | string | | Whether navigate to new tab, new window, or refresh the same page. Valid options are `newWindow` and `newTab`. |
| options | object | | While navigating to the new window, optional attributes for the window can be provided as options. |

## NotificationCellProperty

An object representing a property to display in a notification cell. The property can only be used for aw-notification-cell.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| cellHeader1 | string | | Specifies the title of the cell. |
| notificationId | string | | Represents the notification cell indexing. |
| notificationLevel | NotificationLevelTypeEnum | | Adds a cell decorator on the right side of the cell based on the notification level and also set the color of the title of the cell. |

## NotificationLevelTypeEnum

Represents the values which adds cell decorator (color) on the left edge of the cell based on the level and also set the color of the title of the cell.

### Values

- `"high"` High importance
- `"medium"` Medium importance
- `"low"` Low importance

## PanelCollapseEnum

Specifies the panel toggle options for components **aw-command-panel-section**, **aw-panel-section**.

### Values

- `"true"`
- `"false"`
- `"disabled"`

## PicClassEnum

Defines the possible `class` values for **aw-pic** component.

### Values

- `"sw-pic-thumbnail"`
- `"sw-pic-tile"`
- `"sw-pic-small"`
- `"sw-pic-medium"`
- `"sw-pic-large"`

## PlacementOptions

Represents the placement options for popup.

### Values

- `"top"`
- `"top-start"`
- `"top-end"`
- `"right"`
- `"right-start"`
- `"right-end"`
- `"bottom"`
- `"bottom-start"`


Siemens Web Framework
Unpublished work. © 2025 Siemens

- `"bottom-end"`
- `"left"`
- `"left-start"`
- `"left-end"`

## PointCoordinates

Specifies the coordinates of a point on the screen, typically used to indicate an offset in terms of x and y axis.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| x | number | | Number of pixels on the x axis. |
| y | number | | Number of pixels on the y axis. |

## PopupAlignmentEnum

Specifies where to align the arrow relative to reference element.

### Values

- `"auto"`
- `"center"`
- `"start"`
- `"end"`

## PopupOptions

Specifies the popup options to configure **aw-popup**. Note that the `preset` value can be used to configure popups with common behaviors more easily, instead of configuring each single option.

## Properties

| Name | Type | Default | Description |
|---|---|---|---|
| adaptiveShift | boolean | `false` | If the content of the popup is growing after the popup is opened, flips the main side in positioning to enable growing from bottom to top or right to left and get its correct size. |
| advancePositioning | boolean | `false` | If set to `true`, prevents overlaps or flashing when the content inside the popup grows. This is particularly useful when the popup content is loaded asynchronously or when the flex container has the potential to grow vertically or horizontally. |
| anchor | string | | Represents a comma-separated list of identifiers used to identify the placement of commands, to be referenced in commandsViewModel.json files. |
| arrowOptions | ArrowOptions | | Represents arrow alignment, with other configuration items. |
| autoFocus | boolean | `true` | If set to `false`, the popup will receive focus when opened. |
| boundary | Element | `window` | The element to used as a boundary for the popup. |
| caption | string | | Represents the title of popup. |
| clickOutsideToClose | boolean | `true` | If set to `true`, the popup will be closed if user clicks outside the popup area. |
| closeWhenEsc | boolean | `true` | If set to `true`, the popup will be closed when the ESC key is pressed. |
| disableClose | boolean | `false` | If set to `true`, the popup cannot be closed once it has been displayed. |
| draggable | boolean | `false` | If set to `true`, the user can drag popup over the screen. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| draggableClassName | string | | The name of a custom CSS class to add to the popup draggable handler, if the popup is draggable. |
| enableAppend | boolean | `true` | If set to `true`, the popup will be appended to a container element specified via the `parent` property. Set to `false` when nesting popups. |
| flipBehavior | **FlipBehaviorOptions** | `'clockwise'` | Represents the the position engine to search a available space to place the popup. |
| forceCloseOthers | boolean | `true` | If set to `false` other popups will be closed when the popup is displayed. |
| handle | **Element** | | The element used as the draggable handle of the popup, if the popup is draggable. It can be set to a valid Element or a **CSS Selector** string. |
| hasArrow | boolean | `false` | If set to `true` an arrow will be added to the popup (used in baloon and tooltip popups). |
| hasCloseButton | boolean | `false` | If set to `true`, close button will be shown at top right corner of popup. |
| hasMask | boolean | `false` | If set to `true` a mask will be added around the popup to prevent clicking through (used in modal popups). |
| height | string | | Specifies height of the popup. |
| hideDuration | number | | Specifies the duration of the animation to hide the popup (used in tooltips). |
| hooks | **PopupHooks** | | An object that can be used to specify popup lifecycle hooks. |
| ignoreLimit | boolean | | In case of modal popups, allows the popup to exceed its default size limits in relation to the viewport. By default, the limits are 0.6 times the viewport width |

| | | | |
|---|---|---|---|
| | | | and 0.8 times the viewport height. |
| innerClassName | string | | The name of a custom CSS class to add to the popup container. |
| isModal | boolean | `true` | *Deprecated*, use `hasMask` instead. |
| marginBufferSize | number | `20` | Specifies the gap/buffer size for the drop shadow when the available space is limited. |
| minSize | number | `50` | Specifies the minimum size of the popup, in pixels, for both the width and height. |
| multiple | boolean | `false` | If set to `true`, multiple popups are allowed to be displayed at the same time. |
| overlapOnReference | boolean | `false` | If set to `true`, enables popup overlapping with the reference element. |
| padding | **PointCoordinates** | | Specifies the padding between the popup and reference element. Use this option when you expect a offset relative to the reference element. |
| parent | **Element** | `document.body` | The parent element that will contain the popup. It can be set to a valid Element or a **CSS Selector** string. |
| placement | **PlacementOptions** | | Specifies where to place the popup relative to the reference element. It is only valid when you have `reference` specified. |
| preset | string | | Specifies a preset for the popup configuration. This corresponds to a set of options (see the dedicated section). |
| reference | **Element** | | Represents the the reference element where the popup element will be relatively placed to, it accepts native element or css selector. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| relative | boolean | `true` | If set to `true`, position the popup relative to the parent element. |
| resizable | boolean | `false` | If set to `true`, the user can resize popup. |
| resizeToClose | boolean | `false` | If set to `true`, the popup will be closed if the viewport is resized. |
| showDuration | number | | Sets the delay duration before displaying the tooltip popup. |
| targetEvent | string | | Use this option when you want to position popup on the current mouse position. It could be the native targetEvent which triggers to show the popup, or any event with clientX and clientY information. |
| toggleMode | boolean | `false` | If set to `true`, if the popup is open, clicking the element that triggered the popup will cause the popup to close. |
| useOutsideScrollEvent | boolean | `true` | Leverage scroll events outside the popup to detect scrolling, and behave according to the value specified for `whenParentScrolls`. In some cases, setting this to `false` may be needed to prevent the popup from closing. |
| view | string | | Specifies the view that user wants to display inside aw-popup. |
| whenParentScrolls | **ScrollingParentBehaviors** | `'follow'` | Specify what happens to the popup when scrolling occurs inside its parent. |
| width | string | | Specifies width of the popup. |
| subPanelContext | object | | An object containing data that will be passed to the component displayed within the popup. |

### Popup presets

The following sections contain the default options that are set when the `preset` property is set to one of the allowed values.

### modal

```
{
    relative: false,
    hasMask: true,
    innerClassName: 'aw-popup-Modal',
    clickOutsideToClose: false,
    adaptiveShift: true,
    draggable: true,
    handle: '.aw-layout-panelTitle, .panel-header',
    whenParentScrolls: 'close'
}
```

### balloon

```
{
    hasArrow: true,
    innerClassName: 'aw-popup-balloon',
    clickOutsideToClose: true,
    adaptiveShift: true,
    placement: [ 'top', 'bottom', 'left', 'right' ],
    flipBehavior: 'opposite',
    padding: {
        x: 4,
        y: 4
    },
    arrowOptions: {
        alignment: 'auto',
        offset: 5,
        shift: 15
    }
}
```

### tooltip

```
{
    autoFocus: false,
    showDuration: 100,
    hideDuration: 100,
    placement: [ 'top', 'bottom', 'right', 'left' ],
    flipBehavior: 'opposite',
    whenParentScrolls: 'close',
    adaptiveShift: true,
    resizeToClose: true,
    advancePositioning: true,
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    hasArrow: true,
    padding: { x: 4, y: 4 },
    arrowOptions: {
        alignment: 'center'
    },
    minSize: 5,
    hasPadding: true,
    forceCloseOthers: false,
    innerClassName: 'sw-popup-tooltip sw-popup-easein'
}
```

## PopupHooks

Specifies callback functions to be executed during the popup lifecycle.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| whenOpened | function | | A function to be executed when the popup is opened (the reference to the popup element is passed as parameter to this function). |
| whenUpdated | function | | A function to be executed when the popup is updated (the reference to the popup element is passed as parameter to this function). |
| whenClosed | function | | A function to be executed when the popup is closed (the reference to the popup element is passed as parameter to this function). |

## PropertyLoadInput

PropertyLoadInput contains an array of PropertyLoadRequest objects

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| propertyLoadRequests | Array.object | | Array requests to fulfill. |

## PropertyTypeEnum

Defines the type of a **ViewModelProperty**.

### Values

- `"STRING"` String
- `"INTEGER"` Integer
- `"BOOLEAN"` Boolean
- `"DATE"` Date
- `"DATETIME"` Date and time
- `"FLOAT"` Float
- `"CHAR"` Single character
- `"OBJECT"` Object
- `"CHECKBOXARRAY"` Array of checkboxes
- `"DOUBLEARRAY"` Array of double values
- `"DATEARRAY"` Array of date values
- `"DATETIMEARRAY"` Array of date/time values
- `"INTEGERARRAY"` Array of integer values
- `"STRINGARRAY"` Array of string values
- `"BOOLEANARRAY"` Array of boolean values
- `"OBJECTARRAY"` Array of object values
- `"TIME"` time

## PropertyValue

An object representing a property value defined via an internal value and a display value.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| propDisplayValue | string | | The displayable name of the property. |
| propInternalValue | string | | The value of the property. |
| dispValue | string | | Represents the displayable value. |
| propDisplayDescription | string | | Represents the displayable description of the property. |
| iconName | string | | Represents the name of icon to be displayed. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## RenderingHintEnum

Defines the possible values for the `renderingHint` property of the **ViewModelProperty** object.

### Values

- `"textarea"` Text area
- `"checkboxoptionlov"` Dropdown with checkboxes for multiple selection (alternate rendering for lov array)
- `"textbox"` Text box
- `"label"` Label
- `"triState"` Tri-state control
- `"textfield"` Text box
- `"radiobutton"` Radio button
- `"togglebutton"` Toggle button
- `"checkbox"` Check box
- `"longtext"` Text area
- `"date"` Date
- `"editLov"` *Teamcenter-specific*
- `"overflow"` *Teamcenter-specific*
- `"password"` Password
- `"disablearrayreorder"` *used to disable re-ordering in arrays*

> **Tip**
>
> You can also specify a custom string in PascalCase as a rendering hint. In this case, the corresponding component will be used to edit the **ViewModelProperty**. For example, if you specify `AwCustomListbox`, the framework will attempt to load a component called `AwCustomListbox` (`aw-custom-listbox`) when the property is being edited.

## RenderingStyleEnum

Specifies the rendering style of property.

### Values

- `"titled"`

- `"headless"`

## ResponsiveLayoutGridSizes

A string used to identify the possible sizes of a layout **column** depending on screen size.

Sizes must be specified as a comma-separated list of *deviceSize*: *size* pairs, where:

- *deviceSize* is one of the following strings, identifying a specific device size: `xlarge`, `large`, `small`, `xsmall`.
- *size* is a valid**LayoutGridSize**.

Example: `xlarge: 2, large: 3, medium: 4, small:6, xsmall: 12`

## ScrollingParentBehaviors

Represents the behavior of the popup when the parent element scrolls.

### Values

- `"follow"` Keep the popup open and follow the parent component when scrolling occurs.
- `"close"` Close the popup when the parent is scrolling.

## SelectionModel

Defines the selection model which contains a list of selected UIDs and other internal state information such as multi-select state, selection mode, and selection status.

Please refer to **SelectionModel** for more information.

### Methods

#### setSelection(SelectionModel: *SelectionModel*, newSelection: *string*)

Sets the selections in the **SelectionModel**


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| SelectionModel | SelectionModel | The SelectionModel object to set the selection on |
| newSelection | string | What to set as selection |

## getSelection() → *Array.string*

Gets the current selections in the SelectionModel

### Returns

Array.string selections being tracked by this SelectionModel

## addToSelection(selections: *string*)

Adds something to the current SelectionModel. The old selections will be concatenated with the new selections.

### Parameters

| Name | Type | Description |
|---|---|---|
| selections | string | Array of selections that are supposed to be added |

## removeFromSelection(selections: *string*)

Removes something from the current SelectionModel.

### Parameters

| Name | Type | Description |
|---|---|---|
| selections | string | Array of selections that are supposed to be removed |

## toggleSelection(selections: *string*)

Toggles the selection of something in the SelectionModel.


Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| selections | string | Array of selections that are supposed to be toggled |

### selectNone()

Clears the current selection from the **SelectionModel**.

### isSelectionModelActivated() → *boolean*

Returns if the **SelectionModel** is activated or not. Active selection model will show selections as active/highlighted while inactive selection model will show selections as inactive/greyed out.

### Returns

boolean if the SelectionModel is activated or not

### setSelectionModelActivated(isSelectionModelActive: *boolean*)

Takes a boolean to activate/inactivate the **SelectionModel** and hence the selections within it

### Parameters

| Name | Type | Description |
|---|---|---|
| isSelectionModelActive | boolean | Boolean to activate/inactivate the Selection Model and hence the selections within it |

## SelectionModelModeEnum

Defines the possible types of selection mode.

### Values

- `"single"` Allows the user to select a single item.
- `"multiple"` Allows the user to select multiple items.
- `"none"` No selection is supported.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## SeriesMarker

An object used to configure the series marker for a **chart** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| symbol | string | | Respresnts the type of chart point. |
| radius | string | | Represents radius of point for scatter chart. |
| lineColor | string | | Represents line color. |
| lineWidth | string | | Represents width of line. |

## SeriesTooltip

An object used to configure the series tooltip for a **chart** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| headerFormat | string | | |
| pointFormat | string | | |

## SeriesTypeEnum

Specifies the type of series supported for **chart-points**.

### Values

- `"column"`
- `"pie"`
- `"line"`
- `"scatter"`

## SideNavConfigObject

An object used to configure a **aw-sidenav** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| slide | SlideTypeEnum | | This parameter defines the slide behavior of the sidenav. |
| direction | DirectionTypeEnum | `"LEFT_TO_RIGHT"` | This attribute will decide the direction of sidenav panel. The direction defines the shadow and rounding of corners only. |
| animation | boolean | `true` | This attribute decides animation for opening and closing of sidenav. The animation of a panel cannot be changed from eventData. |
| width | SideNavWidthEnum | `"STANDARD"` | Represents the width for GLobal navigation panel and Information panel. |
| height | SideNavHeightEnum | `"FULL"` | Represents the height for GLobal navigation panel and Information panel. |

## SideNavHeightEnum

Represents the height for GLobal navigation panel and Information panel.

### Values

- `"DEFAULT"` Default height of the panel is equal to the width defined for the panel.
- `"LARGE"` Height of the panel will be 75% of the container.
- `"FULL"` Height of the panel will be 100% of the container.

## SideNavWidthEnum

Represents the width for GLobal navigation panel and Information panel.

### Values

- `"STANDARD"` Width for Global navigation panel is 180px and Width for Information panel is 360px.
- `"WIDE"` Width for Global navigation panel is 280px and Width for Information panel is 540px..


Siemens Web Framework
Unpublished work. © 2025 Siemens

## SidenavModeEnum

Represents the possible values for the sidenavMode property that is used to determine the current size of a dialog (see **DialogOptions** for more information).

### Values

- `"mobile"` Set if the width of the dialog is less than 400px.
- `"tablet"` Set if the width of the dialog is between 400px and 800px.
- `"desktop"` Set if the width of the dialog is more than 800px.

## SliderLabelMap

Represents a label displayed by an **aw-slider**, used in the **configuration** of slider values.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| key | string | | Represents the display label value of slider. It can take values between `min` and `max` of **SliderOption**. |
| value | string | | Represents the slider label to be displayed for the corresponding key. |

## SliderOption

Represents an option used in the **configuration** of the value of an **aw-slider** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| disabled | boolean | `false` | Must be set to false. If set to true, slider configuration gets disabled. |
| min | number | | Specifies minimum display label value of the slider. |
| max | number | | Specifies maximum display label value of the slider. |
| step | number | | Specifies change in label value for each step. |
| value | number | | Specifies initial default value of slider. |

| | | | |
|---|---|---|---|
| range | boolean | | If set to "true" it enables the slider to be scrolled from both the ends to specify a range. |
| values | Array.number | | This property works with `range` property. If `range` is set to true, then this represents an array with two numeric values, representing default state of slider. |
| orientation | SliderOrientation | | Specifies the orientation of the slider. |

## SliderOrientation

Defines the orientation of a slider

### Values

- `"horizontal"`
- `"vertical"`

## SliderValue

Represents an object used to configure the value of an aw-slider component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| sliderOption | SliderOption | | Set of options to configure the behavior of slider. |
| showIncrementButtons | boolean | | Must be a boolean value. If set to "true" we get increment or decrement buttons to toggle the slider position. |
| labelValuePrefix | string | | Specifies a string literal to be prefixed with slider display label value. |
| labelValueSuffix | string | | Specifies a string literal to be prefixed with slider display label value. |
| showSliderDisplayValueLabel | boolean | | If set to true, display value label of slider is shown based on configuration of `sliderOption`. |
| sliderCreateEventAction | string | | Specifies name of action to be executed when slider is created at first. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| sliderStopEventAction | string | | Specifies name of action to be executed when slider has stopped after "sliderChangeEventAction". |
| sliderChangeEventAction | string | | Specifies name of action to be executed every time slider position is changed. |
| displayValueLabelMap | Array.SliderLabelMap | | Represents an array of objects to map slider display labels with slider values. |

## SlideTypeEnum

Defines the slide behavior of the sidenav.

### Values

- `"FLOAT"` It will open sidenav above all UI elements in a floating panel.
- `"PUSH"` It will open sidenav that pushes the page content.

## SortCriteriaConfiguration

An object representing a way to sort data in an **aw-splm-table** component. Used when **configuring a column provider**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| fieldName | string | | The name of the field to sort upon. |
| sortDirection | SortDirectionEnum | | The direction to sort the data. |

## SortDirectionEnum

Defines all possible sorting directions. Used in **SortCriteriaConfiguration**.

### Values

- `"ASC"` Ascending order.
- `"DESC"` Descending order.

## SourceEditorConfig

An object used to configure an **aw-source-editor** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| language | EditorLanguageEnum | | Specifies the language written in editor. |
| theme | EditorThemeEnum | `'vs'` | Represents the theme of the editor. |
| height | string | | Represents the height of the editor. |
| width | string | | Represents the width of the editor. |
| options | EditorConfigOptions | | Represents an options to configure editor. |

## StepConfiguration

An object used to configure a **step**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| dbValue | string | | The real value of the property, i.e. internal (database) representation of the property value.. |
| propertyDisplayName | string | | Represents the displayable name of step. |
| isCompleted | boolean | `false` | It will set to `true` when the step is completed. |
| isCurrentActive | boolean | `false` | It will set to `true` for a step when user reaches on that particular step while navigating. |
| isInProgress | boolean | `false` | It will set to `true` for a step which is in process of completion. |

## SupportedViewModes

An object used to represent the supported view modes in **Workarea**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| pwaViewModes | Array.string | | Represents an array of supported PWA views. |
| pwaSingleViewModes | Array.string | | Represents an array of pwaViewModes that should be shown in **'Awp0SinglePaneLayout'**. |
| secondaryWorkarea | boolean | | Represents if secondaryWorkarea is supported. |

## TabConfiguration

Represents an object used to configure a tab, used by the **aw-tab-set** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| name | string | | Name to be displayed on the tab. |
| tabkey | string | | Unique key to identify the tab within the tab set. |
| panelId | string | | Unique id to be used as View and ViewModel to load the declarative content of a tab. This should be used to organize each tab's content in a separate view/view model. |
| visibleWhen | string | | Is evaluated for visibility of the tab. This should be a Boolean value or expression or condition object. |
| iconId | string | | Icon to be shown on the tab title. |
| source | string | | URL to image to be shown on the tab title. |
| closeable | boolean | `false` | If set to `true`, tab will be shown as closeable. |
| closeCommandIcon | string | | Specify an icon that used as the close button, if the tab is closeable. |
| selectedTab | string | | Is evaluated for selection of the tab on load. This should be a Boolean value. |

## TableContext

An object representing the context and settings of a reusable **aw-splm-table** component. Besides the properties listed below, this object can contain arbitrary properties as well.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| isBulkEditing | boolean | | If set to `true`, the table is in edit mode. |
| startEdit | boolean | | If set to `true`, the table will be placed in edit mode. |
| cancelEdit | boolean | | If set to `true`, the table will cancel edit mode. |
| saveEdit | boolean | | If set to `true`, the table will save changes and exit edit mode. |
| showCheckbox | boolean | | If set to `true`, the table will display checkboxes for selection. |

## TableLoadResult

The object is used by the **aw-splm-table** component when used in *tree* mode to communicate the results of a single paged loading of child nodes in a parent node.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| rowsLoaded | Array.**ViewModelObject** | | Array of ViewModelObjects resulting from a single load execution in the context of a flat table. |
| totalFound | number | | The total number of known rows regardless of how many are currently loaded into the flat table. |
| nextSearchIndex | number | | The 'page' to start the 'next' page loading at (or -1 if no more rows exist). |

## TableRowHeightEnum

Defines all possible row heights for an **aw-splm-table** component.

### Values

- `"LARGE"` Large (56px)
- `"XLARGE"` Extra Large (72px)
- `"XXLARGE"` Extra Extra Large (88px)


Siemens Web Framework
Unpublished work. © 2025 Siemens

## TableTreeNode

An object used to represent a node (row) of **aw-splm-table** component in tree mode.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| uid | string | | The unique identifier of the object. |
| id | string | | An additional unique identifier of the object. |
| alternateID | string | | An alternative identifier for the object. |
| props | Record.<string, **ViewModelProperty**> | | The properties of the object. |
| typeIconURL | string | | The URL to the icon representing the type of the object. |
| levelNdx | number | | The number of levels down from the root or top of the tree table. |
| parentNode | **TableTreeNode** | | The parent node. |
| childNodes | Array.**TableTreeNode** | | The list of child nodes. |
| isLeaf | boolean | | If set to `true`, the node is a leaf node and does not have any children. |

## TextPositionEnum

Defines the possible placements for text in **aw-avatar**.

### Values

- `"RIGHT"` Display the text to the right of the avatar.
- `"BOTTOM"` Display the text below the avatar.

## ThemeIndexEnum

Represents possible values for the theme index.

### Values

- `"0"` Light: light tile color from theme.

- `"1"` Medium: Medium tile color from theme.
- `"2"` Highlight: Highlight tile color from theme.
- `"3"` Alternate: this color is not yet used by ootb themes.

## TileActionObject

An object representing type of an action to be executed on click of tile.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| actionType | TileActionTypeEnum | | Represents the possible values for type of action object. |
| url | string | | Represents the type of url based on action type. |
| commandId | string | | Represents command Id which needs to be executed after clicking the tile. This can only used for action Type 3 i.e. command type. |

## TileActionTypeEnum

Represents possible values for the type of action object.

### Values

- `"0"` Action Type - Default. On click go to provided history token.
- `"1"` Action Type - External Link. On click open provided url in a new window or tab.
- `"2"` Action Type - Static Resource. On click open provided resource in a new window or tab. Resource link should be relative to host html file.
- `"3"` Action Type - Command Resource. On click execute the command with the provided id.

## TileConfiguration

An object used to configure an **aw-tile** component.


Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| action | TileActionObject | | Represents the action to execute on click of tile. |
| content | TileContentObject | | Represents the content value and name inside tile. |
| displayName | string | | Specifies the displyable name of the tile. |
| icons | TileIconsObject | | Represents the type of tile icon. |
| isProtected | boolean | | Determines if the tile can be unpin. If set to `true`, tile cannot be unpinned and unpin command will not be shown when tile is in edit mode. |
| orderNumber | string | | Represents the order number which determines the order of the tile in a tile group. |
| styleOverride | string | | Specifies custom CSS class name which can be provided that gets added to tile container. |
| themeIndex | ThemeIndexEnum | | Represents possible values for the tile index. |
| tileSize | TileSizeEnum | | Represents possible values for the tile size. |

## TileContentObject

An object representing tile content.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| contentValue | string | | When content value is provided it is shown in the center of the tile for wide and large tile sizes. |
| contentName | string | | When content name is provided it is shown in the center of the tile below content value for wide and large tile sizes. |

## TileGroupConfiguration

An object used to configure an aw-tile-group/aw-tile-canvas component.


Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| groupName | string | | Represents group name of tile group. This property is only used internally and not represented in the UI. |
| tiles | Array.TileConfiguration | | Represents array of tile object. |

## TileIconsObject

An object representing type of a tile icon.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| primaryIcon | string | | Represents the iconId which is shown at the center of the tile. To show a thumbnail image, thumbnail image ticket has to be provided. |
| secondaryIcon | string | | Represents the iconId which is shown at the left corner of the tile when provided. If TYPEICON is provided as value then typeIcon of the object is shown. |

## TileSizeEnum

Represents possible values for the tile size.

### Values

- `"0"` Small: 1x1 tile size
- `"1"` Wide: 2x1 tile size
- `"2"` Large: 2x2 tile size

## TransistionHooks

A transition has numerous lifecycle events. You can tap into each of the lifecycle events by registering a “Transition Hook”. A Transition Hook is a callback function that is run during the specific lifecycle event of a Transition.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| onEnter | OnEnterStateTransition | | Registers a lifecycle hook, which is invoked (during a transition) when a specific state is being entered. |
| onSuccess | OnSuccessStateTransition | | Registers a transition lifecycle hook, which is invoked after a transition successfully completes. |
| onError | OnErrorStateTransition | | Registers a transition lifecycle hook, which is invoked after a transition has been rejected for any reason. |
| onBefore | OnBeforeStateTransition | | Registers a transition lifecycle hook, which is invoked before a transition even begins. |
| This hook can be useful to implement logic which prevents a transition from even starting,such as authentication, redirection | | | |

## TreeData

An object used to represent a TreeData of **aw-tree** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| label | string | | Display name of the tree nodes. |
| value | string | | Value associated with the tree node label. |
| expanded | boolean | | Specifies if the node is expanded or not. Default is `false`; |
| children | Array.TreeData | | The list of child nodes. |

## TreeLoadInput

The object used to represent the input structure to a request to load a page of rows (nodes) into a tree-table.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| parentNode | ViewModelTreeNode | | The 'parent' ViewModelTreeNode used to access any 'child' nodes in |
| the next level down in the hierarchy. | | | |
| startChildNdx | number | | The 'childNdx' of the node to start the 'next' page loading at. |
| startChildId | string | | The 'id' of the node to start the 'next' page loading at. |
| cursorNodeId | string | | ID of an existing node in the 'parent' (and, presumably a |
| ViewModelCollection) to insert any new nodes after (or before...depending on value of 'addAfter'). | | | |
| addAfter | boolean | | TRUE if any new children should be added AFTER the optional 'cursorNodeId' |
| (Default: TRUE) | | | |
| pageSize | number | | The maximum # of 'child' nodes to return in any single paged access |
| rootNode | ViewModelTreeNode | | The 'tree' ViewModelTreeNode used to access any 'child' nodes in the |
| next level down in the hierarchy. | | | |
| focusLoadAction | boolean | | (Optional) TRUE if action is called to load selected object that is |
| currently not present in tree | | | |

## UploadModel

Represents an array of file upload **model object** used to configure an **aw-file-upload** component.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| Model | Array.UploadModel_Object | | Object Represents an array of file upload model object. |

## UploadModel_Object

An object used to define a model for upload-model property of **aw-file-upload** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| name | string | | Specifies the Label to be displayed. |
| priority | string | | Specifies the order of priority for multiple models. Models are sorted as per priority. |
| iconName | string | | Specifies the icon to be displayed alongside the label file . |
| isFolder | boolean | | Specifies if the given model supports folder upload. |
| popupViewId | string | | Specifies the view component to load when user choose given model. |
| options | PopupOptions | | Typical popup options to define the behavior for popupViewId. |

## UploadedFileMetadata

An object representing the metadata of a file uploaded via an **aw-file-upload** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| fileName | string | | The full name of the uploaded file. |
| fileNameNoExt | string | | The name of the uploaded file without extension. |
| fileExt | string | | The extension of the uploaded file. |
| formData | FormData | | The FormData object containing the contents of the uploaded file. |

## UploadedFiles

An object containing data and information on files that have been uploaded via the **aw-file-upload** component.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| files | Array.**File** | | An array of File objects. |
| selectedFileInfo | **UploadedFilesMetadata** | | An object containing information on each file that was uploaded |

## UploadedFilesMetadata

An object representing the metadata of a set of files uploaded via an **aw-file-upload** component. Accessible through an **UploadedFiles** object.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| fileExt | Array.string | | Array of file extensions. |
| fileName | Array.string | | Array of file names with extension. |
| fileNameNoExt | Array.string | | Array of file names without extension. |
| formData | Array.**FormData** | | Array of FormData objects containing file data for a specific file. |
| validFile | boolean | | Whether the uploaded files are valid or not. |

## ViewModeContext

An object used to represent the current layout context in **Workarea**. Examples and further details about the layout context can be found in **ViewModes and Layouts**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| displayedViewModes | **DisplayedViewModes** | | An object used to represent the currently displayed view modes. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| supportedLayouts | Array.string | | An array of supported layouts id. |
| supportedViewModes | SupportedViewModes | | An object which contains supported view modes. |
| viewTuples | Array.ViewTuple | | An array of view tuples which are visible. |

## ViewModelCollection

Defines an object which maintains the loaded ViewModelObjects and exposes APIs to access the information of collection.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| totalFound | number | | The number VMOs in the collection, including those that are not loaded. |
| totalObjectsLoaded | number | | The number of VMOs currently loaded in the collection. |

### Methods

#### getViewModelObject(uid: *string*) → *ViewModelObject*

Retrieves a ViewModelObject in the collection by its uid.

### Parameters

| Name | Type | Description |
|---|---|---|
| uid | string | The uid of the VMO to retrieve. |

### Returns

ViewModelObject The VMO identified by the specified uid.

#### getLoadedViewModelObjects() → *Array.ViewModelObject*

Retrieves a list of ViewModelObjects that are currently loaded in the collection.

### Returns

Array.**ViewModelObject** The list of loaded VMOs.

## ViewModelProperty

An object representing a typed value. A view model property (VMP) is typically used to configure property widgets.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| dbValue | any | | The real value of the property, i.e. internal (database) representation of the property value. |
| propName | string | | The unique identifier of the property. |
| displayName | string | | The displayable name of the property. |
| displayValue | string | | Represents the current display value(s) of the property. |
| type | **PropertyTypeEnum** | | The data type of the property. |
| isArray | boolean | `false` | If set to `true`, the value of the property is an array. |
| isEditable | boolean | `false` | If set to `true`, the user can change the property value, otherwise the property will be read-only. |
| isRequired | boolean | `false` | If set to `true`, the user must enter a value for the property for it to be valid. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| isEnabled | boolean | `true` | If set to `true`, the property value is displayed normally, if set to `false` it will be grayed out and it will not react to use input (even if editable). |
| isModifiable | boolean | `true` | If set to `true`, the user should have the ability to modify the value. If set to `false`, it should be treated as read-only. |
| isAutoAssignable | boolean | `false` | If set to `true`, the property value should be assigned automatically by the server. |
| isRichText | boolean | `false` | *(Teamcenter-specific)* If set to `true`, the string value of the property is in HTML format and should be displayed using HTML formatting rules and edited with a rich text editor. |
| hasLov | boolean | `false` | If set to `true`, the property has a specific list of value associated to it. |
| labelPosition | **LabelPositionEnum** | `"PROPERTY_LABEL_AT_SIDE"` | Determines the placement of the property label. |
| maxLength | number | | If not equal to `-1` or `0`, specifies the maximum number of characters allowed for the value of the property, if its type is set to `STRING`. |
| maxArraySize | number | `-1` | If `isArray` is set to `true`, specifies the maximum number |

| | | | |
|---|---|---|---|
| | | | of array elements permitted. |
| requiredText | string | `"Required"` | The text to display as placeholder when the property is marked as required. |
| referenceTypeName | string | | *(Teamcenter-specific)* The name of the corresponding typed reference property. |
| renderingHint | **RenderingHintEnum** | | Determines the control to use to render the property. It can also be set to the name of a custom component that will be loaded when editing the property. |
| isSelectOnly | boolean | `false` | If set to `true`, does not allow the user to enter custom values via a LOV control (**aw-listbox**). |
| autofocus | boolean | `false` | If set to `true`, the control displaying the property will be focused. |
| uid | string | | *(Teamcenter-specific)* The uid of the original ModelObject (in the client datamodel cache) that was used to originate this ViewModelProperty object. |
| patterns | Array.string | | *(Teamcenter-specific)* The list of patterns associated to this property. |
| preferredPattern | string | | *(Teamcenter-specific)* The pattern to select. If no 'preferredPattern'is |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | present, then the first entry in `patterns` will be used. |
| validationCriteria | string | | The condition to use to validate the value of the property. |
| oldValue | any | | *(Teamcenter-specific)* The old value of the property. |
| oldValues | Array.any | | *(Teamcenter-specific)* Array of old values of the property. |
| deltaValues | Array.any | | An array of objects representing the changes in the property value. Each object in the array has two properties: `type` and `value`. type is a string that indicates the type of change. It can be *Unchanged*, *Removed*, or *Added*. value is a string representing the actual value associated with the change. |
| vertical | boolean | `false` | *(Used if an **aw-radiobutton** is used to render the property)* If set to `true`, the radio buttons will be displayed vertically. |
| propertyRadioTrueText | string | | *(Used if an **aw-radiobutton** is used to render the property)* The label for the `false` radio button. |
| propertyRadioFalseText | string | | *(Used if an **aw-radiobutton** is used to render the* |

| | | | |
|---|---|---|---|
| | | | *property)* The label for the `false` radio button. |
| srcObjLsd | object | | *(Teamcenter-specific)* Source object last-saved date. |
| srcObjTypeName | object | | *(Teamcenter-specific)* Source object type name. |
| numberOfLines | number | | If not equal to `-1` or `0`, specifies the number of lines allowed in the property. |
| dataProvider | string | | The named of the data provider associated with the property, to fetch the list of permitted values (if the property is rendered as a list of values). |

## Creating ViewModelProperty objects

A ViewModelProperty object is created automatically if an object defined in the `data` section of the ViewModel contains one or more of the following properties. To avoid this, prepend an underscore (_) to the name of the object defined in the `data` section: dbValue, displayName, dispValue, isArray, isEditable, isRequired, isEnabled, displayValue, value, isModifiable, isAutoAssignable, labelPosition, requiredText, type, propName, hasLov, dataProvider, isSelectOnly, isRichText, referenceTypeName, maxLength, maxArraySize, renderingHint, autofocus, patterns, preferredPattern, condition, validationCriteria, oldValue, oldValues, vertical, propertyRadioTrueText, propertyRadioFalseText, numberOfLines.

## Input grammar, output grammar, and field properties

In some cases, the names of some ViewModelProperty object properties may be different from the ones used when first creating the ViewModelProperty object in the view model.

The following table lists the correspondence between the names of the ViewModelProperty object properties used at creation time (documented above), the property names actually present on the object after creation, and the corresponding property names when the ViewModelProperty object is **automatically converted to a Field object**.


Siemens Web Framework
Unpublished work. © 2025 Siemens

**Note**

The following table only includes properties whose name either changes after creation, or is used to populate a field property.

| Input Property Name | Output Property Name | Field Property Name |
|---|---|---|
| displayName | propertyDisplayName | label, aria-label |
| propName | propertyName | name |
| dbValue | dbValue, value, dbValues | value |
| displayValue | uiValue, displayValues, uiValues | label |
| isModifiable | isPropertyModifiable | |
| maxArraySize | arrayLength | |
| requiredText | propertyRequiredText | |
| labelPosition | propertyLabelDisplay | |
| uid | parentUid | |
| srcObjLsd | sourceObjectLastSavedDate | |
| type | type | typex |
| maxLength | maxLength | maxLength |
| requiredText | requiredText | placeholder |
| isRequired | isRequired | required |
| isEnabled | isEnabled | disabled *(negated)* |
| | dirty | dirty |

## Automatically-generated properties

The follwing table is a reference to the internal properties which get automatically created whenever a ViewModelProperty is created.

| Output Grammar | Type | Default | Description |
|---|---|---|---|
| isNull | boolean | `false` | Set to `true` if the current property value is *no value*, set to `false` if the value is valid as is. |
| editable | boolean | `true` | *(Teamcenter-specific)* Set to `true` if the user should have the ability to change the property value. Set to `false` if the value is read-only. |

| | | | |
|---|---|---|---|
| isLocalizable | boolean | `false` | Set to `true` if the property value UI should include the option to allow any user-entered value to be converted from local language (as entered) into another language. |
| hasInitialValue | boolean | `false` | Set to `true` if the property has an initial value. |
| valueUpdated | boolean | `false` | Set to `true` if the property value has been modified. |
| displayValueUpdated | boolean | `false` | Set to `true` if the property display value is modified in the view model. |
| editableInViewModel | boolean | `false` | *(Teamcenter-specific)* |
| error | string | `null` | The message that should be displayed when some aspect of the property value is not correct. |
| editLayoutSide | boolean | `true` | Set to `true` when a label position is specified. |
| inputType | string | | Set to `"text"` only when the ViewModelProperty is of type STRING, CHAR, or STRINGARRAY. |
| displayValsModel | Arrayobject | | *(Created only if `isArray` is set to `true`)* Object array derived from `displayValues` and `dbValues`; used to maintain the selection of the rows in an array widget |
| numberOfCharacters | number | | If not equal to `-1`or `0`, specifies the number of allowed characters for a ViewModelProperty of type STRING. |
| radioBtnApi | object | `{}` | Object created for internal used when the ViewModelProperty is rendered via **aw-radiobutton**. |
| dateApi | object | `true` | Object created for internal used when the ViewModelProperty is rendered via **aw-date** or **aw-datetime**. |
| isDateEnabled | boolean | `true` | Set to `true` if the ViewModelProperty is of type DATE. |
| isTimeEnabled | boolean | | Defaults to `true` if the ViewModelProperty is of type DATETIME (`isDateEnabled` will also be set to `true`). `false` is not supported for array properties. |
| dirty | boolean | | Set to `true` if the value has been changed. |

## Additional Teamcenter-specific properties

The following table is a reference to the Teamcenter-only internal properties which can be explicitly created on a ViewModelProperty.

| Name | Type | Default | Description |
|---|---|---|---|
| clientValidationError | boolean | | If set to `true`, sets client validation error flag of the ViewModelProperty object. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| hasServerValidationError | boolean | | If set to `true`, do not clear the error state of the ViewModelProperty object. |
| moveDownButtonTitle | string | | Localized title for the **Move Down** button in array widgets. |
| moveUpButtonTitle | string | | Localized title for the **Move Up** button in array widgets. |
| removeButtonTitle | string | | Localized title for the **Remove** button in array widgets. |
| isDisplayable | boolean | `true` | Whether the property can be displayed or not. |

### Determining the value of the placeholder property

The following table shows the placeholder value for different combinations of `required` and `requiredText` property.

| Property marked `required`? | `requiredText` specified? | What will show as place holder text? |
|---|---|---|
| True | Yes | As specified on `requiredText` |
| False | Yes | As specified on `requiredText` |
| True | No | The word `Required` |
| False | No | The phrase `Type to add value` for array field |

## ViewModelPropertyAttributeHolder

An object having the necessary attributes to construct a ViewModelProperty.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| displayName | string | | Display name of the property. |
| type | PropertyTypeEnum | | the datatype of the property |
| isRequired | boolean | | If set to `true`, the user must enter a value for the property for it to be valid. |
| isEditable | boolean | | If set to `true`, the user can change the property value, otherwise the property will be read-only. |
| dbValue | object | | The real value of the property, i.e. internal (database) representation of the property value. |
| dispValue | string | | Represents the current display value(s) of the property. |

| | | | |
|---|---|---|---|
| labelPosition | LabelPositionEnum | | Determines the placement of the property label. |
| isArray | boolean | | If set to `true`, the value of the property is an array. |
| requiredText | string | | The text to display as placeholder when the property is marked as required. |

## ViewModelObject

An object representing a dictionary of ViewModelProperty objects. A view model object (VMO) is typically used to represent a record to display in a table or a list.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| uid | string | | The unique identifier of the object. Defaults to a randomly-generated ID. |
| type | string | | The internal type of the object. |
| visible | boolean | `true` | Whether the object is visible or not. |
| props | Record.<string, ViewModelProperty> | | The properties of the object. |
| cellProperties | Record.<string, CellProperty> | | The cell properties of the object. |
| notificationCellProperties | Record.<string, NotificationCellProperty> | | The properties of notification cell object. |
| indicators | Array.Indicator | | The indicators of the object. |
| thumbnailURL | string | | The URL to the thumbnail of the object. |
| hasThumbnail | boolean | | Whether the object has a thumbnail or not. |
| typeIconURL | string | | The URL to the icon representing the type of the object. |
| alternateID | string | | An alternative identifier for the object. |

## ViewModelTreeNode

The object used to represent the node in a tree. It can be a parent node or a child node.


Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| id | string | | Unique ID for this node within the tree-table. |
| type | string | | The type of model object represented by this tree node (i.e. 'Item','DocumentRevision', etc.). |
| displayName | string | | The name to display in the 'navigation' column of the tree node. |
| levelNdx | number | | The # of levels down from the 'root' of the tree-table. |
| childNdx | number | | The index to this 'child' within the immediate 'parent'. This information is |

meant to help in when only a partial (or sparse) range of children have been loaded. This index is stable within the 'parent' and not representative of the order based on the 'id'. | | iconURL | string | | The URL to the icon associated with the display of this node in the 'navigation' column of the tree-table. | | svgString | string | | The String svg icon associated with the display of this node in the 'navigation' column of the tree-table. | | props | Array.**ViewModelProperty** | | Map of propertyName to ViewModelProperty object holding the value/state of that property (or 'undefined' if no properties have been loaded yet). | | totalChildCount | number | | The total # of known 'child' nodes regardless of how many are currently loaded into the 'children' array (or 'undefined' if no expansion has been occured or there are no children). | | children | Array.**ViewModelTreeNode** | | Array of currently loaded ViewModelTreeNode children in the next level down in the hierarchy (or 'undefined' if no expansion has been occurred or there are no children). | | isLeaf | boolean | | TRUE if we have checked and there are NO 'child' nodes beneath this 'parent' node (or 'undefined' if no expansion has been occurred or there are children). |

## ViewModelTreeNodeArray

Array of nodes to load ViewModelProperty objects for.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| | **ViewModelTreeNode** | | The 'parent' ViewModelTreeNode used to access any 'child' nodes in |
| the next level down in the hierarchy. | | | |

## ViewTuple

An object used to represent a current view in **Primary Workarea View**, **Secondary Workarea View**, **Alternate Primary Workarea View**.

### Properties

| Name | Type | Default | Description |
|---|---|---|---|
| viewId | number | | The unique ID to identify the view. |
| pwaViewMode | string | | The view that is currently being rendered in the primary work area. |
| secondaryWorkarea | boolean | | If set to `true`, the secondary work area is enabled for the current view. |
| alternatePWA | string | | The view that is currently being rendered in the alternate primary work area. |
| name | string | | The name for the view. Used in layout manager to represent view tuple. |

## WrapStyleEnum

Defines a wrap style mechanism for the row/column.

### Values

- `"wrap"` Wrap
- `"nowrap"` No wrap
- `"wrap-reverse"` Wrap Reverse

# AppCtx

The **Application Context** is a global object containing information about the application state and environment. You can read and write contexts using the **js/appCtxService** service.

## Properties

| Name | Type | Default | Description |
|---|---|---|---|
| dataProvider_editInProgress | string | | When **true**, indicates that the specific |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | dataProvider is in edit mode. |
| activeNavigationCommand_dialog | **ActiveCommandConfig** | | Indicates the currently-active Navigate command dialog. |
| activeToolsAndInfoCommand_dialog | **ActiveCommandConfig** | | Indicates the currently-active Tools and Info command dialog. |
| locationContext | **LocationContext** | | Provides information on the current location and sublocation. |
| aw_host_type | string | | *(Teamcenter-specific)* The type of the application hosting Active Workspace (used for server side command visibility in hosted mode). |
| aw_hosting_Config | object | | *(Teamcenter-specific)* The application hosting configuration (used for server side command visibility in hosted mode). |
| aw_hosting_enabled | boolean | | *(Teamcenter-specific)* Whether application hosting is enabled or not (used for server side command visibility in hosted mode). |
| editInProgress | boolean | | If set to **true**, indicates that the application is in edit mode. |
| location | **CurrentLocationData** | | Contains information on the current location. |
| mselected | Array.**ViewModelObject** | | An array containing the objects that are actively selected on the page. |
| pselected | Array.**ViewModelObject** | | An array containing the objects that representing the parent selection (if any) of the |

| | | | |
|---|---|---|---|
| | | | current active selection on the page. |
| selected | ViewModelObject | | The currently selected object (active selection). |
| showUserSettingsBar | boolean | | If set to **true**, shows the User Settings Bar in the footer. |
| state | CurrentStateData | | Provides information of the current URL state. |
| sublocation | CurrentSublocationData | | Contains information on the current sublocation. |
| sublocationTitleErrorMessage | string | | Indicates the error message which should show in common header title, if configured. |
| taskbarfullscreen | boolean | | Instructs the common application frame to go into taskbar full screen mode (no global navigation or right edge). |
| theme | string | `ui-lightTheme` | The current theme used for the application. |
| user | ViewModelObject | | Contains information about the currently logged-in user. |
| userSession | ViewModelObject | | Contains information on the current user session. |
| workspace | object | | Contains the full **workspace configuration** of the currently-active **workspace**. |


Unpublished work. © 2025 Siemens

# Configuration points

This section contains the reference documentation of all public configuration points of Siemens Web Framework.

## Action Schema

This schema points to the **actionDef** definition.

### Definitions

This schema includes the following definitions:

- **CommandActionInputDataDef**
- **CommandDef**
- **CopyDef**
- **EditDef**
- **EventDef**
- **GraphQLDef**
- **GraphQLInputDataDef**
- **JSFunctionAsyncDef**
- **JSFunctionDef**
- **NavigateDef**
- **RESTServiceDef**
- **SyncDef**
- **SyncInputDataDef**
- **TcSoaServiceDef**
- **actionDef**
- **actionMessagesDef**
- **batchJobDef**
- **dataParsersDef**
- **dataProviderActionDef**
- **dialogDef**
- **eventTypeDef**
- **policyPropertiesDef**

- popupDef
- popupInputDataDef

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeActionSchema-1.0.0.json",
  "description": "Actions can be of different actionType (REST, JSFunction,
JSFunctionAsync, Event, Command or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
  "definitions": {
    "eventTypeDef": {
      "description": "An event object can have event name and eventData OR
a message and a condition statement which would evaluate to boolean",
      "oneOf": [
        {
          "type": "array",
          "items": {
            "required": [
              "message"
            ],
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "condition": {
                "type": "string",
                "description": "A condition statement which would evaluate
to boolean."
              },
              "message": {
                "type": "string",
                "description": "The message string that would be passed
with event, if above condition evaluates to true."
              }
            }
          }
        },
        {
          "type": "array",
          "items": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "name": {
                "type": "string",
                "description": "Name of the event. The name is used by
interested listeners."
              },
              "eventData": {
                "type": "object",
                "description": "Event data is a JSON object for data to
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
provide on the event. The structure of the eventData is specific to a given
event type."
              },
              "condition": {
                "type": [
                  "object",
                  "string",
                  "boolean"
                ],
                "description": "A condition statement which would evaluate
to boolean."
              },
              "excludeLocalDataCtx": {
                "type": "boolean",
                "description": "Excludes the local ctx from event data"
              }
            },
            "required": [
              "name"
            ]
          }
        }
      ]
    },
    "actionMessagesDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Actions can send array of events/actionMessages in
case of success and failure of the action.",
      "properties": {
        "success": {
          "$ref": "#/definitions/eventTypeDef"
        },
        "failure": {
          "$ref": "#/definitions/eventTypeDef"
        }
      }
    },
    "TcSoaServiceDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "TcSoaService"
        },
        "serviceName": {
          "type": "string",
          "description": "Used with actionType 'TcSoaService'. Name of the
SOA service that should be invoked."
        },
        "method": {
          "type": "string",
```

```
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "headerState": {
          "type": "object"
        },
        "policy": {
          "type": "object",
          "description": "Used with actionType 'TcSoaService'. This is a
definition of property policy provided with SOA",
          "items": {
            "type": "array",
            "additionalProperties": false,
            "properties": {
              "name": {
                "type": "string"
              },
              "properties": {
                "$ref": "#/definitions/policyPropertiesDef"
              }
            },
            "required": [
              "name",
              "properties"
            ]
          }
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
```


Unpublished work. © 2025 Siemens

```
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "dataProviderActionDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "dataProvider"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "methods": {
          "type": "array",
          "description": "Used with actionType 'dataProvider'. Array of
dataProviders that should be invoked as part of this action."
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "EventDef": {
```

```
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "Event"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "policy": {
          "type": "object",
          "description": "Used with actionType 'TcSoaService'. This is a
definition of property policy provided with SOA",
          "items": {
            "type": "array",
            "additionalProperties": false,
            "properties": {
              "name": {
                "type": "string"
              },
              "properties": {
                "$ref": "#/definitions/policyPropertiesDef"
              }
            },
            "required": [
              "name",
              "properties"
            ]
          }
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "RESTServiceDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "RESTService"
        },
        "serviceName": {
          "type": "string",
          "description": "Used with actionType 'TcSoaService'. Name of the
SOA service that should be invoked."
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
```

```
the result of the action is mapped to view model data."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "GraphQLInputDataDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "JSON data for GraphQL input, which needs endPoint and
request.",
      "properties": {
        "endPoint": {
          "type": "string"
        },
        "request": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        }
      }
    },
    "GraphQLDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "GraphQL"
        },
        "serviceName": {
          "type": "string",
          "description": "Used with actionType 'TcSoaService'. Name of the
SOA service that should be invoked."
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
```


Unpublished work. © 2025 Siemens

```
        "inputData": {
          "$ref": "#/definitions/GraphQLInputDataDef",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "SyncInputDataDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "JSON data for GraphQL input, which needs endPoint and
request.",
      "properties": {
        "port": {
          "type": "string"
        },
        "syncObject": {
          "type": [
            "object",
            "string"
          ],
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        }
      },
      "required": [
        "port"
```

```
      ]
    },
    "SyncDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "Sync"
        },
        "inputData": {
          "$ref": "#/definitions/SyncInputDataDef",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "parameters": {
          "type": "object"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "CopyDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "type": "string",
          "const": "Copy"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "NavigateDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "Navigate"
        },
        "navigationParams": {
          "type": [
            "object",
            "string"
          ],
          "description": "Used with actionType 'Navigate'. Specifies the
parameters to be passed while navigating to the state"
```

```
        },
        "navigateIn": {
          "type": "string",
          "description": "Used with actionType 'Navigate'. Specifies if the
new url should be opened in 'newTab' or 'newWindow'"
        },
        "navigateTo": {
          "type": "string",
          "description": "Used with actionType 'Navigate'. Specifies the
state to navigate to"
        },
        "options": {
          "type": "object",
          "description": "Used with actionType 'Navigate'. Can be used to
specify the 'top', 'left', 'height', 'width' for the 'newWindow'"
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "dialogDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "dialog"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
```



Unpublished work. © 2025 Siemens

```
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        },
        "options": {
          "type": "object"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'dialog'. Points to the
method name that should be invoked from the js file specified in 'deps'"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        }
      }
    },
    "popupDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "popup"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "outputData": {
          "type": "object",
```


Unpublished work. © 2025 Siemens

```
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        },
        "options": {
          "type": "object"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'popup'. Points to the
method name that should be invoked from the js file specified in 'deps'"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        }
      }
    },
    "popupInputDataDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Actions can send array of events/actionMessages in
case of success and failure of the action.",
      "properties": {
        "options": {
          "type": "object",
          "description": "Options data"
        }
      }
    },
    "JSFunctionAsyncDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "JSFunctionAsync"
        },
        "method": {
```


Unpublished work. © 2025 Siemens

```
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "policy": {
          "type": "object",
          "description": "Used with actionType 'TcSoaService'. This is a
definition of property policy provided with SOA",
          "items": {
            "type": "array",
            "additionalProperties": false,
            "properties": {
              "name": {
                "type": "string"
              },
              "properties": {
                "$ref": "#/definitions/policyPropertiesDef"
              }
            },
            "required": [
              "name",
              "properties"
            ]
          }
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
```

```
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "JSFunctionDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "JSFunction"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "policy": {
          "type": "object",
          "description": "Used with actionType 'TcSoaService'. This is a
definition of property policy provided with SOA",
          "items": {
            "type": "array",
            "additionalProperties": false,
            "properties": {
              "name": {
                "type": "string"
              },
              "properties": {
                "$ref": "#/definitions/policyPropertiesDef"
              }
            },
            "required": [
              "name",
              "properties"
            ]
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          }
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "EditDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "Edit"
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "events": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "Events can be fired and listened/handled in
declarative View Models. They are a means of a View Model communicating
with other View Models or UI elements. 'Actions' can fire events."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef"
        }
      }
    },
    "CommandActionInputDataDef": {
      "additionalProperties": false,
```

```
      "required": [
        "commandId"
      ],
      "type": "object",
      "description": "JSON data for Command Action input, which requires
commandId and optionaly accepts commandContext.",
      "properties": {
        "commandId": {
          "type": "string",
          "description": "The command Id for the command to be executed."
        },
        "commandContext": {
          "type": [
            "string",
            "object"
          ],
          "description": "Additional context to pass to the command for
condition evaluation."
        }
      }
    },
    "CommandDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType",
        "inputData"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "Command"
        },
        "inputData": {
          "$ref": "#/definitions/CommandActionInputDataDef",
          "description": "JSON data for the action input."
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        }
      }
    },
    "batchJobDef": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "const": "batchJob"
        },
        "steps": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "type": "array",
          "description": "Collection of actions that has to executed one
after another",
          "items": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "action": {
                "type": "string"
              },
              "outputArg": {
                "type": "array",
                "description": "Array of object given to subsequent next
action."
              },
              "inputArg": {
                "type": "object",
                "description": "Input from previous action and mapped to
parameters of action."
              },
              "condition": {
                "type": "string",
                "description": "A condition statement which would evaluate
to boolean."
              }
            },
            "required": [
              "action"
            ]
          }
        },
        "outputData": {
          "type": "object"
        }
      }
    },
    "policyPropertiesDef": {
      "type": "object",
      "items": {
        "required": [
          "name"
        ],
        "additionalProperties": false,
        "properties": {
          "name": {
            "type": "string"
          },
          "modifiers": {
            "type": "array"
          }
        }
      }
    },
    "dataParsersDef": {
      "type": "array",
      "description": "In the View Model, an action can have an optional
```

```
'dataParsers' construct. It is an array of objects, each having an 'id:
specifying which dataParseDefinition to use' & a 'responseObjs: path to
the response object(s) to convert.'. The 'responseObjs' can be an array
of objects, or a single one. properties below. The dataParsers accepts
the response data and converts them to 'afx objects' each containing
ViewModelProperties. These afx objects are directly consumable by the
widgets provided by afx.",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "id": {
            "type": "string"
          },
          "responseObjs": {
            "type": "string"
          }
        },
        "required": [
          "id",
          "responseObjs"
        ]
      }
    },
    "actionDef": {
      "oneOf": [
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "TcSoaService"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/TcSoaServiceDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "dataProvider"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/dataProviderActionDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
```


Unpublished work. © 2025 Siemens

```
                "const": "Event"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/EventDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "RESTService"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/RESTServiceDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "GraphQL"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/GraphQLDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Copy"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/CopyDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Navigate"
              }
            }
          },
```

```
            "then": {
              "$ref": "#/definitions/NavigateDef"
            },
            "else": false
          },
          {
            "if": {
              "properties": {
                "actionType": {
                  "const": "popup"
                }
              }
            },
            "then": {
              "$ref": "#/definitions/popupDef"
            },
            "else": false
          },
          {
            "if": {
              "properties": {
                "actionType": {
                  "const": "dialog"
                }
              }
            },
            "then": {
              "$ref": "#/definitions/dialogDef"
            },
            "else": false
          },
          {
            "if": {
              "properties": {
                "actionType": {
                  "const": "JSFunctionAsync"
                }
              }
            },
            "then": {
              "$ref": "#/definitions/JSFunctionAsyncDef"
            },
            "else": false
          },
          {
            "if": {
              "properties": {
                "actionType": {
                  "const": "JSFunction"
                }
              }
            },
            "then": {
              "$ref": "#/definitions/JSFunctionDef"
            },
            "else": false
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Edit"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/EditDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "batchJob"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/batchJobDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Sync"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/SyncDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Command"
              }
            }
          },
          "then": {
            "$ref": "#/definitions/CommandDef"
          },
          "else": false
        }
      ]
    }
  },
```

```
  "$ref": "#/definitions/actionDef",
  "__defs": []
}
```

## actionDef Definition (Action)

This definition must satisfy *one* of the following constraints:

- if **actionType** is set to `TcSoaService` **TcSoaServiceDef**
- if **actionType** is set to `dataProvider` **dataProviderActionDef**
- if **actionType** is set to `Event` **EventDef**
- if **actionType** is set to `RESTService` **RESTServiceDef**
- if **actionType** is set to `GraphQL` **GraphQLDef**
- if **actionType** is set to `Copy` **CopyDef**
- if **actionType** is set to `Navigate` **NavigateDef**
- if **actionType** is set to `popup` **popupDef**
- if **actionType** is set to `dialog` **dialogDef**
- if **actionType** is set to `JSFunctionAsync` **JSFunctionAsyncDef**
- if **actionType** is set to `JSFunction` **JSFunctionDef**
- if **actionType** is set to `Edit` **EditDef**
- if **actionType** is set to `batchJob` **batchJobDef**
- if **actionType** is set to `Sync` **SyncDef**
- if **actionType** is set to `Command` **CommandDef**

## actionMessagesDef Definition (Action)

Actions can send array of events/actionMessages in case of success and failure of the action.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `success` | **eventTypeDef** | | |
| `failure` | **eventTypeDef** | | |


Unpublished work. © 2025 Siemens

## batchJobDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `batchJob`. |
| `steps` | Array**StepsObject** | | Collection of actions that has to executed one after another |
| `outputData` | **OutputDataObject** | | |

### Referenced Objects

The following objects are referenced within this definition.

#### InputArgObject

Input from previous action and mapped to parameters of action.

#### OutputDataObject

*No description specified.*

#### StepsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `action` | string | Yes | |
| `outputArg` | array | | Array of object given to subsequent next action. |
| `inputArg` | **InputArgObject** | | Input from previous action and mapped to parameters of action. |
| `condition` | string | | A condition statement which would evaluate to boolean. |

## CommandActionInputDataDef Definition (Action)

JSON data for Command Action input, which requires commandId and optionaly accepts commandContext.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `commandId` | string | Yes | The command Id for the command to be executed. |
| `commandContext` | CommandContextObject | | Additional context to pass to the command for condition evaluation. |

### Referenced Objects

The following objects are referenced within this definition.

#### CommandContextObject

Additional context to pass to the command for condition evaluation.

## CommandDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `Command`. |
| `inputData` | CommandActionInputDataDef | Yes | JSON data for the action input. |
| `outputData` | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |

### Referenced Objects

The following objects are referenced within this definition.


Unpublished work. © 2025 Siemens

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

## CopyDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `Copy`. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | ParametersObject | | |
| `outputData` | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `events` | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | actionMessagesDef | | User messages. It can be made condition based |
| `dataParsers` | dataParsersDef | | |

#### Referenced Objects

The following objects are referenced within this definition.

#### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

#### ParametersObject

*No description specified.*

## dataParsersDef Definition (Action)

This value is an array of DataParsersDefObject objects.

#### Referenced Objects

The following objects are referenced within this definition.

#### DataParsersDefObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | |
| `responseObjs` | string | Yes | |

## dataProviderActionDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `dataProvider`. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `methods` | array | | Used with actionType 'dataProvider'. Array of dataProviders that should be invoked as part of this action. |
| `inputData` | **InputDataObject** | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `outputData` | **OutputDataObject** | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `events` | **actionMessagesDef** | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | **actionMessagesDef** | | User messages. It can be made condition based |
| `dataParsers` | **dataParsersDef** | | |

#### Referenced Objects

The following objects are referenced within this definition.

#### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

## dialogDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `dialog`. |

| | | | |
|---|---|---|---|
| `inputData` | **InputDataObject** | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | **ParametersObject** | | |
| `outputData` | **OutputDataObject** | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `events` | **actionMessagesDef** | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | **actionMessagesDef** | | User messages. It can be made condition based |
| `dataParsers` | **dataParsersDef** | | |
| `options` | **OptionsObject** | | |
| `method` | string | | Used with actionType 'dialog'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |

### Referenced Objects

The following objects are referenced within this definition.

### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

### OptionsObject

*No description specified.*

### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.


Unpublished work. © 2025 Siemens

**ParametersObject**

*No description specified.*

## EditDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `Edit`. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `events` | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | actionMessagesDef | | User messages. It can be made condition based |
| `dataParsers` | dataParsersDef | | |

## EventDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `Event`. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |

| | | | |
|---|---|---|---|
| parameters | ParametersObject | | |
| deps | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| policy | PolicyObject | | Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA |
| outputData | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| events | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| actionMessages | actionMessagesDef | | User messages. It can be made condition based |
| dataParsers | dataParsersDef | | |

### Referenced Objects

The following objects are referenced within this definition.

### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

### ParametersObject

*No description specified.*

### PolicyObject

Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA

This object can contain the following properties:


Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | |
| `properties` | policyPropertiesDef | Yes | |

## eventTypeDef Definition (Action)

An event object can have event name and eventData OR a message and a condition statement which would evaluate to boolean

This definition must satisfy *one* of the following constraints:

- it must be set to a ArrayEventTypeDefObject1 value
- it must be set to a ArrayEventTypeDefObject2 value

### Referenced Objects

The following objects are referenced within this definition.

#### ConditionObject

A condition statement which would evaluate to boolean.

#### EventDataObject

Event data is a JSON object for data to provide on the event. The structure of the eventData is specific to a given event type.

#### EventTypeDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | | A condition statement which would evaluate to boolean. |
| `message` | string | Yes | The message string that would be passed with event, if above condition evaluates to true. |

### EventTypeDefObject2

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | Name of the event. The name is used by interested listeners. |
| `eventData` | EventDataObject | | Event data is a JSON object for data to provide on the event. The structure of the eventData is specific to a given event type. |
| `condition` | ConditionObject | | A condition statement which would evaluate to boolean. |
| `excludeLocalDataCtx` | boolean | | Excludes the local ctx from event data |

## GraphQLDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `GraphQL`. |
| `serviceName` | string | | Used with actionType 'TcSoaService'. Name of the SOA service that should be invoked. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | GraphQLInputDataDef | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | ParametersObject | | |
| `events` | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `outputData` | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `actionMessages` | actionMessagesDef | | User messages. It can be made condition based |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| `dataParsers` | dataParsersDef | | |

#### Referenced Objects

The following objects are referenced within this definition.

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

#### ParametersObject

*No description specified.*

## GraphQLInputDataDef Definition (Action)

JSON data for GraphQL input, which needs endPoint and request.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `endPoint` | string | | |
| `request` | RequestObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |

#### Referenced Objects

The following objects are referenced within this definition.

### RequestObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

## JSFunctionAsyncDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `JSFunctionAsync`. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | ParametersObject | | |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| `policy` | PolicyObject | | Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA |
| `outputData` | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `events` | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | actionMessagesDef | | User messages. It can be made condition based |
| `dataParsers` | dataParsersDef | | |

### Referenced Objects

The following objects are referenced within this definition.


Unpublished work. © 2025 Siemens

#### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

#### ParametersObject

*No description specified.*

#### PolicyObject

Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | |
| `properties` | policyPropertiesDef | Yes | |

## JSFunctionDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `JSFunction`. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |

| | | | |
|---|---|---|---|
| parameters | ParametersObject | | |
| deps | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| policy | PolicyObject | | Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA |
| outputData | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| events | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| actionMessages | actionMessagesDef | | User messages. It can be made condition based |
| dataParsers | dataParsersDef | | |

### Referenced Objects

The following objects are referenced within this definition.

### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

### ParametersObject

*No description specified.*

### PolicyObject

Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA

This object can contain the following properties:


Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| name | string | Yes | |
| properties | policyPropertiesDef | Yes | |

## NavigateDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| actionType | string | Yes | It must be set to Navigate. |
| navigationParams | NavigationParamsObject | | Used with actionType 'Navigate'. Specifies the parameters to be passed while navigating to the state |
| navigateIn | string | | Used with actionType 'Navigate'. Specifies if the new url should be opened in 'newTab' or 'newWindow' |
| navigateTo | string | | Used with actionType 'Navigate'. Specifies the state to navigate to |
| options | OptionsObject | | Used with actionType 'Navigate'. Can be used to specify the 'top', 'left', 'height', 'width' for the 'newWindow' |
| outputData | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| events | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| actionMessages | actionMessagesDef | | User messages. It can be made condition based |
| dataParsers | dataParsersDef | | |

### Referenced Objects

The following objects are referenced within this definition.

#### NavigationParamsObject

Used with actionType 'Navigate'. Specifies the parameters to be passed while navigating to the state

#### OptionsObject

Used with actionType 'Navigate'. Can be used to specify the 'top', 'left', 'height', 'width' for the 'newWindow'

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

## policyPropertiesDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | |
| `modifiers` | array | | |

## popupDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `popup`. |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | ParametersObject | | |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `outputData` | **OutputDataObject** | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `events` | **actionMessagesDef** | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | **actionMessagesDef** | | User messages. It can be made condition based |
| `dataParsers` | **dataParsersDef** | | |
| `options` | **OptionsObject** | | |
| `method` | string | | Used with actionType 'popup'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |

### Referenced Objects

The following objects are referenced within this definition.

### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

### OptionsObject

*No description specified.*

### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

### ParametersObject

*No description specified.*

## popupInputDataDef Definition (Action)

Actions can send array of events/actionMessages in case of success and failure of the action.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `options` | OptionsObject | | Options data |

### Referenced Objects

The following objects are referenced within this definition.

#### OptionsObject

Options data

## RESTServiceDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `RESTService`. |
| `serviceName` | string | | Used with actionType 'TcSoaService'. Name of the SOA service that should be invoked. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | ParametersObject | | |
| `events` | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `outputData` | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `actionMessages` | actionMessagesDef | | User messages. It can be made condition based |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| `dataParsers` | dataParsersDef | | |

#### Referenced Objects

The following objects are referenced within this definition.

#### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

#### ParametersObject

*No description specified.*

## SyncDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to Sync. |
| `inputData` | SyncInputDataDef | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |

| | | | |
|---|---|---|---|
| events | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| outputData | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| actionMessages | actionMessagesDef | | User messages. It can be made condition based |
| parameters | ParametersObject | | |
| deps | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| dataParsers | dataParsersDef | | |

#### Referenced Objects

The following objects are referenced within this definition.

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

#### ParametersObject

*No description specified.*

## SyncInputDataDef Definition (Action)

JSON data for GraphQL input, which needs endPoint and request.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| port | string | Yes | |
| syncObject | SyncObjectObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Referenced Objects

The following objects are referenced within this definition.

### SyncObjectObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

## TcSoaServiceDef Definition (Action)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | It must be set to `TcSoaService`. |
| `serviceName` | string | | Used with actionType 'TcSoaService'. Name of the SOA service that should be invoked. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |
| `parameters` | ParametersObject | | |
| `headerState` | HeaderStateObject | | |
| `policy` | PolicyObject | | Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| `outputData` | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| `events` | actionMessagesDef | | Events can be fired and listened/handled in declarative View Models. They are a means of |

| | | | |
|---|---|---|---|
| | | | a View Model communicating with other View Models or UI elements. 'Actions' can fire events. |
| `actionMessages` | **actionMessagesDef** | | User messages. It can be made condition based |
| `dataParsers` | **dataParsersDef** | | |

### Referenced Objects

The following objects are referenced within this definition.

### HeaderStateObject

*No description specified.*

### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

### ParametersObject

*No description specified.*

### PolicyObject

Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | |
| `properties` | **policyPropertiesDef** | Yes | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

# Adapters Schema

This schema is an array of AdaptersObject objects.

## Definitions

This schema includes the following definitions:

- expressionStringOrObjectDef

## Referenced Objects

The following objects are referenced within this schema.

## AdaptersObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description | |
|---|---|---|---|---|
| conditions | ConditionsAltObject | Yes | Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |
| target | TargetObject | Yes | This section would be used to compute the target object from the source object | |

## ConditionsAltObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

This value must satisfy *one* of the following constraints:

- it must be set to a ConditionsObject1 value
- it must be set to a expressionStringOrObjectDef value

### ConditionsObject1

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_.]*$` | expressionStringOrObjectDef | | |

### TargetObject

This section would be used to compute the target object from the source object

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `prop` | string | Yes | Name of the ViewModelProperty. This property would be used to either compute or return the target object. if 'method' is not provided this prop name would be considered as the adopted object |
| `method` | string | | This is a function name. This should be used when consumer need some custom logic to compute the adopted object |
| `deps` | string | | The file name where the method is defined. |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeAdaptersSchema-1.0.0.json",
  "description": "This schema is used to validate adapters.json files that
are used to find the adapted object for a given model object.",
  "definitions": {
    "expressionStringOrObjectDef": {
      "type": "object",
      "description": "Expression can be defined as object with the support
of below operators.",
      "additionalProperties": false,
      "properties": {
        "$query": {
          "type": "object",
          "description": "Defines the query to be executed on the source"
        },
        "$and": {
          "type": "array",
          "description": "Indicates logical 'AND' of each query result",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "items": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_.]*$": {
                "$ref": "#/definitions/expressionStringOrObjectDef"
              }
            }
          }
        },
        "$or": {
          "type": "array",
          "description": "Indicates logical 'OR' of each query result"
        },
        "$all": {
          "type": "array",
          "description": "Indicates logical 'ALL' of each query result"
        },
        "$eventSource": {
          "type": "string",
          "description": "This operator lets user specify the viewId or
viewName or 'self' to write a condition to enable the events to be filtered
out based on whether or not it is triggered from the same viewModel."
        },
        "$not": {
          "type": "object",
          "description": "Indicates logical 'NOT' of a query result"
        },
        "$in": {
          "type": [
            "array",
            "string"
          ],
          "description": "Indicates logical 'IN'. The IN operator allows
multiple values to be tested against the expression and thus reduces the
use of multiple OR conditions with each test value."
        },
        "$notin": {
          "type": "string",
          "description": "Indicates logical 'NOT IN'. It is the exact
opposite of the IN operator."
        }
      }
    }
  },
  "type": "array",
  "items": {
    "type": "object",
    "additionalProperties": false,
    "propertyNames": {
      "enum": [
        "conditions",
        "target"
      ]
    },
    "properties": {
```

```
      "conditions": {
        "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and can
be used for event handling (onEvent handler). The data value can be live
evaluated by condition. They can leverage Boolean condition expressions
like '&&', '||', '==', '!='",
        "oneOf": [
          {
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_.]*$": {
                "$ref": "#/definitions/expressionStringOrObjectDef"
              }
            }
          },
          {
            "additionalProperties": false,
            "$ref": "#/definitions/expressionStringOrObjectDef"
          }
        ]
      },
      "target": {
        "description": "This section would be used to compute the target
object from the source object",
        "type": "object",
        "propertyNames": {
          "enum": [
            "prop",
            "method",
            "deps"
          ]
        },
        "properties": {
          "prop": {
            "description": "Name of the ViewModelProperty. This property
would be used to either compute or return the target object. if 'method' is
not provided this prop name would be considered as the adopted object",
            "type": "string"
          },
          "method": {
            "description": "This is a function name. This should be used
when consumer need some custom logic to compute the adopted object",
            "type": "string"
          },
          "deps": {
            "description": "The file name where the method is defined.",
            "type": "string"
          }
        },
        "required": [
          "prop"
        ]
      }
    },
    "required": [
      "conditions",
```


Unpublished work. © 2025 Siemens

```
      "target"
    ]
  },
  "__defs": []
}
```

## expressionStringOrObjectDef Definition (Adapters)

Expression can be defined as object with the support of below operators.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$query` | QueryObject | | Defines the query to be executed on the source |
| `$and` | ArrayAndObject | | Indicates logical 'AND' of each query result |
| `$or` | array | | Indicates logical 'OR' of each query result |
| `$all` | array | | Indicates logical 'ALL' of each query result |
| `$eventSource` | string | | This operator lets user specify the viewId or viewName or 'self' to write a condition to enable the events to be filtered out based on whether or not it is triggered from the same viewModel. |
| `$not` | NotObject | | Indicates logical 'NOT' of a query result |
| `$in` | array OR string | | Indicates logical 'IN'. The IN operator allows multiple values to be tested against the expression and thus reduces the use of multiple OR conditions with each test value. |
| `$notin` | string | | Indicates logical 'NOT IN'. It is the exact opposite of the IN operator. |

### Referenced Objects

The following objects are referenced within this definition.

#### AndObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_.]*$ | expressionStringOrObjectDef | | |

#### NotObject

Indicates logical 'NOT' of a query result

#### QueryObject

Defines the query to be executed on the source

## AliasRegistry Schema

This schema provides a way to configure which type icon to display for a list of business object types.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z][a-zA-Z0-9_.]*$ | Arraystring | | Full svg file name without extension, to be used by business object mentioned in property |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeAliasRegistrySchema-1.0.0.json",
  "description": "This schema provides a way to configure which type icon
to display for a list of business object types.",
  "type": "object",
  "additionalProperties": false,
  "patternProperties": {
    "^[a-zA-Z][a-zA-Z0-9_.]*$": {
      "description": "Full svg file name without extension, to be used by
business object mentioned in property",
      "type": "array",
      "items": {
        "description": "List of object types",
        "type": "string"
      }
    }
  },
  "__defs": []
}
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

# BatchJobViewModel Schema

This is the schema the declarative View Model (in JSON format) must adhere to. The Apollo audit check validates all View Model JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actions` | ActionsObject | | In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService ). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button. |

## Definitions

This schema includes the following definitions:

- actionDef
- actionMessagesDef
- dataParsersDef
- eventTypeDef
- policyPropertiesDef

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService ). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| \b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b | actionDef | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeBatchJobViewModelSchema-1.0.0.json",
  "description": "This is the schema the declarative View Model (in JSON
format) must adhere to. The Apollo audit check validates all View Model
JSON files against this schema. For explanation of the keywords of this
schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "eventTypeDef": {
      "description": "An event object can have event name and eventData OR
a message and a condition statement which would evaluate to boolean",
      "oneOf": [
        {
          "type": "array",
          "items": {
            "required": [
              "message"
            ],
            "type": "object",
            "properties": {
              "condition": {
                "type": "string",
                "description": "A condition statement which would evaluate
to boolean."
              },
              "message": {
                "type": "string",
                "description": "The message string that would be passed
with event, if above condition evaluates to true."
              }
            }
          }
        },
        {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "name": {
                "type": "string",
                "description": "Name of the event. The name is used by
interested listeners."
              },
              "eventData": {
                "type": "object",
                "description": "Event data is a JSON object for data to
provide on the event. The structure of the eventData is specific to a given
event type."
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
              }
            },
            "required": [
              "name"
            ]
          }
        }
      ]
    },
    "actionMessagesDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Actions can send array of events/actionMessages in
case of success and failure of the action.",
      "properties": {
        "success": {
          "$ref": "#/definitions/eventTypeDef"
        },
        "failure": {
          "$ref": "#/definitions/eventTypeDef"
        }
      }
    },
    "dataParsersDef": {
      "type": "array",
      "description": "In the View Model, an action can have an optional
'dataParsers' construct. It is an array of objects, each having an 'id:
specifying which dataParseDefinition to use' & a 'responseObjs: path to
the response object(s) to convert.'. The 'responseObjs' can be an array
of objects, or a single one. properties below. The dataParsers accepts
the response data and converts them to 'afx objects' each containing
ViewModelProperties. These afx objects are directly consumable by the
widgets provided by afx.",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "id": {
            "type": "string"
          },
          "responseObjs": {
            "type": "string"
          }
        },
        "required": [
          "id",
          "responseObjs"
        ]
      }
    },
    "actionDef": {
      "type": "object",
      "description": "Actions can be of different actionType (REST,
JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
```

```
      "additionalProperties": false,
      "required": [
        "actionType"
      ],
      "properties": {
        "actionType": {
          "type": "string",
          "description": "This is a mandatory property in action object.
The allowed values are - 'TcSoaService', dataProvider', 'JSFunction',
'Event', 'RESTService', 'JSFunctionAsync', 'Copy', 'Navigate', 'popup'",
          "not": {
            "const": "batchJob"
          }
        },
        "serviceName": {
          "type": "string",
          "description": "Used with actionType 'TcSoaService'. Name of the
SOA service that should be invoked."
        },
        "method": {
          "type": "string",
          "description": "Used with actionType 'JSFunction' or
'JSFunctionAsync'. Points to the method name that should be invoked from
the js file specified in 'deps'"
        },
        "methods": {
          "type": "array",
          "description": "Used with actionType 'dataProvider'. Array of
dataProviders that should be invoked as part of this action."
        },
        "options": {
          "type": "object",
          "description": "Used with actionType 'Navigate'. Can be used to
specify the 'top', 'left', 'height', 'width' for the 'newWindow'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input. For ex. 'request'
header can be the inputData in case of 'RESTService' actionType with method
'Get'."
        },
        "parameters": {
          "type": "object"
        },
        "policy": {
          "type": "object",
          "description": "Used with actionType 'TcSoaService'. This is a
definition of property policy provided with SOA",
          "items": {
            "type": "array",
            "properties": {
              "name": {
                "type": "string"
              },
              "properties": {
                "$ref": "#/definitions/policyPropertiesDef"
```


Unpublished work. © 2025 Siemens

```
              }
            },
            "required": [
              "name",
              "properties"
            ]
          }
        },
        "outputData": {
          "type": "object",
          "description": "JSON data for the action output. As part of this,
the result of the action is mapped to view model data."
        },
        "actionMessages": {
          "$ref": "#/definitions/actionMessagesDef",
          "description": "User messages. It can be made condition based"
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        },
        "dataParsers": {
          "$ref": "#/definitions/dataParsersDef",
          "description": "Data parser accepts the response data and
converts them to 'afx objects' each containing ViewModelProperties."
        }
      }
    },
    "policyPropertiesDef": {
      "type": "object",
      "items": {
        "type": "array",
        "additionalProperties": false,
        "required": [
          "name"
        ],
        "properties": {
          "name": {
            "type": "string"
          },
          "modifiers": {
            "type": "array"
          }
        }
      }
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "actions": {
      "type": "object",
      "description": "In the View Model, actions are declaratively
defined. Actions can be of different 'actionType' (REST, JSFunction,
```

```
JSFunctionAsync, Event or TcSoaService ). All actions have 'inputData'
which specifies the JSON input for the action and 'outputData' for mapping
the output response. Actions can be used in dataProviders or UI elements
such as an aw-button.",
      "patternProperties": {
        "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
          "$ref": "#/definitions/actionDef"
        }
      }
    }
  },
  "__defs": []
}
```

## actionDef Definition (BatchJobViewModel)

Actions can be of different actionType (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have inputData which specifies the JSON input for the action and outputData for mapping the output response.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | Yes | This is a mandatory property in action object. The allowed values are - 'TcSoaService', dataProvider', 'JSFunction', 'Event', 'RESTService', 'JSFunctionAsync', 'Copy', 'Navigate', 'popup' It cannot be set to `batchJob`. |
| `serviceName` | string | | Used with actionType 'TcSoaService'. Name of the SOA service that should be invoked. |
| `method` | string | | Used with actionType 'JSFunction' or 'JSFunctionAsync'. Points to the method name that should be invoked from the js file specified in 'deps' |
| `methods` | array | | Used with actionType 'dataProvider'. Array of dataProviders that should be invoked as part of this action. |
| `options` | OptionsObject | | Used with actionType 'Navigate'. Can be used to specify the 'top', 'left', 'height', 'width' for the 'newWindow' |
| `inputData` | InputDataObject | | JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| parameters | ParametersObject | | |
| policy | PolicyObject | | Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA |
| outputData | OutputDataObject | | JSON data for the action output. As part of this, the result of the action is mapped to view model data. |
| actionMessages | actionMessagesDef | | User messages. It can be made condition based |
| deps | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |
| dataParsers | dataParsersDef | | Data parser accepts the response data and converts them to 'afx objects' each containing ViewModelProperties. |

### Referenced Objects

The following objects are referenced within this definition.

#### InputDataObject

JSON data for the action input. For ex. 'request' header can be the inputData in case of 'RESTService' actionType with method 'Get'.

#### OptionsObject

Used with actionType 'Navigate'. Can be used to specify the 'top', 'left', 'height', 'width' for the 'newWindow'

#### OutputDataObject

JSON data for the action output. As part of this, the result of the action is mapped to view model data.

#### ParametersObject

*No description specified.*

#### PolicyObject

Used with actionType 'TcSoaService'. This is a definition of property policy provided with SOA

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | |
| `properties` | policyPropertiesDef | Yes | |

## actionMessagesDef Definition (BatchJobViewModel)

Actions can send array of events/actionMessages in case of success and failure of the action.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `success` | eventTypeDef | | |
| `failure` | eventTypeDef | | |

## dataParsersDef Definition (BatchJobViewModel)

This value is an array of DataParsersDefObject objects.

### Referenced Objects

The following objects are referenced within this definition.

### DataParsersDefObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | |
| `responseObjs` | string | Yes | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## eventTypeDef Definition (BatchJobViewModel)

An event object can have event name and eventData OR a message and a condition statement which would evaluate to boolean

This definition must satisfy *one* of the following constraints:

- it must be set to a Array**EventTypeDefObject1** value
- it must be set to a Array**EventTypeDefObject2** value

### Referenced Objects

The following objects are referenced within this definition.

### EventDataObject

Event data is a JSON object for data to provide on the event. The structure of the eventData is specific to a given event type.

### EventTypeDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | | A condition statement which would evaluate to boolean. |
| `message` | string | Yes | The message string that would be passed with event, if above condition evaluates to true. |

### EventTypeDefObject2

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| name | string | Yes | Name of the event. The name is used by interested listeners. |
| eventData | EventDataObject | | Event data is a JSON object for data to provide on the event. The structure of the eventData is specific to a given event type. |

### policyPropertiesDef Definition (BatchJobViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| name | string | Yes | |
| modifiers | array | | |

## Build Schema

This is the schema the declarative build (in JSON format) must adhere to. The Apollo audit check validates all site JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| transpilers | TranspilersObject | | |
| jest | JestObject | | Jest unit test specific options |
| sonar | SonarObject | | SonarQube sources information |
| modules | ModulesObject | | Module overrides |
| srcPaths | Arraystring | | Array of the source paths. It must contain at least 1 unique item. |
| imageDir | string | | relative path to the image directory |
| customImages | Arraystring | | Array of the custom images |
| genSchema | string | | relative path to the generate schema tool |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `preBuildChecks` | PreBuildChecksAltObject | | Configuration to control pre build checks |
| `sourceMapExcludePaths` | Arraystring | | Array of RegExp patterns to exclude from source map generation. It must contain only unique items. |
| `watcherIgnorePaths` | Arraystring | | Array of glob-patterns to ignore from watcher. It must contain only unique items. |

## Definitions

This schema includes the following definitions:

- additionalModulesDef

## Referenced Objects

The following objects are referenced within this schema.

### JestObject

Jest unit test specific options

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `transformIgnorePatterns` | Arraystring | | Path patterns to ignore during babel transformations. It must contain at least 1 unique item. |

### ModulesObject

Module overrides

**Note**
This object *can* have additional properties.

### PreBuildChecksAltObject

Configuration to control pre build checks

This value must satisfy *one* of the following constraints:

- it must be set to a **PreBuildChecksObject1** value
- it must be set to a boolean value

### PreBuildChecksObject1

*No description specified*.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `logMemoryDetails` | boolean | | Flag to enable or disable logging memory details in the console |
| `checkHomeDirAccess` | boolean | | Flag to enable or disable Home directory access check |
| `checkOutDirAccess` | boolean | | Flag to enable or disable Out directory access check |

### SonarObject

SonarQube sources information

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `additionalModules` | Array**additionalModulesDef** | | The list of additional modules which should be considered for sonarQube analysis. It must contain at least 1 unique item. |

### TranspilersObject

*No description specified*.

You can define properties on this object with a name matching one of the following patterns:


Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | TranspilersPatternObject | | |

### TranspilersPatternObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| packageName | string | | Package name |
| outputDirectory | string | | Output directory name |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeBuildSchema-1.0.0.json",
  "description": "This is the schema the declarative build (in JSON format)
must adhere to. The Apollo audit check validates all site JSON files
against this schema. For explanation of the keywords of this schema, please
visit https://ajv.js.org/keywords.html",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "additionalModulesDef": {
      "type": "object",
      "additionalProperties": false,
      "description": "schema for additinalModules",
      "properties": {
        "name": {
          "type": "string",
          "description": "The name of the module"
        },
        "baseDir": {
          "type": "string",
          "description": "The base / project directory for module"
        },
        "sources": {
          "type": "string",
          "description": "Sources information needed to apply sonar rules"
        },
        "inclusions": {
          "type": "array",
          "description": "The source inclusion regex",
          "minItems": 1,
          "items": {
```

```
            "type": "string"
          }
        },
        "exclusions": {
          "type": "array",
          "description": "The source exclusions regex",
          "minItems": 1,
          "items": {
            "type": "string"
          }
        },
        "coverage": {
          "type": "object",
          "description": "Coverage information",
          "additionalProperties": false,
          "properties": {
            "exclusions": {
              "type": "array",
              "description": "File types to be excluded for coverage",
              "minItems": 1,
              "items": {
                "type": "string"
              }
            }
          }
        },
        "tests": {
          "type": "string",
          "description": " The test directory path, relative to cwd"
        }
      },
      "required": [
        "name",
        "baseDir"
      ]
    }
  },
  "properties": {
    "transpilers": {
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": "object",
          "properties": {
            "packageName": {
              "type": "string",
              "description": "Package name"
            },
            "outputDirectory": {
              "type": "string",
              "description": "Output directory name"
            }
          }
        }
      }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    },
    "jest": {
      "type": "object",
      "description": "Jest unit test specific options",
      "additionalProperties": false,
      "properties": {
        "transformIgnorePatterns": {
          "type": "array",
          "description": "Path patterns to ignore during babel
transformations",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "uniqueItems": true
        }
      }
    },
    "sonar": {
      "type": "object",
      "description": "SonarQube sources information",
      "additionalProperties": false,
      "properties": {
        "additionalModules": {
          "type": "array",
          "description": "The list of additional modules which should be
considered for sonarQube analysis",
          "items": {
            "type": "object",
            "$ref": "#/definitions/additionalModulesDef"
          },
          "minItems": 1,
          "uniqueItems": true
        }
      }
    },
    "modules": {
      "type": "object",
      "description": "Module overrides",
      "additionalProperties": true
    },
    "srcPaths": {
      "type": "array",
      "description": "Array of the source paths",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "imageDir": {
      "type": "string",
      "description": "relative path to the image directory"
    },
    "customImages": {
      "type": "array",
```

```
      "description": "Array of the custom images",
      "items": {
        "type": "string"
      }
    },
    "genSchema": {
      "type": "string",
      "description": "relative path to the generate schema tool"
    },
    "preBuildChecks": {
      "description": "Configuration to control pre build checks",
      "oneOf": [
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "logMemoryDetails": {
              "type": "boolean",
              "description": "Flag to enable or disable logging memory
details in the console"
            },
            "checkHomeDirAccess": {
              "type": "boolean",
              "description": "Flag to enable or disable Home directory
access check"
            },
            "checkOutDirAccess": {
              "type": "boolean",
              "description": "Flag to enable or disable Out directory
access check"
            }
          }
        },
        {
          "type": "boolean",
          "description": "Set it to true to enable all checks, and false
otherwise"
        }
      ]
    },
    "sourceMapExcludePaths": {
      "type": "array",
      "description": "Array of RegExp patterns to exclude from source map
generation",
      "items": {
        "type": "string"
      },
      "uniqueItems": true
    },
    "watcherIgnorePaths": {
      "type": "array",
      "description": "Array of glob-patterns to ignore from watcher",
      "items": {
        "type": "string"
      },
      "uniqueItems": true
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    }
  },
  "__defs": []
}
```

## additionalModulesDef Definition (Build)

schema for additinalModules

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | The name of the module |
| `baseDir` | string | Yes | The base / project directory for module |
| `sources` | string | | Sources information needed to apply sonar rules |
| `inclusions` | Arraystring | | The source inclusion regex. It must contain at least 1 item. |
| `exclusions` | Arraystring | | The source exclusions regex. It must contain at least 1 item. |
| `coverage` | CoverageObject | | Coverage information |
| `tests` | string | | The test directory path, relative to cwd |

### Referenced Objects

The following objects are referenced within this definition.

### CoverageObject

Coverage information

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `exclusions` | Arraystring | | File types to be excluded for coverage. It must contain at least 1 item. |

## CommandsViewModel Schema

This is the schema the declarative Commands View Model (in JSON format) must adhere to. The Apollo audit check validates all Commands View Model JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with |
| `actions` | **ActionsObject** | | In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button. |
| `data` | **DataAltObject** | | The View can refer to data and view state, defined in the View Model data section. The braces '{{ }}' allow declarative binding to the live data. |
| `preferences` | Arraystring | | undefined. It must contain only unique items. |
| `functions` | **functionsDef** | | |
| `messages` | **MessagesObject** | | Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | |
|---|---|---|---|---|
| `conditions` | **ConditionsObject** | | Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |
| `i18n` | **I18nObject** | | All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown. | |
| `onEvent` | Array**OnEventObject** | | The View Model can register to react to events by declaring an onEvent section. It can then conditionally execute a declarative action. | |
| `commands` | **CommandsObject** | | Decalarative commands provide a graphical means for the user to work with their data. You can define a Command including ID, Icon, Title (aka: tooltip), placement in UI, visibility condition and execution as either a REST invocation, JavaScript function or SOA invocation | |
| `commandHandlers` | **CommandHandlersObject** | | A command can have many Command Handlers. However at any given time, there may at most be only one active commandHandler for a command. A declarative | |

| | | | |
|---|---|---|---|
| | | | 'activeWhen' condition which 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 'specfic' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the command handler will determine command visiblity and What a command will do when executed. |
| `commandPlacements` | CommandPlacementsObject | | A command can have many placements. A placement is the actual visual rendering of the display of the command. |
| `ctx` | CtxObject | | |

## Definitions

This schema includes the following definitions:

- commandDef
- commandHandlersDef
- commandPlacementsDef

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button.

You can define properties on this object with a name matching one of the following patterns:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | Action | | |

## CommandHandlersObject

A command can have many Command Handlers. However at any given time, there may at most be only one active commandHandler for a command. A declarative 'activeWhen' condition which 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 'specfic' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the command handler will determine command visiblity and What a command will do when executed.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | commandHandlersDef | | |

## CommandPlacementsObject

A command can have many placements. A placement is the actual visual rendering of the display of the command.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | commandPlacementsDef | | |

## CommandsObject

Decalarative commands provide a graphical means for the user to work with their data. You can define a Command including ID, Icon, Title (aka: tooltip), placement in UI, visibility condition and execution as either a REST invocation, JavaScript function or SOA invocation

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | commandDef | | |

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | Condition | | |

## CriteriaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `panelId` | string | | |
| `destPanelId` | string | | |

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | number OR string OR boolean | | |

## CtxObject

*No description specified.*

## DataAltObject

The View can refer to data and view state, defined in the View Model data section. The braces '{{ }}' allow declarative binding to the live data.

This value must satisfy *any* of the following constraints:


Unpublished work. © 2025 Siemens

- it must be set to a **DataObject1** value
- it must be set to a **DataObject2** value

### DataObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `objects` | viewModelObjectDef | | |

### DataObject2

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | viewModelPropDef | | |

### EventDataObject

*No description specified.*

### I18nObject

All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | array OR string | | |


Unpublished work. © 2025 Siemens

## InputArgsObject

*No description specified.*

## MessagesObject

Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | messageDef | | |

## OnEventObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `eventId` | string | | The id of an event that the viewModel is interested to listen to. |
| `criteria` | CriteriaObject | | |
| `eventSource` | string | | (Optional) set it to 'current' to make event listeners only listen to events fired by the current view model |
| `action` | string | | An action that is defined in the actions section of the viewModel which would be executed only if the specified condition evaluates to true. |
| `message` | string | | |
| `inputArgs` | InputArgsObject | | |
| `cacheEventData` | boolean | | |
| `condition` | string | | |
| `eventData` | EventDataObject | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeCommandsViewModelSchema-1.0.0.json",
  "description": "This is the schema the declarative Commands View Model
(in JSON format) must adhere to. The Apollo audit check validates all
Commands View Model JSON files against this schema. For explanation of the
keywords of this schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "commandDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "iconId": {
          "description": "The id of the command's icon. It refers to the
SVG icon that is stored in the 'src/image' folder in the site location",
          "type": "string"
        },
        "hasDialog": {
          "description": "Indicates if command opens dialog. If action does
not open dialog immediately but subsequently then this flag is set to true",
          "type": "boolean"
        },
        "isGroup": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "Should evaluate to true if a group command, false
otherwise"
        },
        "title": {
          "oneOf": [
            {
              "description": "The title to display on the tooltip of the
command. It should be either an i18n key or as just a non-18n string. Even
though schema doesn't enforces it, title is required, which could be either
string or object, so that functionality can work properly",
              "type": "string"
            },
            {
              "description": "The title to display on the tooltip of the
command. It should be parametrized i18n key along with its params",
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "text": {
                  "type": "string"
                },
                "params": {
                  "type": "array",
                  "items": {
                    "type": "string"
                  }
                }
              }
```

```
            }
          ]
        },
        "description": {
          "oneOf": [
            {
              "description": "The description of the command to be
specified here. It should be either an i18n key or as just a non-18n
string. Even though schema doesn't enforces it, title is required, which
could be either string or object, so that functionality can work properly",
              "type": "string"
            },
            {
              "description": "The description of the command to be
specified here. It should be parametrized i18n key along with its params",
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "text": {
                  "type": "string"
                },
                "params": {
                  "type": "array",
                  "items": {
                    "type": "string"
                  }
                }
              }
            }
          ]
        },
        "extendedTooltip": {
          "oneOf": [
            {
              "description": "The advanced title to display on the tooltip
of the command. It will have the view as key where it will have the
template that can be shown in the tooltip box.",
              "type": "object",
              "properties": {
                "view": {
                  "type": "string"
                }
              }
            }
          ]
        },
        "template": {
          "oneOf": [
            {
              "type": "object",
              "description": "It denotes the template for the command.",
              "additionalProperties": false,
              "properties": {
                "text": {
                  "description": "The text to display for this template.",
                  "type": "string"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                },
                "visibleWhen": {
                  "description": "The name of the declarative condition
which will determine when the associated template isVisible.",
                  "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/conditionOrBoolDef"
                },
                "class": {
                  "description": "The css class to be used on this
template.",
                  "type": "string"
                }
              }
            },
            {
              "description": "It denotes the template for the command.",
              "type": "string"
            }
          ]
        },
        "isShuttle": {
          "type": "boolean",
          "description": "It provides a way to group related commands under
a single command. The icon and title of selected command from group is
reflected at the group level. It overrides isGroup value."
        },
        "isRibbon": {
          "type": "boolean",
          "description": "It provides a way to group related commands under
a single command. The visibility is independent of commands in the group.
It overrides isGroup and isShuttle value."
        },
        "displayType": {
          "type": "string",
          "description": "This can be set to menu (default) or
toolbarSection. Toolbar will render this command a collection of child
commands. Menu will behave as the default isGroup: true."
        },
        "isToggle": {
          "type": "boolean",
          "description": "It determine the toggle behavior of the specified
command."
        },
        "selected": {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "iconId": {
              "description": "The id of the command's icon. It refers to
the SVG icon that is stored in the 'src/image' folder in the site location",
              "type": "string"
            },
            "title": {
              "oneOf": [
                {
                  "description": "The title to display on the tooltip of
```

```
the command. It should be either an i18n key or as just a non-18n string.
Even though schema doesn't enforces it, title is required, which could be
either string or object, so that functionality can work properly",
                  "type": "string"
                },
                {
                  "description": "The title to display on the tooltip of
the command. It should be parametrized i18n key along with its params",
                  "type": "object",
                  "additionalProperties": false,
                  "properties": {
                    "text": {
                      "type": "string"
                    },
                    "params": {
                      "type": "array",
                      "items": {
                        "type": "string"
                      }
                    }
                  }
                }
              ]
            },
            "description": {
              "oneOf": [
                {
                  "description": "The description of the command to be
specified here. It should be either an i18n key or as just a non-18n
string. Even though schema doesn't enforces it, title is required, which
could be either string or object, so that functionality can work properly",
                  "type": "string"
                },
                {
                  "description": "The description of the command to be
specified here. It should be parametrized i18n key along with its params",
                  "type": "object",
                  "additionalProperties": false,
                  "properties": {
                    "text": {
                      "type": "string"
                    },
                    "params": {
                      "type": "array",
                      "items": {
                        "type": "string"
                      }
                    }
                  }
                }
              ]
            },
            "extendedTooltip": {
              "oneOf": [
                {
                  "description": "The advanced title to display on the
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
tooltip of the command. It will have the view as key where it will have
the template that can be shown in the tooltip box.",
                  "type": "object",
                  "additionalProperties": false,
                  "properties": {
                    "view": {
                      "type": "string"
                    }
                  }
                }
              ]
            }
          }
        },
        "scope": {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "condition": {
              "type": [
                "string"
              ],
              "description": "The name of the declarative condition which
will determine if the command is applicable for further processing of
command handlers like activeWhen/visibleWhen/enableWhen"
            }
          }
        },
        "shortcut": {
          "type": "string",
          "description": "The keyboard shortcut to be used for the command."
        },
        "showPopupOnSingleCommand": {
          "type": "boolean",
          "description": "The boolean value which determines popup should
be visible even if only single command is visible in group command."
        }
      },
      "required": []
    },
    "commandHandlersDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "description": "The command Id for which the handler is
associated.",
          "type": "string"
        },
        "action": {
          "description": "The name of the declarative action
(REST,SOA,Event or JSFunction) to be executed.",
          "type": "string"
        },
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
```

```
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when this command handler is active for the associated command."
        },
        "visibleWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when the associated command isVisible. This condition is only
evaluated if this command handler is active."
        },
        "selectWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine whether a command will be in selected or unselected state. This
condition is only evaluated if this command handler is active."
        },
        "enableWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine whether a command will be in enabled or disabled state. This
condition is only evaluated if this command handler is active."
        },
        "openWhen": {
          "type": "boolean",
          "description": "It determine the value openWhen property of the
specified command."
        }
      },
      "required": [
        "id",
        "activeWhen"
      ]
    },
    "commandPlacementsDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "description": "The command Id for which the placement is
associated.",
          "type": "string"
        },
        "parentGroupId": {
          "description": "It denotes the group the command belongs to.
Multiple command can have same 'parentGroupId'",
          "type": "string"
        },
        "uiAnchor": {
          "description": "A well known name for a given  element.  Anchors
could be created where needed by Solutions.",
          "type": "string"
        },
        "priority": {
```


Unpublished work. © 2025 Siemens

```
          "description": "The relative priority order to render commands in
ascending order unless the command bar has reverse alignment. Even though
schema doesn't enforces it, title is required so that functionality can
work properly.",
          "type": "number"
        },
        "relativeTo": {
          "description": "The command id of this command will be placed
relative to.  Priority as defined in the 'priority' property will be
applied relative to the 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.",
          "type": "string"
        },
        "showGroupSelected": {
          "type": "boolean",
          "description": "The boolean value which handle the default
selected state of the group command."
        },
        "showChevron": {
          "type": "boolean",
          "enum": [
            true,
            false
          ],
          "description": "If showChevron is false , it should hide group
command carrot"
        },
        "isOverflow": {
          "type": "boolean",
          "enum": [
            true,
            false
          ],
          "description": "If isOverflow is true , it should add more
commands if overflow is needed"
        },
        "cellDisplay": {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "hover": {
              "type": "boolean",
              "description": "The property which enable or disable hover
behavior."
            },
            "position": {
              "type": "string",
              "description": "It describes the cell display position."
            },
            "selected": {
              "type": "boolean",
              "description": "The boolean value which handle the default
```

```
selected state of the 'cellDisplay'."
            }
          }
        },
        "areas": {
          "type": "array",
          "items": {
            "type": "string",
            "description": "Name of the area to be specified in array"
          }
        },
        "isIcon": {
          "type": "boolean",
          "description": "The boolean value which determines isIcon
property of specified command."
        },
        "displayOption": {
          "enum": [
            "ON_HOVER",
            "ON_HOVER_AND_SELECTION",
            "ON_SELECTION",
            "ALWAYS"
          ],
          "description": "This is specific to aw-list cells, which helps
to decide the display state of cell commands. It describes the cell
display option should be one of the (ON_HOVER, ON_HOVER_AND_SELECTION,
ON_SELECTION, ALWAYS)."
        },
        "isSelected": {
          "type": "boolean",
          "description": "The boolean value which determines Whether the
command is currently selected."
        },
        "position": {
          "type": "string",
          "description": "It determines the position"
        },
        "style": {
          "enum": [
            "base",
            "primary"
          ],
          "description": "This determines command display style. 'base' and
'primary' are the supported display styles."
        },
        "showTitle": {
          "enum": [
            "always",
            "never"
          ],
          "description": "This determines whether or not to show command
label. 'always' and 'never' are the supported values. If not defined then
Command label will be based on global label setting"
        },
        "displayType": {
          "enum": [
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "menu",
            "toolbarSection",
            "splitButton"
          ],
          "description": "This can be set to menu (default),
toolbarSection, or splitButton. Requires isGroup: true on Command
definition. Toolbar will render this command a collection of child
commands. Split renders as split command. Menu is default behaviour."
        },
        "hideOnSingleChoice": {
          "type": "boolean",
          "description": "The boolean value which determines group/ shuttle
command should be visible if only single command is visible in it"
        }
      },
      "required": [
        "id"
      ]
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
    },
    "actions": {
      "type": "object",
      "description": "In the View Model, actions are declaratively
defined. Actions can be of different 'actionType' (REST, JSFunction,
JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which
specifies the JSON input for the action and 'outputData' for mapping the
output response. Actions can be used in dataProviders or UI elements such
as an aw-button.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeActionSchema-1.0.0.json"
        }
      }
    },
    "data": {
      "type": "object",
      "description": "The View can refer to data and view state, defined in
the View Model data section. The braces '{{  }}' allow declarative binding
to the live data.",
      "anyOf": [
        {
          "properties": {
            "objects": {
              "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/viewModelObjectDef"
```

```
            }
          }
        },
        {
          "type": "object",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/viewModelPropDef"
            }
          }
        }
      ]
    },
    "preferences": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "functions": {
      "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
functionsDef"
    },
    "messages": {
      "type": "object",
      "description": "Messages are declaratively defined in the View Model.
Messages are launched by events and actions. They cover information,
warning, error notifications. They give the user more deeper customization
options. They are localized. They can fire some actions. They can even
leverage view model data and conditions.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
messageDef"
        }
      }
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and
can be used for event handling (onEvent handler). The data value can
be live evaluated by condition. The conditions can refer to view model
state and can update view model state. They can leverage Boolean condition
expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    },
    "i18n": {
      "type": "object",
```


Unpublished work. © 2025 Siemens

```
      "description": "All user facing text should be localized unless
a web client solution supports only one language. The View refers to
localizations defined in the 'i18n' section of the View Model. You can
specify the actual string to be displayed (however, it would not be
localized) OR the name of JSON bundle which holds the localized text to
be shown.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": [
            "array",
            "string"
          ],
          "items": {
            "type": "string",
            "description": "Name of the i18n bundle file to be specified in
array"
          }
        }
      }
    },
    "onEvent": {
      "type": "array",
      "description": "The View Model can register to react to events
by declaring an onEvent section. It can then conditionally execute a
declarative action.",
      "required": [
        "eventId",
        "action"
      ],
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "eventId": {
            "type": "string",
            "description": "The id of an event that the viewModel is
interested to listen to."
          },
          "criteria": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
                "type": [
                  "number",
                  "string",
                  "boolean"
                ]
              }
            },
            "properties": {
              "panelId": {
                "type": "string"
              },
              "destPanelId": {
                "type": "string"
```

```
              }
            }
          },
          "eventSource": {
            "type": "string",
            "description": "(Optional) set it to 'current' to make event
listeners only listen to events fired by the current view model"
          },
          "action": {
            "type": "string",
            "description": "An action that is defined in the actions
section of the viewModel which would be executed only if the specified
condition evaluates to true."
          },
          "message": {
            "type": "string"
          },
          "inputArgs": {
            "type": "object"
          },
          "cacheEventData": {
            "type": "boolean"
          },
          "condition": {
            "type": "string"
          },
          "eventData": {
            "type": "object"
          }
        }
      }
    },
    "commands": {
      "type": "object",
      "description": "Decalarative commands provide a graphical means for
the user to work with their data. You can define a Command including
ID, Icon, Title (aka: tooltip), placement in UI, visibility condition
and  execution as either a REST invocation, JavaScript function or SOA
invocation",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/commandDef"
        }
      }
    },
    "commandHandlers": {
      "type": "object",
      "description": "A command can have many Command Handlers.  However
at any given time, there may at most be only one active commandHandler
for a command.  A declarative 'activeWhen' condition which 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
'specfic' condition (i.e. the condition with longer expression) is chosen
to be active.  By being active, the command handler will determine command
visiblity and What a command will do when executed.",
      "patternProperties": {
```


Unpublished work. © 2025 Siemens

```
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/commandHandlersDef"
        }
      }
    },
    "commandPlacements": {
      "type": "object",
      "description": "A command can have many placements. A placement is
the actual visual rendering of the display of the command.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/commandPlacementsDef"
        }
      }
    },
    "ctx": {
      "type": "object"
    }
  },
  "__defs": []
}
```

## commandDef Definition (CommandsViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `iconId` | string | | The id of the command's icon. It refers to the SVG icon that is stored in the 'src/image' folder in the site location |
| `hasDialog` | boolean | | Indicates if command opens dialog. If action does not open dialog immediately but subsequently then this flag is set to true |
| `isGroup` | conditionOrBoolDef | | Should evaluate to true if a group command, false otherwise |
| `title` | TitleAltObject | | |
| `description` | DescriptionAltObject | | |
| `extendedTooltip` | ExtendedTooltipAltObject | | |

| | | | |
|---|---|---|---|
| `template` | **TemplateAltObject** | | |
| `isShuttle` | boolean | | It provides a way to group related commands under a single command. The icon and title of selected command from group is reflected at the group level. It overrides isGroup value. |
| `isRibbon` | boolean | | It provides a way to group related commands under a single command. The visibility is independent of commands in the group. It overrides isGroup and isShuttle value. |
| `displayType` | string | | This can be set to menu (default) or toolbarSection. Toolbar will render this command a collection of child commands. Menu will behave as the default isGroup: true. |
| `isToggle` | boolean | | It determine the toggle behavior of the specified command. |
| `selected` | **SelectedObject** | | |
| `scope` | **ScopeObject** | | |
| `shortcut` | string | | The keyboard shortcut to be used for the command. |
| `showPopupOnSingleCommand` | boolean | | The boolean value which determines popup should be visible even if only single command is visible in group command. |

### Referenced Objects

The following objects are referenced within this definition.

### DescriptionAltObject

*No description specified.*


Siemens Web Framework
Unpublished work. © 2025 Siemens

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **DescriptionObject2** value

### DescriptionObject1

The description of the command to be specified here. It should be parametrized i18n key along with its params

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | |
| `params` | Arraystring | | |

### DescriptionObject2

The description of the command to be specified here. It should be parametrized i18n key along with its params

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | |
| `params` | Arraystring | | |

### ExtendedTooltipAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a **ExtendedTooltipObject2** value

### ExtendedTooltipObject1

The advanced title to display on the tooltip of the command. It will have the view as key where it will have the template that can be shown in the tooltip box.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `view` | string | | |

### ExtendedTooltipObject2

The advanced title to display on the tooltip of the command. It will have the view as key where it will have the template that can be shown in the tooltip box.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `view` | string | | |

### ScopeObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | | The name of the declarative condition which will determine if the command is applicable for further processing of command handlers like activeWhen/visibleWhen/enableWhen |

### SelectedObject

*No description specified.*

This object can contain the following properties:


Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| `iconId` | string | | The id of the command's icon. It refers to the SVG icon that is stored in the 'src/image' folder in the site location |
| `title` | TitleAltObject | | |
| `description` | DescriptionAltObject | | |
| `extendedTooltip` | ExtendedTooltipAltObject | | |

### TemplateAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a TemplateObject1 value
- it must be set to a string value

### TemplateObject1

It denotes the template for the command.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | The text to display for this template. |
| `visibleWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine when the associated template isVisible. |
| `class` | string | | The css class to be used on this template. |

### TitleAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a TitleObject2 value

#### TitleObject1

The title to display on the tooltip of the command. It should be parametrized i18n key along with its params

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | |
| `params` | Arraystring | | |

#### TitleObject2

The title to display on the tooltip of the command. It should be parametrized i18n key along with its params

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | |
| `params` | Arraystring | | |

## commandHandlersDef Definition (CommandsViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | The command Id for which the handler is associated. |
| `action` | string | | The name of the declarative action (REST,SOA,Event or JSFunction) to be executed. |
| `activeWhen` | conditionOrBoolDef | Yes | The name of the declarative condition which will determine when this command handler is active for the associated command. |
| `visibleWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine when the associated command isVisible. |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | This condition is only evaluated if this command handler is active. |
| `selectWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine whether a command will be in selected or unselected state. This condition is only evaluated if this command handler is active. |
| `enableWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine whether a command will be in enabled or disabled state. This condition is only evaluated if this command handler is active. |
| `openWhen` | boolean | | It determine the value openWhen property of the specified command. |

## commandPlacementsDef Definition (CommandsViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | The command Id for which the placement is associated. |
| `parentGroupId` | string | | It denotes the group the command belongs to. Multiple command can have same 'parentGroupId' |
| `uiAnchor` | string | | A well known name for a given element. Anchors could be created where needed by Solutions. |
| `priority` | number | | The relative priority order to render commands in ascending order unless the command bar has reverse alignment. Even though schema doesn't enforces it, title is required so that functionality can work properly. |
| `relativeTo` | string | | The command id of this command will be placed relative to. Priority as defined in the 'priority' property will be applied relative to the 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. |
| `showGroupSelected` | boolean | | The boolean value which handle the default selected state of the group command. |
| `showChevron` | boolean | | It can be set to one of the following values: `true`, `false`. If showChevron is false , it should hide group command carrot |
| `isOverflow` | boolean | | It can be set to one of the following values: `true`, `false`. If isOverflow is true , it should add more commands if overflow is needed |
| `cellDisplay` | **CellDisplayObject** | | |
| `areas` | Arraystring | | |
| `isIcon` | boolean | | The boolean value which determines isIcon property of specified command. |
| `displayOption` | | | It can be set to one of the following values: `ON_HOVER`, `ON_HOVER_AND_SELECTION`, `ON_SELECTION`, `ALWAYS`. This is specific to aw-list cells, which helps to decide the display state of cell commands. It describes the cell display option should be one of the (ON_HOVER, ON_HOVER_AND_SELECTION, ON_SELECTION, ALWAYS). |
| `isSelected` | boolean | | The boolean value which determines Whether the command is currently selected. |
| `position` | string | | It determines the position |
| `style` | | | It can be set to one of the following values: `base`, `primary`. This determines command display style. 'base' and 'primary' are the supported display styles. |
| `showTitle` | | | It can be set to one of the following values: `always`, `never`. This determines whether or not to show command label. 'always' and 'never' are the supported values. If not defined then Command label will be based on global label setting |
| `displayType` | | | It can be set to one of the following values: `menu`, `toolbarSection`, `splitButton`. This can be set to menu (default), toolbarSection, or splitButton. Requires |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | isGroup: true on Command definition. Toolbar will render this command a collection of child commands. Split renders as split command. Menu is default behaviour. |
| `hideOnSingleChoice` | boolean | | The boolean value which determines group/ shuttle command should be visible if only single command is visible in it |

#### Referenced Objects

The following objects are referenced within this definition.

#### CellDisplayObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `hover` | boolean | | The property which enable or disable hover behavior. |
| `position` | string | | It describes the cell display position. |
| `selected` | boolean | | The boolean value which handle the default selected state of the 'cellDisplay'. |

## Condition Schema

Expressions can be defined as simple strings or object.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `expression` | expressionStringOrObjectDef | Yes | |
| `trackValues` | boolean | | |

### Definitions

This schema includes the following definitions:

- expressionStringOrObjectDef

- sourceStringOrObjectDef

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeConditionSchema-1.0.0.json",
  "description": "Expressions can be defined as simple strings or object.",
  "definitions": {
    "sourceStringOrObjectDef": {
      "oneOf": [
        {
          "type": "string"
        },
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "$adapt": {
              "type": "string",
              "description": "Indicates that the resulting source should be
adapted before evaluating the query"
            }
          }
        }
      ]
    },
    "expressionStringOrObjectDef": {
      "oneOf": [
        {
          "type": "object",
          "description": "Expression can be defined as object with the
support of below operators.",
          "additionalProperties": false,
          "properties": {
            "$source": {
              "$ref": "#/definitions/sourceStringOrObjectDef",
              "description": "Indicates the reference on data context node
to be used as starting point for evaluation. For ex. 'ctx.selected'"
            },
            "$query": {
              "type": "object",
              "description": "Defines the query to be executed on the
source"
            },
            "$and": {
              "type": "array",
              "description": "Indicates logical 'AND' of each query result"
            },
            "$or": {
              "type": "array",
              "description": "Indicates logical 'OR' of each query result"
            },
            "$all": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
              "type": "array",
              "description": "Indicates logical 'ALL' of each query result"
            },
            "$eventSource": {
              "type": "string",
              "description": "This operator lets user specify the viewId or
viewName or 'self' to write a condition to enable the events to be filtered
out based on whether or not it is triggered from the same viewModel."
            },
            "$not": {
              "type": "object",
              "description": "Indicates logical 'NOT' of a query result"
            }
          }
        },
        {
          "type": "string",
          "description": "Expression as simple string: 'expression':
'ctx.mselected && ctx.mselected.length > 1'"
        }
      ]
    }
  },
  "additionalProperties": false,
  "type": "object",
  "properties": {
    "expression": {
      "$ref": "#/definitions/expressionStringOrObjectDef"
    },
    "trackValues": {
      "type": "boolean"
    }
  },
  "required": [
    "expression"
  ],
  "__defs": []
}
```

## expressionStringOrObjectDef Definition (Condition)

This definition must satisfy *one* of the following constraints:

- it must be set to a **ExpressionStringOrObjectDefObject1** value
- it must be set to a string value

### Referenced Objects

The following objects are referenced within this definition.

### ExpressionStringOrObjectDefObject1

Expression can be defined as object with the support of below operators.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$source` | sourceStringOrObjectDef | | Indicates the reference on data context node to be used as starting point for evaluation. For ex. 'ctx.selected' |
| `$query` | QueryObject | | Defines the query to be executed on the source |
| `$and` | array | | Indicates logical 'AND' of each query result |
| `$or` | array | | Indicates logical 'OR' of each query result |
| `$all` | array | | Indicates logical 'ALL' of each query result |
| `$eventSource` | string | | This operator lets user specify the viewId or viewName or 'self' to write a condition to enable the events to be filtered out based on whether or not it is triggered from the same viewModel. |
| `$not` | NotObject | | Indicates logical 'NOT' of a query result |

### NotObject

Indicates logical 'NOT' of a query result

### QueryObject

Defines the query to be executed on the source

## sourceStringOrObjectDef Definition (Condition)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a SourceStringOrObjectDefObject1 value


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Referenced Objects

The following objects are referenced within this definition.

#### SourceStringOrObjectDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$adapt` | string | | Indicates that the resulting source should be adapted before evaluating the query |

## Decorators Schema

This schema is used to validate decorators.json files that are used to set color decorators on a view model object.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_-]+[a-zA-Z0-9_-]*$` | DecoratorsPatternObject | | This section contained decorator objects to be evaluated and applied. |

### Definitions

This schema includes the following definitions:

- conditionDef
- expressionStringOrObjectDef
- sourceStringOrObjectDef

### Referenced Objects

The following objects are referenced within this schema.

## ConditionsObject

Conditions for when the decorator should be applied. Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | conditionDef | | |

## DecoratorsPatternObject

This section contained decorator objects to be evaluated and applied.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `cellClassName` | string | Yes | This is a css class added to list cell decorator when decorator is active. |
| `gridClassName` | string | | This is a css class added to color decorator element on table row when decorator is active. |
| `subLocationName` | string | | This property ensures the processing/ potentially application of this decorator when in this sublocation. |
| `title` | string | | This is the title (tooltip) to set on the decorator element. This also supports i18n. |
| `method` | string | | Function name to call with the vmo to determine if decorator should be applied. Function should return a boolean. |
| `deps` | string | | The js dependency file where the method resides in. |
| `modelTypes` | array | | Array of objects for which the color decorators should be valid. |
| `conditions` | ConditionsObject | | Conditions for when the decorator should be applied. Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses ', '==', '!=' |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | | |
|---|---|---|---|---|
| | | | and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. They can leverage Boolean condition expressions like '&&', ' | |
| i18n | I18nObject | | All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown. | |

## I18nObject

All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown.

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeDecoratorsSchema-1.0.0.json",
  "description": "This schema is used to validate decorators.json files
that are used to set color decorators on a view model object.",
  "definitions": {
    "sourceStringOrObjectDef": {
      "oneOf": [
        {
          "type": "string"
        },
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "$adapt": {
              "type": "string",
              "description": "Indicates that the resulting source should be
adapted before evaluating the query"
            }
          }
        }
      ]
    },
    "expressionStringOrObjectDef": {
      "oneOf": [
        {
```

```
          "type": "object",
          "description": "Expression can be defined as object with the
support of below operators.",
          "additionalProperties": false,
          "properties": {
            "$source": {
              "$ref": "#/definitions/sourceStringOrObjectDef",
              "description": "Indicates the reference on data context node
to be used as starting point for evaluation. For ex. 'ctx.selected'"
            },
            "$query": {
              "type": "object",
              "description": "Defines the query to be executed on the
source"
            },
            "$and": {
              "type": "array",
              "description": "Indicates logical 'AND' of each query result"
            },
            "$or": {
              "type": "array",
              "description": "Indicates logical 'OR' of each query result"
            },
            "$all": {
              "type": "array",
              "description": "Indicates logical 'ALL' of each query result"
            },
            "$eventSource": {
              "type": "string",
              "description": "This operator lets user specify the viewId or
viewName or 'self' to write a condition to enable the events to be filtered
out based on whether or not it is triggered from the same viewModel."
            },
            "$not": {
              "type": "object",
              "description": "Indicates logical 'NOT' of a query result"
            }
          }
        },
        {
          "type": "string",
          "description": "Expression as simple string: 'expression':
'ctx.mselected && ctx.mselected.length > 1'"
        }
      ]
    },
    "conditionDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Expressions can be defined as simple strings or
object.",
      "properties": {
        "expression": {
          "$ref": "#/definitions/expressionStringOrObjectDef"
        }
      },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "required": [
        "expression"
      ]
    }
  },
  "type": "object",
  "minProperties": 1,
  "patternProperties": {
    "^[a-zA-Z_-]+[a-zA-Z0-9_-]*$": {
      "description": "This section contained decorator objects to be
evaluated and applied.",
      "type": "object",
      "propertyNames": {
        "enum": [
          "cellClassName",
          "gridClassName",
          "subLocationName",
          "title",
          "method",
          "deps",
          "modelTypes",
          "conditions",
          "i18n"
        ]
      },
      "properties": {
        "cellClassName": {
          "description": "This is a css class added to list cell decorator
when decorator is active.",
          "type": "string"
        },
        "gridClassName": {
          "description": "This is a css class added to color decorator
element on table row when decorator is active.",
          "type": "string"
        },
        "subLocationName": {
          "description": "This property ensures the processing/potentially
application of this decorator when in this sublocation.",
          "type": "string"
        },
        "title": {
          "description": "This is the title (tooltip) to set on the
decorator element. This also supports i18n.",
          "type": "string"
        },
        "method": {
          "description": "Function name to call with the vmo to determine
if decorator should be applied. Function should return a boolean.",
          "type": "string"
        },
        "deps": {
          "description": "The js dependency file where the method resides
in.",
          "type": "string"
        },
```


Unpublished work. © 2025 Siemens

```
        "modelTypes": {
          "description": "Array of objects for which the color decorators
should be valid.",
          "type": "array"
        },
        "conditions": {
          "description": "Conditions for when the decorator should be
applied. Conditions provide live logic evaluation. They can evaluate to
true/false. They can be used in visible-when clauses and can be used for
event handling (onEvent handler). The data value can be live evaluated by
condition. They can leverage Boolean condition expressions like '&&', '||',
'==', '!='",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "$ref": "#/definitions/conditionDef"
            }
          }
        },
        "i18n": {
          "description": "All user facing text should be localized unless
a web client solution supports only one language. The View refers to
localizations defined in the 'i18n' section of the View Model. You can
specify the actual string to be displayed (however, it would not be
localized) OR the name of JSON bundle which holds the localized text to
be shown.",
          "type": "object"
        }
      },
      "minProperties": 2,
      "required": [
        "cellClassName"
      ]
    }
  },
  "__defs": []
}
```

## conditionDef Definition (Decorators)

Expressions can be defined as simple strings or object.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `expression` | expressionStringOrObjectDef | Yes | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## expressionStringOrObjectDef Definition (Decorators)

This definition must satisfy *one* of the following constraints:

- it must be set to a **ExpressionStringOrObjectDefObject1** value
- it must be set to a string value

### Referenced Objects

The following objects are referenced within this definition.

### ExpressionStringOrObjectDefObject1

Expression can be defined as object with the support of below operators.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$source` | **sourceStringOrObjectDef** | | Indicates the reference on data context node to be used as starting point for evaluation. For ex. 'ctx.selected' |
| `$query` | **QueryObject** | | Defines the query to be executed on the source |
| `$and` | array | | Indicates logical 'AND' of each query result |
| `$or` | array | | Indicates logical 'OR' of each query result |
| `$all` | array | | Indicates logical 'ALL' of each query result |
| `$eventSource` | string | | This operator lets user specify the viewId or viewName or 'self' to write a condition to enable the events to be filtered out based on whether or not it is triggered from the same viewModel. |
| `$not` | **NotObject** | | Indicates logical 'NOT' of a query result |

### NotObject

Indicates logical 'NOT' of a query result

#### QueryObject

Defines the query to be executed on the source

## sourceStringOrObjectDef Definition (Decorators)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **SourceStringOrObjectDefObject1** value

#### Referenced Objects

The following objects are referenced within this definition.

#### SourceStringOrObjectDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$adapt` | string | | Indicates that the resulting source should be adapted before evaluating the query |

# Delta Schema

This schema is an array of **DeltaObject** objects.

### Referenced Objects

The following objects are referenced within this schema.

### AuthorObject

*No description specified.*

This object can contain the following properties:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | | The author id of the new delta. |

### ContentObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `query` | string | | the query the delta follows. |
| `variables` | VariablesObject | | the different variables the delta interacts with. |
| `operationName` | null | | |
| `fileName` | string | | |

### DataAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a DataObject1 value
- it must be set to a string value

### DataObject1

*No description specified.*

**Note**
This object *can* have additional properties.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `action` | string | | the action that the update correlates to. |
| `helpTitle` | string | | |

## DeltaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | IdAltObject | | |
| `parent` | ParentAltObject | | |
| `author` | AuthorObject | | |
| `timestamp` | string | | The timestamp the delta was created. |
| `content` | ContentObject | | |
| `updates` | ArrayUpdatesObject | | |

## IdAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a null value

## KeyI18NObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `key` | string | | |
| `source` | string | | |

## MessagesObject

*No description specified.*


Siemens Web Framework
Unpublished work. © 2025 Siemens

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `locale` | string | | It can be set to one of the following values: `en`, `cs_CZ`, `de`, `es`, `fr`, `it`, `ja_JP`, `ko_KR`, `pl_PL`, `pt_BR`, `ru_RU`, `zh_CN`, `zh_TW`. |
| `message` | string | | |

## ParentAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a **ParentObject1** value
- it must be set to a null value

## ParentObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | | The parent id of the new delta. |

## UpdatesObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `config` | string | | The type of file the delta interacts with. |
| `action` | string | | It can be set to one of the following values: `UPDATE`, `DELETE`, `CREATE`. the type of action the delta will be performing. |
| `path` | string | | the path the delta file will be updating. |
| `data` | **DataAltObject** | | |

## VariablesObject

the different variables the delta interacts with.

**Note**
This object *can* have additional properties.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| keyI18N | KeyI18NObject | | |
| messages | ArrayMessagesObject | | |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeDeltaSchema-1.0.0.json",
  "description": "This is the schema the declarative delta files (in JSON
format) must adhere to. The Apollo audit check validates all deltas JSON
files against this schema. For explanation of the keywords of this schema,
please visit https://ajv.js.org/keywords.html",
  "definitions": {},
  "type": "array",
  "additionalItems": false,
  "items": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "id": {
        "oneOf": [
          {
            "type": "string",
            "description": "This is the id of the delta being created."
          },
          {
            "type": "null"
          }
        ]
      },
      "parent": {
        "oneOf": [
          {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "id": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "type": "string",
                "description": "The parent id of the new delta."
              }
            }
          },
          {
            "type": "null",
            "additionalProperties": false
          }
        ]
      },
      "author": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "id": {
            "type": "string",
            "description": "The author id of the new delta."
          }
        }
      },
      "timestamp": {
        "type": "string",
        "description": "The timestamp the delta was created.",
        "additionalProperties": false
      },
      "content": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "query": {
            "type": "string",
            "description": "the query the delta follows."
          },
          "variables": {
            "type": "object",
            "description": "the different variables the delta interacts
with.",
            "additionalProperties": true,
            "properties": {
              "keyI18N": {
                "type": "object",
                "additionalProperties": false,
                "properties": {
                  "key": {
                    "type": "string"
                  },
                  "source": {
                    "type": "string"
                  }
                }
              },
              "messages": {
                "type": "array",
                "additionalItems": false,
                "items": {
```

```
                "type": "object",
                "additionalProperties": false,
                "properties": {
                  "locale": {
                    "type": "string",
                    "enum": [
                      "en",
                      "cs_CZ",
                      "de",
                      "es",
                      "fr",
                      "it",
                      "ja_JP",
                      "ko_KR",
                      "pl_PL",
                      "pt_BR",
                      "ru_RU",
                      "zh_CN",
                      "zh_TW"
                    ]
                  },
                  "message": {
                    "type": "string"
                  }
                }
              }
            }
          }
        },
        "operationName": {
          "type": "null"
        },
        "fileName": {
          "type": "string"
        }
      }
    },
    "updates": {
      "type": "array",
      "additionalItems": false,
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "config": {
            "type": "string",
            "description": "The type of file the delta interacts with."
          },
          "action": {
            "type": "string",
            "enum": [
              "UPDATE",
              "DELETE",
              "CREATE"
            ],
            "description": "the type of action the delta will be
```



Unpublished work. © 2025 Siemens

```
performing."
            },
            "path": {
              "type": "string",
              "description": "the path the delta file will be updating."
            },
            "data": {
              "oneOf": [
                {
                  "type": "object",
                  "additionalProperties": true,
                  "properties": {
                    "action": {
                      "type": "string",
                      "description": "the action that the update correlates
to."
                    },
                    "helpTitle": {
                      "type": "string"
                    }
                  }
                },
                {
                  "type": "string"
                }
              ]
            }
          }
        }
      }
    },
    "__defs": []
}
```

## DragAndDrop Schema

This is the schema the declarative drag and drop (in JSON format) must adhere to. The Apollo audit check validates all dragAndDrop JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with |
| `dragHandlers` | **DragHandlersObject** | | Defines the drag handlers |
| `dropHandlers` | **DropHandlersObject** | | Defines the drop handlers |

| | | | | |
|---|---|---|---|---|
| actions | ActionsObject | Yes | In the drag and drop configuration, actions are declaratively defined. Actions in drag and drop configuration can only be of actionType 'syncFunction'. | |
| conditions | ConditionsObject | | Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in active-when clauses. The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |

## Definitions

This schema includes the following definitions:

- dragActionsDef
- dragHandlerDef
- dropActionsDef
- dropHandlerDef
- syncActionDef

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the drag and drop configuration, actions are declaratively defined. Actions in drag and drop configuration can only be of actionType 'syncFunction'.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | syncActionDef | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in active-when clauses. The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | Condition | | |

## DragHandlersObject

Defines the drag handlers

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | dragHandlerDef | | |

## DropHandlersObject

Defines the drop handlers

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | dropHandlerDef | | |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeDragAndDropSchema-1.0.0.json",
  "description": "This is the schema the declarative drag and drop (in JSON
format) must adhere to. The Apollo audit check validates all dragAndDrop
JSON files against this schema. For explanation of the keywords of this
schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "dragActionsDef": {
      "type": "object",
```

```
      "properties": {
        "dragStart": {
          "type": "string",
          "description": "Name of the action to be executed for drag start
event."
        },
        "dragEnd": {
          "type": "string",
          "description": "Name of the action to be executed for drag end
event."
        }
      },
      "required": [
        "dragStart"
      ]
    },
    "dropActionsDef": {
      "type": "object",
      "properties": {
        "dragEnter": {
          "type": "string",
          "description": "Name of the action to be executed for drag enter
event."
        },
        "dragLeave": {
          "type": "string",
          "description": "Name of the action to be executed for drag leave
event."
        },
        "dragOver": {
          "type": "string",
          "description": "Name of the action to be executed for drag over
event."
        },
        "drop": {
          "type": "string",
          "description": "Name of the action to be executed for drop event."
        }
      },
      "required": [
        "dragOver"
      ]
    },
    "dragHandlerDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "The drag handler definition corresponding to views.",
      "properties": {
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when this drag handler is active for the specified views."
        },
        "views": {
          "type": "array",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "description": "Array of views to set drop handlers on.",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "dragActions": {
          "$ref": "#/definitions/dragActionsDef",
          "description": "Actions for drag operation."
        }
      },
      "required": [
        "views",
        "dragActions"
      ]
    },
    "dropHandlerDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "The drop handler definition corresponding to views.",
      "properties": {
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when this drag handler is active for the specified views."
        },
        "views": {
          "type": "array",
          "description": "Array of views to set drop handlers on.",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "dropActions": {
          "$ref": "#/definitions/dropActionsDef",
          "description": "Actions for drag operation."
        }
      },
      "required": [
        "views",
        "dropActions"
      ]
    },
    "syncActionDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Action type 'syncFunction' executes an action
synchronously.",
      "properties": {
        "actionType": {
          "type": "string",
          "const": "syncFunction"
```

```
        },
        "method": {
          "type": "string",
          "description": "Points to the method name that should be invoked
from the js file specified in 'deps'"
        },
        "inputData": {
          "type": "object",
          "description": "JSON data for the action input."
        },
        "deps": {
          "type": "string",
          "description": "Dependent files needed by the action. For ex. In
case of actionType 'JSFunction', you would need to add a deps on some js
file path."
        }
      }
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
    },
    "dragHandlers": {
      "type": "object",
      "description": "Defines the drag handlers",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/dragHandlerDef"
        }
      }
    },
    "dropHandlers": {
      "type": "object",
      "description": "Defines the drop handlers",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/dropHandlerDef"
        }
      }
    },
    "actions": {
      "type": "object",
      "description": "In the drag and drop configuration, actions are
declaratively defined. Actions in drag and drop configuration can only be
of actionType 'syncFunction'.",
      "additionalProperties": false,
      "patternProperties": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/syncActionDef"
        }
      }
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in active-when clauses. The data
value can be live evaluated by condition. The conditions can refer to view
model state and can update view model state. They can leverage Boolean
condition expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    }
  },
  "required": [
    "actions"
  ],
  "__defs": []
}
```

## dragActionsDef Definition (DragAndDrop)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `dragStart` | string | Yes | Name of the action to be executed for drag start event. |
| `dragEnd` | string | | Name of the action to be executed for drag end event. |

## dragHandlerDef Definition (DragAndDrop)

The drag handler definition corresponding to views.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `activeWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine when this drag handler is active for the specified views. |
| `views` | Arraystring | Yes | Array of views to set drop handlers on. It must contain at least 1 unique item. |
| `dragActions` | dragActionsDef | Yes | Actions for drag operation. |

## dropActionsDef Definition (DragAndDrop)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `dragEnter` | string | | Name of the action to be executed for drag enter event. |
| `dragLeave` | string | | Name of the action to be executed for drag leave event. |
| `dragOver` | string | Yes | Name of the action to be executed for drag over event. |
| `drop` | string | | Name of the action to be executed for drop event. |

## dropHandlerDef Definition (DragAndDrop)

The drop handler definition corresponding to views.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `activeWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine when this drag handler is active for the specified views. |
| `views` | Arraystring | Yes | Array of views to set drop handlers on. It must contain at least 1 unique item. |
| `dropActions` | dropActionsDef | Yes | Actions for drag operation. |


Unpublished work. © 2025 Siemens

## syncActionDef Definition (DragAndDrop)

Action type 'syncFunction' executes an action synchronously.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | string | | It must be set to `syncFunction`. |
| `method` | string | | Points to the method name that should be invoked from the js file specified in 'deps' |
| `inputData` | InputDataObject | | JSON data for the action input. |
| `deps` | string | | Dependent files needed by the action. For ex. In case of actionType 'JSFunction', you would need to add a deps on some js file path. |

#### Referenced Objects

The following objects are referenced within this definition.

#### InputDataObject

JSON data for the action input.

# Indicators Schema

Indicators are used to render an icon on a ViewModelObject based on some condition i.e checkout, checkin, released. This is the schema for declarative indicators (in JSON format) must adhere to. The Apollo audit check validates all indicators JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html** .

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$schema` | string | Yes | The schema against which the JSON file is validated. It should be the same as the file name. For example - 'declarativeIndicatorsSchema-1.0.0.json' is valid, 'declarativeIndicatorsSchema-1.0.0' is not. |
| `indicators` | IndicatorsObject | Yes | The collection of indicators. |

## Referenced Objects

The following objects are referenced within this schema.

## IndicatorsObject

The collection of indicators.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | IndicatorObject | | Name of the unique indicator. |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeIndicatorsSchema-1.0.0.json",
  "description": "Indicators are used to render an icon on a
ViewModelObject based on some condition i.e checkout, checkin, released.
This is the schema for declarative indicators (in JSON format) must adhere
to. The Apollo audit check validates all indicators JSON files against
this schema. For explanation of the keywords of this schema, please visit
https://ajv.js.org/keywords.html .",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "$schema": {
      "type": "string",
      "description": "The schema against which the JSON file
is validated. It should be the same as the file name.
For example - 'declarativeIndicatorsSchema-1.0.0.json' is valid,
'declarativeIndicatorsSchema-1.0.0' is not."
    },
    "indicators": {
      "type": "object",
      "description": "The collection of indicators.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "description": "Name of the unique indicator.",
          "$ref": "declarativeIndicatorObjectSchema-1.0.0.json"
        }
      }
    }
  },
  "required": [
    "$schema",
    "indicators"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
  ],
  "__defs": []
}
```

## indicatorObjDef Definition (IndicatorObject)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `iconId` | string | Yes | The name of the icon/indicator to display without extension or size suffix. For example - 'Failed' is valid, 'indicatorFailed16.svg' is not. |
| `tooltip` | TooltipObject | Yes | The tooltip that should show up on the applied indicator. It should be specified as a collection (array) of properties present on the context vmo. |
| `type` | typeDef | Yes | The type name of the View Model Object referenced by the View Model Property. |
| `isDefaultIndicator` | boolean OR string | | Boolean value indicating if the the indicator is default indicator. Default indicator get's applied if it's the only indicator matching for given vmo. |

### Referenced Objects

The following objects are referenced within this definition.

### TooltipObject

The tooltip that should show up on the applied indicator. It should be specified as a collection (array) of properties present on the context vmo.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `showPropDisplayName` | boolean | | When specified as true the tooltip would include the property display name. default value false. |
| `propNames` | Arraystring | | The collection of properties that appear on tooltip. |

## nameDef Definition (IndicatorObject)

This value is an array of **string** objects.

## typeDef Definition (IndicatorObject)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `names` | nameDef | Yes | List of ViewModelObject types on which the indicators need to be evaluated. |
| `prop` | PropAltObject | | Object containing vmo property on which the indicators need to be evaluated. |

### Referenced Objects

The following objects are referenced within this definition.

#### PropAltObject

Object containing vmo property on which the indicators need to be evaluated.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `names` | nameDef | | List of ViewModelProperties on which the indicators need to be evaluated. |
| `type` | typeDef | | The type name of the ViewModelObject referenced by the ViewModelProperty. This is optional - needed if a referenced type needs to be traversed. |
| `condition` | expressionStringOrObjectDef | | The condition under which the indicator needs to be applied. It is a standard declarative condition which can be either string based or object based. |

This value must satisfy *one* of the following constraints:


Siemens Web Framework
Unpublished work. © 2025 Siemens

- the following properties must be specified: **names**, **type**
- the following properties must be specified: **names**, **condition**

## IndicatorObject Schema

This schema points to the **indicatorObjDef** definition.

### Definitions

This schema includes the following definitions:

- **indicatorObjDef**
- **nameDef**
- **typeDef**

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeIndicatorObjectSchema-1.0.0.json",
  "description": "Indicators are used to render an icon on a
ViewModelObject based on some condition i.e checkout, checkin, released.
This is the schema for declarative indicators (in JSON format). The SWF
audit check validates all indicators JSON files against this schema. For
explanation of the keywords of this schema, please visit https://ajv.js.org/
keywords.html .",
  "definitions": {
    "nameDef": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "typeDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "names": {
          "description": "List of ViewModelObject types on which the
indicators need to be evaluated.",
          "$ref": "#/definitions/nameDef"
        },
        "prop": {
          "type": "object",
          "additionalProperties": false,
          "description": "Object containing vmo property on which the
indicators need to be evaluated.",
          "properties": {
```

```
            "names": {
              "description": "List of ViewModelProperties on which the
indicators need to be evaluated.",
              "$ref": "#/definitions/nameDef"
            },
            "type": {
              "description": "The type name of the ViewModelObject
referenced by the ViewModelProperty. This is optional - needed if a
referenced type needs to be traversed.",
              "$ref": "#/definitions/typeDef"
            },
            "condition": {
              "description": "The condition under which the indicator needs
to be applied. It is a standard declarative condition which can be either
string based or object based.",
              "$ref": "declarativeConditionSchema-1.0.0.json#/definitions/
expressionStringOrObjectDef"
            }
          },
          "oneOf": [
            {
              "required": [
                "names",
                "type"
              ]
            },
            {
              "required": [
                "names",
                "condition"
              ]
            }
          ]
        }
      },
      "required": [
        "names"
      ]
    },
    "indicatorObjDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "iconId": {
          "type": "string",
          "description": "The name of the icon/indicator to display
without extension or size suffix. For example - 'Failed' is valid,
'indicatorFailed16.svg' is not."
        },
        "tooltip": {
          "type": "object",
          "description": "The tooltip that should show up on the applied
indicator. It should be specified as a collection (array) of properties
present on the context vmo.",
          "additionalProperties": false,
          "properties": {
```


Unpublished work. © 2025 Siemens

```
            "showPropDisplayName": {
              "type": "boolean",
              "description": "When specified as true the tooltip would
include the property display name. default value false."
            },
            "propNames": {
              "type": "array",
              "items": {
                "type": "string",
                "description": "The property values that should be used to
create the tooltip on the indicator."
              },
              "description": "The collection of properties that appear on
tooltip."
            }
          }
        },
        "type": {
          "description": "The type name of the View Model Object referenced
by the View Model Property.",
          "$ref": "#/definitions/typeDef"
        },
        "isDefaultIndicator": {
          "description": "Boolean value indicating if the the indicator
is default indicator. Default indicator get's applied if it's the only
indicator matching for given vmo.",
          "type": [
            "boolean",
            "string"
          ]
        }
      },
      "required": [
        "iconId",
        "tooltip",
        "type"
      ]
    }
  },
  "$ref": "#/definitions/indicatorObjDef",
  "__defs": []
}
```

## Kit Schema

This is the schema the declarative kit (in JSON format) must adhere to. The Apollo audit check validates all kit JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Desc |
|---|---|---|---|
| `name` | string | Yes | A nar the k |
| `author` | string | | Name auth organ |
| `srcDir` | string | | A sou direc name defau src_k |
| `srcModule` | string | | A nar sourc file d fx-ap |
| `zipFile` | string | | A sta direc file |
| `level` | integer | Yes | A lay numb the k it wil |
| `skipSonar` | boolean | | sona gena for th defau is tru false not g the s repor |
| `enableWhenToVisibleWhen` | boolean | | |
| `kitDeps` | Arraystring | | Array kits d on th curre must only items |
| `soaDeps` | Arraystring | | Array soas on th |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | curre must at lea uniqu |
| `extraDeps` | Arraystring | | Array to fo inclu the w build conta least uniqu |
| `karmaKitDeps` | Arraystring | | Array kits i whic requi the t the k run. conta least uniqu |
| `modules` | Arraystring | | Array modu inclu while build kit. It conta uniqu items |
| `configuration` | **ConfigurationObject** | | This i decla way exter modu wher can s the confi on fram |
| `solutionDef` | **SolutionDefObject** | | Defin the s |

| | | | |
|---|---|---|---|
| viewAndViewModelRepoConfiguration | ViewAndViewModelRepoConfigurationObject | | Defin proto fetch view view |
| dynamicConfigRepoConfiguration | DynamicConfigRepoConfigurationObject | | Defin proto fetch confi data |
| bundles | BundlesObject | | This is dep in the 6.2 re |
| ignoreMissingKits | boolean | | Set to ignor refer kits t missi |
| OOTB | boolean | | Set to indic this i out-o box A kit. |

## Definitions

This schema includes the following definitions:

- configurationDef

## Referenced Objects

The following objects are referenced within this schema.

## BundlerObject

bundler configuration

This object can contain the following properties:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| `chunkMinSize` | integer | | Override minimum chunk size for bundles in production mode |
| `chunkMaxSize` | integer | | Override maximum chunk size for bundles in production mode |
| `entryFiles` | Arraystring | | Entry file names array. It must contain only unique items. |
| `bootstrapFile` | string | | The path to the bootstrap file starting from the site dir |

## BundlesObject

This property is deprecated in the SWF 6.2 release.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `include` | Arraystring | | undefined. It must contain at least 1 unique item. |
| `deps` | Arraystring | | undefined. It must contain at least 1 unique item. |
| `exclude` | Arraystring | | undefined. It must contain at least 1 unique item. |
| `optimize` | string | | |
| `allowSourceOverwrites` | boolean | | |
| `preserveLicenseComments` | boolean | | |
| `generateSourceMaps` | boolean | | |

## ConfigurationObject

This is the declarative way of extension module where user can specify the configuration on framework.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `\b[A-Za-z]+([A-Z]*[a-z]*[0-9]*)*\b` | configurationDef | | |

### DefaultDragAndDropHandlersObject

This is the declarative configuration point for the default drag and drop handlers.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `dragHandler` | string | Yes | |
| `dropHandler` | string | Yes | |

### DefaultPasteHandlerConfigurationObject

This is the declarative way of extension module where user can specify the image configuration on framework.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `bestTargetFitFinder` | string | Yes | |
| `bestSourceFitFinder` | string | Yes | |
| `pasteHandler` | string | Yes | |
| `deps` | string | Yes | |

### DynamicConfigRepoConfigurationObject

Define HTTP protocol for fetching configuration data

### ImageRepositoryConfigurationObject

This is the declarative way of extension module where user can specify the image configuration on framework.

This object can contain the following properties:


Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| actionType | string | Yes | |
| url | string | Yes | |

## SolutionDefObject

Defination of the solution.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| solutionId | string | Yes | A unique na of the solut |
| brandName | string | Yes | Brand name the solution |
| browserTitle | string | Yes | The title to display in th browser window |
| solutionName | string | Yes | Name of th solution |
| solutionVersion | string | Yes | Version nur of the solut |
| copyrightText | string | Yes | Copyright t |
| companyName | string | Yes | Name of th company. |
| solutionWar | string | | This proper deprecated the SWF 6.2 release. |
| workspaces | array | | String array the worksp defined in t solution. |
| defaultWorkspace | string | | default workspace name of th soluion . It should be f |

| | | | |
|---|---|---|---|
| | | | the worksp<br>defined in t<br>war. |
| `clipboard` | string | | clipboard<br>service file<br>the solution |
| `defaultCommandbarIconSize` | string | | It can be se<br>one of the<br>following<br>values: `sma`<br>`large`.<br>Determines<br>size of the<br>command i<br>across the<br>solution (1<br>or 24px).<br>Defaults to<br>'small'. |
| `authenticator` | string | | Aunthticato<br>service for<br>solution wh<br>will aunthti<br>the user |
| `analytics` | string | | AnalyticsSe<br>is a service<br>manages<br>enablemen<br>logging of<br>messages t<br>backend<br>analytics<br>logging ser |
| `defaultTheme` | | | It can be se<br>one of the<br>following<br>values: `ui-`<br>`lightThe`<br>`ui-`<br>`darkThem`<br>Default the<br>for the solu |
| `commandVisibility` | string | | Service nan<br>for the |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | command visibility log |
| `bundler` | **BundlerObject** | | bundler configuratio |
| `defaultPasteHandlerConfiguration` | **DefaultPasteHandlerConfigurationObject** | | This is the declarative of extensio module wh user can sp the image configuratio on framewo |
| `defaultDragAndDropHandlers` | **DefaultDragAndDropHandlersObject** | | This is the declarative configuratio point for th default dra and drop handlers. |
| `imageRepositoryConfiguration` | **ImageRepositoryConfigurationObject** | | This is the declarative of extensio module wh user can sp the image configuratio on framewo |

## ViewAndViewModelRepoConfigurationObject

Define HTTP protocol for fetching view and view model

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `actionType` | | Yes | It can be set to one of the following values: `GET`, `POST`. HTTP protocol way for calling the methods. |
| `viewUrl` | string | Yes | URL name of the page. |
| `viewModelUrl` | string | Yes | Model defination URL of the string |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeKitSchema-1.0.0.json",
  "description": "This is the schema the declarative kit (in JSON format)
must adhere to. The Apollo audit check validates all kit JSON files against
this schema. For explanation of the keywords of this schema, please visit
https://ajv.js.org/keywords.html",
  "definitions": {
    "configurationDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "output": {
          "enum": [
            "javascript",
            "json"
          ],
          "description": "Type of the configuration files"
        },
        "format": {
          "enum": [
            "array",
            "object"
          ],
          "description": "Format of the contennt of the file"
        },
        "definition": {
          "enum": [
            "module",
            "kit"
          ],
          "description": "Defination of the file at module level or kit
level"
        },
        "multiple": {
          "type": "boolean",
          "description": ""
        },
        "uniqueDepth": {
          "type": "integer",
          "description": ""
        },
        "depsPath": {
          "type": "array",
          "description": "Path to the dependencies"
        }
      }
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "name": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "type": "string",
      "description": "A name of the kit"
    },
    "author": {
      "type": "string",
      "description": "Name of the authoring organization"
    },
    "srcDir": {
      "type": "string",
      "description": "A source directory name /path default is src_kit"
    },
    "srcModule": {
      "type": "string",
      "description": "A name of source war file default is fx-appbase"
    },
    "zipFile": {
      "type": "string",
      "description": "A staging directory zip file"
    },
    "level": {
      "type": "integer",
      "description": "A layer number of the kit where it will reside"
    },
    "skipSonar": {
      "type": "boolean",
      "description": "sonar genaration for the kit. by default value is
true and if false it will not generate the sonar report"
    },
    "enableWhenToVisibleWhen": {
      "type": "boolean"
    },
    "kitDeps": {
      "type": "array",
      "description": "Array of the kits depend on the current kit",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "soaDeps": {
      "type": "array",
      "description": "Array of the soas depend on the current kit",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "extraDeps": {
      "type": "array",
      "description": "Array of deps to force include in the webpack build",
      "items": {
        "type": "string"
      },
      "minItems": 1,
```


Unpublished work. © 2025 Siemens

```
      "uniqueItems": true
    },
    "karmaKitDeps": {
      "type": "array",
      "description": "Array of the  kits  included which is required for
the test of the kit to run",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "modules": {
      "type": "array",
      "description": "Array of the  modules included while building this
kit",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "configuration": {
      "type": "object",
      "description": "This is the declarative way of extension module where
user can specify the configuration on framework.",
      "additionalProperties": false,
      "patternProperties": {
        "\\b[A-Za-z]+([A-Z]*[a-z]*[0-9]*)*\\b": {
          "$ref": "#/definitions/configurationDef"
        }
      }
    },
    "solutionDef": {
      "type": "object",
      "description": "Defination of the solution.",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "solutionId": {
            "type": "string",
            "description": "A unique name of the solution"
          },
          "brandName": {
            "type": "string",
            "description": "Brand name of the solution"
          },
          "browserTitle": {
            "type": "string",
            "description": "The title to display in the browser window"
          },
          "solutionName": {
            "type": "string",
            "description": "Name of the solution"
          },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "solutionVersion": {
            "type": "string",
            "description": "Version number of the solution."
          },
          "copyrightText": {
            "type": "string",
            "description": "Copyright text."
          },
          "companyName": {
            "type": "string",
            "description": "Name of the company."
          },
          "solutionWar": {
            "type": "string",
            "description": "This property is deprecated in the SWF 6.2
release."
          },
          "workspaces": {
            "type": "array",
            "description": "String array of the workspaces defined in the
solution."
          },
          "defaultWorkspace": {
            "type": "string",
            "description": "default workspace name of the soluion . It
should be from the workspaces defined in the war."
          },
          "clipboard": {
            "type": "string",
            "description": "clipboard service file for the solution."
          },
          "defaultCommandbarIconSize": {
            "type": "string",
            "enum": [
              "small",
              "large"
            ],
            "description": "Determines the size of the command icons across
the solution (16px or 24px). Defaults to 'small'."
          },
          "authenticator": {
            "type": "string",
            "description": "Aunthticator service for the solution which
will aunthticate the user"
          },
          "analytics": {
            "type": "string",
            "description": "AnalyticsService is a service that manages
enablement & logging of messages to a backend analytics logging server."
          },
          "defaultTheme": {
            "enum": [
              "ui-lightTheme",
              "ui-darkTheme"
            ],
            "description": "Default theme for the solution."
```

```
          },
          "commandVisibility": {
            "type": "string",
            "description": "Service name for the command visibility logic."
          },
          "bundler": {
            "type": "object",
            "description": "bundler configuration",
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "chunkMinSize": {
                  "type": "integer",
                  "description": "Override minimum chunk size for bundles
in production mode"
                },
                "chunkMaxSize": {
                  "type": "integer",
                  "description": "Override maximum chunk size for bundles
in production mode"
                },
                "entryFiles": {
                  "type": "array",
                  "description": "Entry file names array",
                  "items": {
                    "type": "string"
                  },
                  "uniqueItems": true,
                  "minItems": 0
                },
                "bootstrapFile": {
                  "type": "string",
                  "description": "The path to the bootstrap file starting
from the site dir"
                }
              }
            }
          },
          "defaultPasteHandlerConfiguration": {
            "type": "object",
            "description": "This is the declarative way of extension module
where user can specify the image configuration on framework.",
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "bestTargetFitFinder": {
                  "type": "string"
                },
                "bestSourceFitFinder": {
                  "type": "string"
                },
                "pasteHandler": {
                  "type": "string"
                },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "deps": {
                  "type": "string"
                }
              },
              "required": [
                "bestTargetFitFinder",
                "bestSourceFitFinder",
                "pasteHandler",
                "deps"
              ]
            }
          },
          "defaultDragAndDropHandlers": {
            "type": "object",
            "description": "This is the declarative configuration point for
the default drag and drop handlers.",
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "dragHandler": {
                  "type": "string"
                },
                "dropHandler": {
                  "type": "string"
                }
              },
              "required": [
                "dragHandler",
                "dropHandler"
              ]
            }
          },
          "imageRepositoryConfiguration": {
            "type": "object",
            "description": "This is the declarative way of extension module
where user can specify the image configuration on framework.",
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "actionType": {
                  "type": "string"
                },
                "url": {
                  "type": "string"
                }
              },
              "required": [
                "actionType",
                "url"
              ]
            }
          }
        },
        "required": [
```

```
          "solutionId",
          "brandName",
          "solutionName",
          "solutionVersion",
          "browserTitle",
          "companyName",
          "copyrightText"
        ]
      }
    },
    "viewAndViewModelRepoConfiguration": {
      "type": "object",
      "description": "Define HTTP protocol for fetching view and view
model",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "actionType": {
            "enum": [
              "GET",
              "POST"
            ],
            "description": "HTTP protocol way for calling the methods."
          },
          "viewUrl": {
            "type": "string",
            "description": "URL name of the page."
          },
          "viewModelUrl": {
            "type": "string",
            "description": "Model  defination URL of the string"
          }
        },
        "required": [
          "actionType",
          "viewUrl",
          "viewModelUrl"
        ]
      }
    },
    "dynamicConfigRepoConfiguration": {
      "type": "object",
      "description": "Define HTTP protocol for fetching configuration data",
      "items": {
        "type": "object",
        "additionalProperties": true
      }
    },
    "bundles": {
      "type": "object",
      "description": "This property is deprecated in the SWF 6.2 release.",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "include": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
          },
          "deps": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
          },
          "exclude": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
          },
          "optimize": {
            "type": "string"
          },
          "allowSourceOverwrites": {
            "type": "boolean"
          },
          "preserveLicenseComments": {
            "type": "boolean"
          },
          "generateSourceMaps": {
            "type": "boolean"
          }
        }
      }
    },
    "ignoreMissingKits": {
      "type": "boolean",
      "description": "Set to true to ignore any referenced kits that are
missing."
    },
    "OOTB": {
      "type": "boolean",
      "description": "Set to true to indicate that this is an out-of-the-
box Apollo kit."
    }
  },
  "required": [
    "name",
    "level"
  ],
```

```
  "__defs": []
}
```

### configurationDef Definition (Kit)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `output` | | | It can be set to one of the following values: `javascript`, `json`. Type of the configuration files |
| `format` | | | It can be set to one of the following values: `array`, `object`. Format of the contennt of the file |
| `definition` | | | It can be set to one of the following values: `module`, `kit`. Defination of the file at module level or kit level |
| `multiple` | boolean | | |
| `uniqueDepth` | integer | | |
| `depsPath` | array | | Path to the dependencies |

## Module Schema

This is the schema the declarative module definition (in JSON format) must adhere to. The Apollo audit check validates all module JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | Yes | A name of the module |
| `description` | string | | A description of the module |
| `siteDir` | string | | site directory path for base site file |
| `techScoreboardURL` | string | | only valid for the type 'repo' , used to reference 3rd party approvals |
| `srcDir` | string | | source directory path for module file |
| `testDir` | string | | test directory path for module file |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `version` | string | | only valid for the type 'repo' , used to indicate 3rd party version |
| `pathOffset` | string | | Path offset relative to the site assets directory. |
| `copyAssets` | array | | Configuration for copying assets |
| `jars` | array | | **This property is deprecated** (Only used to support GWT/Selenium.) Array of the 3rd party jar dependancy on the module |
| `npmPaths` | Arraystring | | **This property is deprecated** (No longer used) Array of the npm modules in the solution. It must contain only unique items. |
| `libPath` | string | | **This property is deprecated** (No longer used) library path relative to assets/lib |
| `showcaseFiles` | ShowcaseFilesObject | | Contribution to showcase. Additional files required to support this module in the showcase |
| `sonarCoverageExclusions` | Arraystring | | Array of file patterns to be excluded from SonarQube coverage processing. It must contain at least 1 unique item. |
| `sonarExclusions` | Arraystring | | Array of file patterns to be excluded from SonarQube processing. It must contain at least 1 unique item. |
| `type` | Arraystring | | Array of file patterns to be excluded from SonarQube processing. It must contain at least 1 unique item. |
| `excludeFromAuditDeps` | boolean | | if value is true then it will not check the audit for the module |
| `karmaInclude` | boolean | | **This property is deprecated** (No longer used) It used to indicate that module should be included in the Karma setup assets directory |
| `maintainStructure` | boolean | | **This property is deprecated** (No longer used) |
| `skipAudit` | boolean | | if value is true then this module will be exempt from the audit for the module |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `skipJSDoc` | boolean | | **This property is deprecated** (No longer used) if value is true then this module will be exempt from check the JS Doc for the module |
| `skipTest` | boolean | | **This property is deprecated** (No longer used) if value is true then this module will be exempt from check the test for the module |
| `themeExtensions` | Arraystring | | Array of consumer provided theme (.scss) files extending AFX's theme files. For supporting modular-theme-extensions. It must contain at least 1 unique item. |
| `_internal` | **_internalObject** | | |

## Definitions

This schema includes the following definitions:

- **showcaseFileDef**

## Referenced Objects

The following objects are referenced within this schema.

## OwnershipInfoObject

Includes ownership details for the module.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `team` | string | | The team overseeing the module. |
| `ART` | string | | The ART to which the team belongs |
| `valueStream` | string | | The value stream to which the team belongs |
| `segment` | string | | The segment to which the value stream belongs |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## ShowcaseFilesObject

Contribution to showcase. Additional files required to support this module in the showcase

**Note**
This object *can* have additional properties.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `\b[A-Za-z]+([A-Z]*[a-z]*[0-9]*)*\b` | showcaseFileDef | | |

## _internalObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `ownershipInfo` | OwnershipInfoObject | | Includes ownership details for the module. |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeModuleSchema-1.0.0.json",
  "description": "This is the schema the declarative module definition (in
JSON format) must adhere to. The Apollo audit check validates all module
JSON files against this schema. For explanation of the keywords of this
schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "showcaseFileDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "moduleName": {
          "type": "string",
          "description": "Module name"
        },
        "path": {
          "type": "string",
          "description": "path of the showcase file"
        }
      }
```

```
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "name": {
      "type": "string",
      "description": "A name of the module"
    },
    "description": {
      "type": "string",
      "description": "A description of the module"
    },
    "siteDir": {
      "type": "string",
      "description": "site directory path for base site file "
    },
    "techScoreboardURL": {
      "type": "string",
      "description": "only valid for the type 'repo' , used to reference
3rd party approvals"
    },
    "srcDir": {
      "type": "string",
      "description": "source directory path for module file "
    },
    "testDir": {
      "type": "string",
      "description": "test directory path for module file "
    },
    "version": {
      "type": "string",
      "description": "only valid for the type 'repo' , used to indicate 3rd
party version"
    },
    "pathOffset": {
      "type": "string",
      "description": "Path offset relative to the site assets directory."
    },
    "copyAssets": {
      "type": "array",
      "description": "Configuration for copying assets",
      "minItems": 0
    },
    "jars": {
      "type": "array",
      "description": "Array of the  3rd party jar dependancy on the module",
      "deprecated": "Only used to support GWT/Selenium."
    },
    "npmPaths": {
      "type": "array",
      "description": "Array of the  npm modules in the solution",
      "items": {
        "type": "string"
      },
      "minItems": 0,
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "uniqueItems": true,
      "deprecated": "No longer used"
    },
    "libPath": {
      "type": "string",
      "description": "library path relative to assets/lib",
      "deprecated": "No longer used"
    },
    "showcaseFiles": {
      "type": "array",
      "description": "Contribution to showcase. Additional files required
to support this module in the showcase",
      "additionalProperties": true,
      "patternProperties": {
        "\\b[A-Za-z]+([A-Z]*[a-z]*[0-9]*)*\\b": {
          "$ref": "#/definitions/showcaseFileDef"
        }
      }
    },
    "sonarCoverageExclusions": {
      "type": "array",
      "description": "Array of file patterns to be excluded from SonarQube
coverage processing",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "sonarExclusions": {
      "type": "array",
      "description": "Array of file patterns to be excluded from SonarQube
processing",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "type": {
      "type": "array",
      "description": "Array of file patterns to be excluded from SonarQube
processing",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "excludeFromAuditDeps": {
      "type": "boolean",
      "description": "if value is true then it will not check the audit for
the module"
    },
    "karmaInclude": {
      "type": "boolean",
```

```
      "description": "It used to indicate that module should be included in
the Karma setup assets directory",
      "deprecated": "No longer used"
    },
    "maintainStructure": {
      "type": "boolean",
      "deprecated": "No longer used"
    },
    "skipAudit": {
      "type": "boolean",
      "description": "if value is true then this module will be exempt from
the audit for the module"
    },
    "skipJSDoc": {
      "type": "boolean",
      "description": "if value is true then this module will be exempt from
check the JS Doc for the module",
      "deprecated": "No longer used"
    },
    "skipTest": {
      "type": "boolean",
      "description": "if value is true then this module will be exempt from
check the test for the module",
      "deprecated": "No longer used"
    },
    "themeExtensions": {
      "type": "array",
      "description": "Array of consumer provided theme (.scss) files
extending AFX's theme files. For supporting modular-theme-extensions.",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "_internal": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "ownershipInfo": {
          "type": "object",
          "description": "Includes ownership details for the module.",
          "additionalProperties": false,
          "properties": {
            "team": {
              "type": "string",
              "description": "The team overseeing the module."
            },
            "ART": {
              "type": "string",
              "description": "The ART to which the team belongs"
            },
            "valueStream": {
              "type": "string",
              "description": "The value stream to which the team belongs"
            },
```


Unpublished work. © 2025 Siemens

```
            "segment": {
              "type": "string",
              "description": "The segment to which the value stream belongs"
            }
          }
        }
      }
    }
  },
  "required": [
    "name"
  ],
  "__defs": []
}
```

### showcaseFileDef Definition (Module)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `moduleName` | string | | Module name |
| `path` | string | | path of the showcase file |

## NavigationURLToken Schema

This configuration file can be used to have a navigation url token that can be used to navigate to specific page in the application.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$schema` | string OR integer | Yes | Version of the schema it complies with |
| `navigationHandlers` | NavigationHandlersObject | Yes | A navigation can have many navigation handlers. However at any given time, there may at most be only one active navigationHandler for a navigation. A declarative 'activeWhen' condition which evaluates a boolean expression |

| | | | | |
|---|---|---|---|---|
| | | | controls whether a handler is currently active. If more than one handler for a given navigate evaluates to true, then the more 'specfic' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the navigate handler will determine where to navigate. | |
| `actions` | **ActionsObject** | Yes | In navigationURLToken.json, actions of actionType 'navigate' is only supported. It is used to navigate to a specific url using configurations like navigateTo, navigateIn, navigationParams. | |
| `conditions` | **ConditionsObject** | Yes | Conditions provide live logic evaluation. They can evaluate to true/false. The data value can be live evaluated by condition. The conditions can refer to localContext. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |

## Definitions

This schema includes the following definitions:

- **navigationHandlersDef**

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In navigationURLToken.json, actions of actionType 'navigate' is only supported. It is used to navigate to a specific url using configurations like navigateTo, navigateIn, navigationParams.

You can define properties on this object with a name matching one of the following patterns:


Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | NavigateDef | | |

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. The data value can be live evaluated by condition. The conditions can refer to localContext. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | Condition | | |

## NavigationHandlersObject

A navigation can have many navigation handlers. However at any given time, there may at most be only one active navigationHandler for a navigation. A declarative 'activeWhen' condition which evaluates a boolean expression controls whether a handler is currently active. If more than one handler for a given navigate evaluates to true, then the more 'specfic' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the navigate handler will determine where to navigate.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | navigationHandlersDef | | |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeNavigationURLTokenSchema-1.0.0.json",
  "description": "This configuration file can be used to have a navigation
url token that can be used to navigate to specific page in the
application.",
  "definitions": {
    "navigationHandlersDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "action": {
          "description": "The name of the declarative navigate action to be
```

```
executed.",
          "type": "string"
        },
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which
will determine when this navigation handler is active for the associated
navigation."
        }
      },
      "required": [
        "action",
        "activeWhen"
      ]
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "$schema": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
    },
    "navigationHandlers": {
      "type": "object",
      "description": "A navigation can have many navigation handlers.
However at any given time, there may at most be only one active
navigationHandler for a navigation.  A declarative 'activeWhen' condition
which evaluates a boolean expression controls whether a handler is
currently active.  If more than one handler for a given navigate evaluates
to true, then the more 'specfic' condition (i.e. the condition with longer
expression) is chosen to be active.  By being active, the navigate handler
will determine where to navigate.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/navigationHandlersDef"
        }
      }
    },
    "actions": {
      "type": "object",
      "description": "In navigationURLToken.json, actions of actionType
'navigate' is only supported. It is used to navigate to a specific url
using configurations like navigateTo, navigateIn, navigationParams.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
NavigateDef"
        }
      }
    },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. The data value can be live evaluated by condition.
The conditions can refer to localContext. They can leverage Boolean
condition expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    }
  },
  "required": [
    "$schema",
    "navigationHandlers",
    "actions",
    "conditions"
  ],
  "__defs": []
}
```

### navigationHandlersDef Definition (NavigationURLToken)

*No description specified*.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `action` | string | Yes | The name of the declarative navigate action to be executed. |
| `activeWhen` | **conditionOrBoolDef** | Yes | The name of the declarative condition which will determine when this navigation handler is active for the associated navigation. |

## Paste Schema

This is the schema the declarative pastes View Model (in JSON format) must adhere to. The Apollo audit check validates all pastes View Model JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with |
| `actions` | ActionsObject | | In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button. |
| `preferences` | Arraystring | | undefined. It must contain only unique items. |
| `functions` | functionsDef | | |
| `messages` | MessagesObject | | Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can |


Unpublished work. © 2025 Siemens

| | | | | |
|---|---|---|---|---|
| | | | even leverage view model data and conditions. | |
| `conditions` | **ConditionsObject** | | Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |
| `i18n` | **I18nObject** | | All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown. | |
| `onEvent` | Array**OnEventObject** | | The View Model can register to react to events by declaring an onEvent | |

| | | | |
|---|---|---|---|
| | | | section. It can then conditionally execute a declarative action. |
| `pasteHandlers` | PasteHandlersObject | Yes | There can many paste Handler. However at any given time, there may at most be only one active pasteHandler for a context. A declarative 'activeWhen' condition which evaluates a boolean expression controls whether a handler is currently active. If more than one handler for a given context evaluates to true, then the more 'specfic' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the paste handler will determine What a action to be executed. |
| `defaultPasteHandler` | DefaultPasteHandlerObject | | |
| `defaultPasteFileHandler` | DefaultPasteFileHandlerObject | | |

## Definitions

This schema includes the following definitions:

- actionDef
- pasteHandlersDef


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | actionDef | | |

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | Condition | | |

## CriteriaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `panelId` | string | | |
| `destPanelId` | string | | |

You can define properties on this object with a name matching one of the following patterns:


Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | number OR string OR boolean | | |

### DefaultPasteFileHandlerObject

*No description specified.*

### DefaultPasteHandlerObject

*No description specified.*

### EventDataObject

*No description specified.*

### I18nObject

All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | array OR string | | |

### InputArgsObject

*No description specified.*

### MessagesObject

Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions.


Siemens Web Framework
Unpublished work. © 2025 Siemens

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | messageDef | | |

## OnEventObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `eventId` | string | | The id of an event that the viewModel is interested to listen to. |
| `criteria` | CriteriaObject | | |
| `eventSource` | string | | (Optional) set it to 'current' to make event listeners only listen to events fired by the current view model |
| `action` | string | | An action that is defined in the actions section of the viewModel which would be executed only if the specified condition evaluates to true. |
| `message` | string | | |
| `inputArgs` | InputArgsObject | | |
| `cacheEventData` | boolean | | |
| `condition` | string | | |
| `eventData` | EventDataObject | | |

## PasteHandlersObject

There can many paste Handler. However at any given time, there may at most be only one active pasteHandler for a context. A declarative 'activeWhen' condition which evaluates a boolean expression controls whether a handler is currently active. If more than one handler for a given context evaluates to true, then the more 'specfic' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the paste handler will determine What a action to be executed.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | pasteHandlersDef | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativePasteSchema-1.0.0.json",
  "description": "This is the schema the declarative pastes View Model (in
JSON format) must adhere to. The Apollo audit check validates all pastes
View Model JSON files against this schema. For explanation of the keywords
of this schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "pasteHandlersDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "action": {
          "description": "The name of the declarative action
(REST,SOA,Event or JSFunction) to be executed.",
          "type": "string"
        },
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when this paste handler is active for the associated paste."
        }
      },
      "required": [
        "action"
      ]
    },
    "actionDef": {
      "description": "Actions can be of different actionType (REST,
JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
      "oneOf": [
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "TcSoaService"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
TcSoaServiceDef"
          },
          "else": false
        },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "dataProvider"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
dataProviderActionDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Event"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
EventDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "RESTService"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
RESTServiceDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Copy"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
CopyDef"
          },
          "else": false
        },
```

```
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Navigate"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
NavigateDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "popup"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
popupDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "JSFunctionAsync"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
JSFunctionAsyncDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "JSFunction"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
JSFunctionDef"
          },
          "else": false
        },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Edit"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
EditDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "batchJob"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
batchJobDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "GraphQL"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
GraphQLDef"
          },
          "else": false
        }
      ]
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
    },
    "actions": {
```

```
      "type": "object",
      "description": "In the View Model, actions are declaratively
defined. Actions can be of different 'actionType' (REST, JSFunction,
JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which
specifies the JSON input for the action and 'outputData' for mapping the
output response. Actions can be used in dataProviders or UI elements such
as an aw-button.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/actionDef"
        }
      }
    },
    "preferences": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "functions": {
      "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
functionsDef"
    },
    "messages": {
      "type": "object",
      "description": "Messages are declaratively defined in the View Model.
Messages are launched by events and actions. They cover information,
warning, error notifications. They give the user more deeper customization
options. They are localized. They can fire some actions. They can even
leverage view model data and conditions.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
messageDef"
        }
      }
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and
can be used for event handling (onEvent handler). The data value can
be live evaluated by condition. The conditions can refer to view model
state and can update view model state. They can leverage Boolean condition
expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    },
    "i18n": {
      "type": "object",
```


Unpublished work. © 2025 Siemens

```
      "description": "All user facing text should be localized unless
a web client solution supports only one language. The View refers to
localizations defined in the 'i18n' section of the View Model. You can
specify the actual string to be displayed (however, it would not be
localized) OR the name of JSON bundle which holds the localized text to
be shown.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": [
            "array",
            "string"
          ],
          "items": {
            "type": "string",
            "description": "Name of the i18n bundle file to be specified in
array"
          }
        }
      }
    },
    "onEvent": {
      "type": "array",
      "additionalProperties": false,
      "description": "The View Model can register to react to events
by declaring an onEvent section. It can then conditionally execute a
declarative action.",
      "required": [
        "eventId",
        "action"
      ],
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "eventId": {
            "type": "string",
            "description": "The id of an event that the viewModel is
interested to listen to."
          },
          "criteria": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
                "type": [
                  "number",
                  "string",
                  "boolean"
                ]
              }
            },
            "properties": {
              "panelId": {
                "type": "string"
              },
              "destPanelId": {
```

```
                "type": "string"
              }
            }
          },
          "eventSource": {
            "type": "string",
            "description": "(Optional) set it to 'current' to make event
listeners only listen to events fired by the current view model"
          },
          "action": {
            "type": "string",
            "description": "An action that is defined in the actions
section of the viewModel which would be executed only if the specified
condition evaluates to true."
          },
          "message": {
            "type": "string"
          },
          "inputArgs": {
            "type": "object"
          },
          "cacheEventData": {
            "type": "boolean"
          },
          "condition": {
            "type": "string"
          },
          "eventData": {
            "type": "object"
          }
        }
      }
    },
    "pasteHandlers": {
      "type": "object",
      "description": "There can many paste Handler.  However at any given
time, there may at most be only one active pasteHandler for a context.
A declarative 'activeWhen' condition which evaluates a boolean expression
controls whether a handler is currently active.  If more than one handler
for a given context evaluates to true, then the more 'specfic' condition
(i.e. the condition with longer expression) is chosen to be active.
By being active, the paste handler will determine What a action to be
executed.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/pasteHandlersDef"
        }
      }
    },
    "defaultPasteHandler": {
      "type": "object"
    },
    "defaultPasteFileHandler": {
      "type": "object"
    }
  },
```


Unpublished work. © 2025 Siemens

```
  "required": [
    "pasteHandlers"
  ],
  "__defs": []
}
```

## actionDef Definition (Paste)

Actions can be of different actionType (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have inputData which specifies the JSON input for the action and outputData for mapping the output response.

This definition must satisfy *one* of the following constraints:

- if **actionType** is set to `TcSoaService` **TcSoaServiceDef**
- if **actionType** is set to `dataProvider` **dataProviderActionDef**
- if **actionType** is set to `Event` **EventDef**
- if **actionType** is set to `RESTService` **RESTServiceDef**
- if **actionType** is set to `Copy` **CopyDef**
- if **actionType** is set to `Navigate` **NavigateDef**
- if **actionType** is set to `popup` **popupDef**
- if **actionType** is set to `JSFunctionAsync` **JSFunctionAsyncDef**
- if **actionType** is set to `JSFunction` **JSFunctionDef**
- if **actionType** is set to `Edit` **EditDef**
- if **actionType** is set to `batchJob` **batchJobDef**
- if **actionType** is set to `GraphQL` **GraphQLDef**

## pasteHandlersDef Definition (Paste)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `action` | string | Yes | The name of the declarative action (REST,SOA,Event or JSFunction) to be executed. |
| `activeWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine when this paste handler is active for the associated paste. |

## PropertyRendererTemplates Schema

This is the schema the declarative property renderer templates (in JSON format) must adhere to. The Apollo audit check validates all propertyRendererTemplates View Model JSON files against this schema. For explanation of the keywords of this schema, please visit https://ajv.js.org/keywords.html

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | PropertyRendererTemplatesPatternObject | | |

### Referenced Objects

The following objects are referenced within this schema.

### PropertyRendererTemplatesPatternObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `grids` | Arraystring | Yes | List of table grid ids to apply this template to. If empty, applies template to all tables |
| `columns` | Arraystring | Yes | List of column names to apply this property renderer template to. |
| `tooltip` | Arraystring | | List of properties to be shown as tooltips when hovering over a cell in the columns listed in columns |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `template` | string | | Inline HTML template code to apply to cells in the columns listed in columns |
| `templateUrl` | string | | HTML file path to apply as a template to cells in the columns listed in columns |
| `renderFunction` | string | | Function to apply to cells in the columns listed in columns |
| `deps` | string | | File path where the render function or header component is located in |
| `dependentServices` | Arraystring | | List of service files that need to be injected/used in template implementation. |
| `headerTemplate` | string | | Inline HTML template code to apply to the header of the columns listed in columns |
| `headerTemplateUrl` | string | | HTML file path to apply as a template to the header of the columns listed in columns |
| `headerRenderFunction` | string | | Function to apply to the header of the columns listed in columns |
| `headerComponent` | string | | A component to render the header of the table |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativePropertyRendererTemplatesSchema-1.0.0.json",
  "description": "This is the schema the declarative property renderer
templates (in JSON format) must adhere to. The Apollo audit check validates
all propertyRendererTemplates View Model JSON files against this schema.
For explanation of the keywords of this schema, please visit https://
ajv.js.org/keywords.html",
  "type": "object",
  "additionalProperties": false,
  "patternProperties": {
    "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "grids": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of table grid ids to apply this template to.
If empty, applies template to all tables"
        },
        "columns": {
          "type": "array",
          "items": {
```

```
            "type": "string"
          },
          "description": "List of column names to apply this property
renderer template to."
        },
        "tooltip": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of properties to be shown as tooltips when
hovering over a cell in the columns listed in columns"
        },
        "template": {
          "type": "string",
          "description": "Inline HTML template code to apply to cells in
the columns listed in columns"
        },
        "templateUrl": {
          "type": "string",
          "description": "HTML file path to apply as a template to cells in
the columns listed in columns"
        },
        "renderFunction": {
          "type": "string",
          "description": "Function to apply to cells in the columns listed
in columns"
        },
        "deps": {
          "type": "string",
          "description": "File path where the render function or header
component is located in"
        },
        "dependentServices": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "List of service files that need to be injected/
used in template implementation."
        },
        "headerTemplate": {
          "type": "string",
          "description": "Inline HTML template code to apply to the header
of the columns listed in columns"
        },
        "headerTemplateUrl": {
          "type": "string",
          "description": "HTML file path to apply as a template to the
header of the columns listed in columns"
        },
        "headerRenderFunction": {
          "type": "string",
          "description": "Function to apply to the header of the columns
listed in columns"
        },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "headerComponent": {
          "type": "string",
          "description": "A component to render the header of the table"
        }
      },
      "required": [
        "grids",
        "columns"
      ]
    }
  },
  "__defs": []
}
```

## ReusableSchemaDefs Schema

### Definitions

This schema includes the following definitions:

- conditionOrBoolDef
- dataMapperEachPropDef
- fieldDef
- functionsDef
- legacyConditionsDef
- messageDef
- paramLocalizationDef
- propDbValueDefForDataMapper
- propDbValueDef
- structDef
- viewModelObjectDef
- viewModelPropDef

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeReusableSchemaDefs-1.0.0.json",
  "description": "This file contains re-usable schema definitions which are
used across multiple files. For explanation of the keywords of this schema,
please visit https://ajv.js.org/keywords.html .",
  "definitions": {
```

```
    "legacyConditionsDef": {
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_().,]*$": {
          "type": "object",
          "propertyNames": {
            "enum": [
              "$eq",
              "$lt",
              "$lte",
              "$gt:",
              "$gte",
              "$ne",
              "$neq",
              "$in",
              "$notin",
              "$isNull",
              "$notNull",
              "$vlookup",
              "$regexp",
              "$notinrange",
              "$notinregexp",
              "$typeOf"
            ]
          },
          "properties": {
            "$eq": {
              "type": [
                "string",
                "integer",
                "boolean",
                "null",
                "array"
              ]
            },
            "$lt": {
              "type": [
                "integer"
              ]
            },
            "$lte": {
              "type": [
                "integer"
              ]
            },
            "$gt": {
              "type": [
                "integer"
              ]
            },
            "$gte": {
              "type": [
                "integer"
              ]
            },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "$ne": {
              "type": [
                "string",
                "integer",
                "boolean",
                "array"
              ]
            },
            "$neq": {
              "type": [
                "string",
                "integer",
                "boolean",
                "array"
              ]
            },
            "$in": {
              "type": "array"
            },
            "$notin": {
              "type": "array"
            },
            "$isNull": {
              "type": [
                "boolean"
              ]
            },
            "$notNull": {
              "type": [
                "boolean"
              ]
            },
            "$vlookup": {
              "type": [
                "object"
              ]
            },
            "$regexp": {
              "type": [
                "string"
              ],
              "format": "regex"
            },
            "$notinrange": {
              "type": "array",
              "items": {
                "type": [
                  "string",
                  "integer"
                ]
              }
            },
            "$notinregexp": {
              "type": [
                "string"
              ],
```

```
              "format": "regex"
            },
            "$typeOf": {
              "type": "string"
            }
          },
          "oneOf": [
            {
              "required": [
                "$eq"
              ]
            },
            {
              "required": [
                "$lt"
              ]
            },
            {
              "required": [
                "$lte"
              ]
            },
            {
              "required": [
                "$gt:"
              ]
            },
            {
              "required": [
                "$gte"
              ]
            },
            {
              "required": [
                "$ne"
              ]
            },
            {
              "required": [
                "$neq"
              ]
            },
            {
              "required": [
                "$in"
              ]
            },
            {
              "required": [
                "$notin"
              ]
            },
            {
              "required": [
                "$isNull"
              ]
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            },
            {
              "required": [
                "$notNull"
              ]
            },
            {
              "required": [
                "$vlookup"
              ]
            },
            {
              "required": [
                "$regexp"
              ]
            },
            {
              "required": [
                "$notinrange"
              ]
            },
            {
              "required": [
                "$notinregexp"
              ]
            },
            {
              "required": [
                "$typeOf"
              ]
            }
          ]
        }
      }
    },
    "structDef": {
      "type": "object",
      "properties": {
        "structName": {
          "type": "string"
        },
        "structValue": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/fieldDef"
          }
        }
      },
      "required": [
        "structName",
        "structValue"
      ]
    },
    "fieldDef": {
      "type": "object",
      "properties": {
```

```
        "fieldName": {
          "type": "string"
        },
        "fieldValue": {
          "type": "string"
        }
      },
      "required": [
        "fieldName",
        "fieldValue"
      ]
    },
    "dataMapperEachPropDef": {
      "additionalProperties": false,
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "description": "Name of property."
        },
        "displayValue": {
          "type": "string",
          "description": "display value of the view model property"
        },
        "displayName": {
          "type": "string",
          "description": "Display name on the DOM"
        },
        "type": {
          "type": "string",
          "description": "PreDefine type of the object. Valid type must be
from above list"
        },
        "isRequired": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "If isRequired is true , it should be compulsory
property to fill by user"
        },
        "isModifiable": {
          "type": [
            "boolean",
            "string"
          ],
          "description": " true if the user should have the ability to
change the property's value. false if the value is read-only."
        },
        "isAutoAssignable": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "   true if the property's value can/should be
assigned automatically by server.   flse if the property's value is not
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
normally assigned/controlled by server."
        },
        "isSelectOnly": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "'isSelectOnly' state of ViewModelProperty.This
property allows the selected lov entry to be select only, i.e, user cannot
type in the widget after   selecting the lov entry. This change is to
support isSelectOnly in aw-widget used as a list box."
        },
        "referenceTypeName": {
          "type": "string",
          "description": "name of typed reference property"
        },
        "oldValue": {
          "type": "string"
        },
        "oldValues": {
          "type": "array"
        },
        "value": {
          "$ref": "#/definitions/propDbValueDefForDataMapper"
        },
        "dataProvider": {
          "type": "string"
        },
        "isEnabled": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "If isEnabled is true, the property available for
modify and default value is true."
        },
        "maxArraySize": {
          "type": "integer",
          "description": "maximum array length for that property"
        },
        "renderingHint": {
          "type": "string",
          "description": "depending on the type, this string indicates
some variation in how the   property's value should be displayed (e.g.
For 'BOOLEAN' type, valid values   include 'radiobutton','togglebutton',
'checkbox'. For 'STRING' type, valid   values include 'label',
'textbox','textfield', 'textarea', 'longtext')"
        },
        "autofocus": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "defines whether the widget needs to be
autofocused or not"
        },
```

```
        "condition": {
          "type": "string",
          "description": "the name of the condition in validation criteria."
        },
        "requiredText": {
          "type": "string",
          "description": "Hint text to the user"
        },
        "propertyRadioTrueText": {
          "type": "string",
          "description": "String displayed on the radio box of the radio is
selected"
        },
        "propertyRadioFalseText": {
          "type": "string",
          "description": "String displayed on the radio box of the radio is
not selected"
        },
        "hasLov": {
          "type": "boolean",
          "description": "If the object has more than one object to hold ,
this value can be true"
        },
        "numberOfLines": {
          "type": "integer",
          "description": "Indicate the number of lines the widget can hold"
        },
        "dbValue": {
          "$ref": "#/definitions/propDbValueDefForDataMapper"
        },
        "maxLength": {
          "type": "integer",
          "description": "maximum length of the object"
        },
        "labelPosition": {
          "enum": [
            "PROPERTY_LABEL_AT_TOP",
            "PROPERTY_LABEL_AT_SIDE",
            "PROPERTY_LABEL_AT_RIGHT"
          ],
          "description": "Decide the postion of the label on the panel"
        },
        "isEditable": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "If it is set , user can edit the property"
        },
        "isArray": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "the property can hold the multiple data "
        },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "dispValue": {
          "type": "string",
          "description": "display value of the view model proeprty"
        },
        "vertical": {
          "type": "boolean",
          "description": "Vertical postioning of the radio buttons"
        },
        "validationCriteria": {
          "type": "array",
          "items": {
            "type": "object",
            "required": [
              "condition",
              "value"
            ],
            "additionalProperties": false,
            "properties": {
              "condition": {
                "type": "string"
              },
              "value": {
                "type": "string"
              }
            }
          }
        },
        "patterns": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "preferredPattern": {
          "type": "string"
        },
        "isRichText": {
          "type": "boolean"
        }
      },
      "anyOf": [
        {
          "required": [
            "name"
          ]
        },
        {
          "required": [
            "displayName"
          ]
        }
      ],
      "required": [
        "type",
        "value",
        "displayValue"
```

```
      ]
    },
    "propDbValueDefForDataMapper": {
      "oneOf": [
        {
          "type": "string"
        },
        {
          "type": "boolean"
        },
        {
          "type": "array"
        }
      ]
    },
    "viewModelPropDef": {
      "additionalProperties": false,
      "type": [
        "object",
        "array"
      ],
      "properties": {
        "displayName": {
          "oneOf": [
            {
              "type": "string",
              "description": "Display name on the DOM"
            },
            {
              "$ref": "#/definitions/paramLocalizationDef"
            }
          ]
        },
        "type": {
          "enum": [
            "STRING",
            "BOOLEAN",
            "DATETIME",
            "",
            "String",
            "INTEGER",
            "DOUBLE",
            "DATE"
          ],
          "description": "PreDefine type of the object. Valid type must be
from above list"
        },
        "isRequired": {
          "enum": [
            "true",
            "false"
          ],
          "description": "If isRequired is true , it should be compulsory
property to fill by user"
        },
        "requiredText": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "oneOf": [
            {
              "type": "string",
              "description": "Hint text to the user"
            },
            {
              "$ref": "#/definitions/paramLocalizationDef"
            }
          ]
        },
        "propertyRadioTrueText": {
          "oneOf": [
            {
              "type": "string",
              "description": "String displayed on the radio box of the
radio is selected"
            },
            {
              "$ref": "#/definitions/paramLocalizationDef"
            }
          ]
        },
        "propertyRadioFalseText": {
          "oneOf": [
            {
              "type": "string",
              "description": "String displayed on the radio box of the
radio is not selected"
            },
            {
              "$ref": "#/definitions/paramLocalizationDef"
            }
          ]
        },
        "hasLov": {
          "type": "boolean",
          "description": "If the object has more than one object to hold ,
this value can be true"
        },
        "numberOfLines": {
          "type": "integer",
          "description": "Indicate the number of lines the widget can hold"
        },
        "dbValue": {
          "$ref": "#/definitions/propDbValueDef"
        },
        "maxLength": {
          "type": "integer",
          "description": "maximum length of the object"
        },
        "labelPosition": {
          "enum": [
            "PROPERTY_LABEL_AT_TOP",
            "PROPERTY_LABEL_AT_SIDE",
            "PROPERTY_LABEL_AT_RIGHT"
          ],
```


Unpublished work. © 2025 Siemens

```
        "description": "Decide the postion of the label on the panel"
      },
      "isEditable": {
        "type": [
          "boolean",
          "string"
        ],
        "description": "If it is set , user can edit the property"
      },
      "isArray": {
        "type": [
          "boolean",
          "string"
        ],
        "description": "the property can hold the multiple data "
      },
      "dispValue": {
        "oneOf": [
          {
            "type": "string",
            "description": "display value of the view model proeprty"
          },
          {
            "$ref": "#/definitions/paramLocalizationDef"
          }
        ]
      },
      "displayValue": {
        "oneOf": [
          {
            "type": "string",
            "description": "display value of the view model proeprty"
          },
          {
            "$ref": "#/definitions/paramLocalizationDef"
          }
        ]
      },
      "defaultValues": {
        "type": "array",
        "items": {
          "type": "string",
          "description": "default value of the view model property"
        }
      },
      "uiValue": {
        "$ref": "#/definitions/propDbValueDef"
      },
      "vertical": {
        "type": "boolean",
        "description": "Vertical postioning of the radio buttons"
      },
      "validationCriteria": {
        "type": "array",
        "items": {
          "type": "object",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "required": [
              "condition",
              "value"
            ],
            "additionalProperties": false,
            "properties": {
              "condition": {
                "type": "string"
              },
              "value": {
                "type": "string"
              }
            }
          }
        },
        "iconName": {
          "type": "string",
          "description": "name of the icon of command"
        },
        "tooltip": {
          "type": "string",
          "description": "name of the tooltip of command"
        },
        "view": {
          "type": "string",
          "description": "view name of the extended tooltip"
        },
        "extendedTooltipContent": {
          "type": "string",
          "description": "Text content of the tooltip which want ballon
popup layout"
        },
        "commandType": {
          "type": "string",
          "description": "command type of the command"
        },
        "destPanelId": {
          "type": "string",
          "description": "In navigation , refers to the destination panel
id"
        },
        "recreatePanel": {
          "type": "boolean",
          "description": "In navigation , it decide to recreate the panel
after navigation"
        },
        "patterns": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "preferredPattern": {
          "type": "string"
        },
        "isRichText": {
```

```
          "type": "boolean"
        },
        "isModifiable": {
          "type": [
            "boolean",
            "string"
          ],
          "description": " true if the user should have the ability to
change the property's value. false if the value is read-only."
        },
        "isAutoAssignable": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "   true if the property's value can/should be
assigned automatically by server.   flse if the property's value is not
normally assigned/controlled by server."
        },
        "isSelectOnly": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "'isSelectOnly' state of ViewModelProperty.This
property allows the selected lov entry to be select only, i.e, user cannot
type in the widget after   selecting the lov entry. This change is to
support isSelectOnly in aw-widget used as a list box."
        },
        "referenceTypeName": {
          "type": "string",
          "description": "name of typed reference property"
        },
        "oldValue": {
          "type": "string"
        },
        "oldValues": {
          "type": "array"
        },
        "value": {
          "$ref": "#/definitions/propDbValueDef"
        },
        "dataProvider": {
          "type": "string"
        },
        "isEnabled": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "If isEnabled is true, the property available for
modify and default value is true."
        },
        "maxArraySize": {
          "type": "integer",
          "description": "maximum array length for that property"
```



Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        },
        "renderingHint": {
          "type": "string",
          "description": "depending on the type, this string indicates
some variation in how the   property's value should be displayed (e.g.
For 'BOOLEAN' type, valid values   include 'radiobutton','togglebutton',
'checkbox'. For 'STRING' type, valid   values include 'label',
'textbox','textfield', 'textarea', 'longtext')"
        },
        "autofocus": {
          "type": [
            "boolean",
            "string"
          ],
          "description": "defines whether the widget needs to be
autofocused or not"
        },
        "condition": {
          "type": "string",
          "description": "the name of the condition in validation criteria."
        }
      }
    },
    "propDbValueDef": {
      "oneOf": [
        {
          "oneOf": [
            {
              "type": "string"
            },
            {
              "$ref": "#/definitions/paramLocalizationDef"
            }
          ]
        },
        {
          "type": "boolean"
        },
        {
          "type": "array"
        }
      ]
    },
    "paramLocalizationDef": {
      "type": "object",
      "properties": {
        "text": {
          "type": "string"
        },
        "params": {
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      },
```

```
      "required": [
        "text",
        "params"
      ]
    },
    "messageDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Messages displayed on the page/panel",
      "properties": {
        "messageType": {
          "type": "string",
          "description": "Type of the messages it can either INFO ,WARNING,
ERROR"
        },
        "messageText": {
          "type": "string",
          "description": "Message text on the messages"
        },
        "messageKey": {
          "type": "string",
          "description": "Unique key of the messages"
        },
        "messageTextParams": {
          "type": "array",
          "description": "any runtime variable that can be added messages",
          "items": {
            "type": "string"
          }
        },
        "messageTextLinks": {
          "type": "array",
          "description": "any runtime variable that can be added messages
and rendered as aw-link element",
          "items": {
            "type": "object"
          }
        },
        "messageData": {
          "type": "object",
          "description": "Any runtime string on messages"
        },
        "expression": {
          "type": "object",
          "description": "Regular expression form for the condition"
        },
        "navigationOptions": {
          "type": "array",
          "description": "Navigation parameters on routing URL",
          "items": {
            "type": "object",
            "required": [
              "text"
            ],
            "additionalProperties": false,
            "properties": {
```


Unpublished work. © 2025 Siemens

```
              "option": {
                "type": "string"
              },
              "text": {
                "type": "string"
              },
              "action": {
                "type": "string"
              }
            }
          }
        },
        "message": {
          "type": "object"
        }
      },
      "required": [
        "messageType"
      ]
    },
    "functionsDef": {
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "additionalProperties": false,
          "type": "object",
          "description": "Js function support for the view model",
          "properties": {
            "functionName": {
              "type": "string"
            },
            "parameters": {
              "type": "array",
              "items": {
                "type": "string"
              }
            },
            "deps": {
              "type": "string"
            }
          },
          "required": [
            "functionName"
          ]
        }
      }
    },
    "viewModelObjectDef": {
      "type": "object",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "uid": {
```

```
              "type": "string",
              "description": "Unique ID of the object"
            },
            "props": {
              "type": "object",
              "description": "View Model Properties of the object"
            },
            "selected": {
              "type": "boolean",
              "description": "For radio or check box properties , if true
shows selected"
            },
            "type": {
              "type": "string",
              "description": "Type of the object"
            }
          }
        }
      }
    },
    "conditionOrBoolDef": {
      "oneOf": [
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "condition": {
              "type": [
                "string",
                "boolean"
              ],
              "description": "Condition expression that evaluates to :
'true','false',true or false"
            }
          }
        },
        {
          "type": "boolean"
        }
      ]
    }
  },
  "__defs": []
}
```

## conditionOrBoolDef Definition (ReusableSchemaDefs)

This definition must satisfy *one* of the following constraints:

- it must be set to a **ConditionOrBoolDefObject1** value
- it must be set to a boolean value


Siemens Web Framework
Unpublished work. © 2025 Siemens

#### Referenced Objects

The following objects are referenced within this definition.

#### ConditionOrBoolDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string OR boolean | | Condition expression that evaluates to : 'true','false',true or false |

## dataMapperEachPropDef Definition (ReusableSchemaDefs)

This definition must satisfy *any* of the following constraints:

- the following properties must be specified: **name**
- the following properties must be specified: **displayName**

## fieldDef Definition (ReusableSchemaDefs)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `fieldName` | string | Yes | |
| `fieldValue` | string | Yes | |

## functionsDef Definition (ReusableSchemaDefs)

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | FunctionsDefPatternObject | | Js function support for the view model |

#### Referenced Objects

The following objects are referenced within this definition.

#### FunctionsDefPatternObject

Js function support for the view model

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `functionName` | string | Yes | |
| `parameters` | Arraystring | | |
| `deps` | string | | |

## legacyConditionsDef Definition (ReusableSchemaDefs)

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_().,]*$` | LegacyConditionsDefPatternAltObject | | |

#### Referenced Objects

The following objects are referenced within this definition.

#### LegacyConditionsDefPatternAltObject

*No description specified.*

This object can contain the following properties:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| `$eq` | string OR integer OR boolean OR null OR array | | |
| `$lt` | integer | | |
| `$lte` | integer | | |
| `$gt` | integer | | |
| `$gte` | integer | | |
| `$ne` | string OR integer OR boolean OR array | | |
| `$neq` | string OR integer OR boolean OR array | | |
| `$in` | array | | |
| `$notin` | array | | |
| `$isNull` | boolean | | |
| `$notNull` | boolean | | |
| `$vlookup` | VlookupObject1 | | |
| `$regexp` | string | | |
| `$notinrange` | Arraystring OR integer | | |
| `$notinregexp` | string | | |
| `$typeOf` | string | | |

This value must satisfy *one* of the following constraints:

- the following properties must be specified: **$eq**
- the following properties must be specified: **$lt**
- the following properties must be specified: **$lte**
- the following properties must be specified: **$gt:**
- the following properties must be specified: **$gte**
- the following properties must be specified: **$ne**
- the following properties must be specified: **$neq**
- the following properties must be specified: **$in**
- the following properties must be specified: **$notin**
- the following properties must be specified: **$isNull**
- the following properties must be specified: **$notNull**
- the following properties must be specified: **$vlookup**

- the following properties must be specified: **$regexp**
- the following properties must be specified: **$notinrange**
- the following properties must be specified: **$notinregexp**
- the following properties must be specified: **$typeOf**

#### VlookupObject1

*No description specified.*

## messageDef Definition (ReusableSchemaDefs)

Messages displayed on the page/panel

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `messageType` | string | Yes | Type of the messages it can either INFO ,WARNING, ERROR |
| `messageText` | string | | Message text on the messages |
| `messageKey` | string | | Unique key of the messages |
| `messageTextParams` | Arraystring | | any runtime variable that can be added messages |
| `messageTextLinks` | Array**MessageTextLinksObject** | | any runtime variable that can be added messages and rendered as aw-link element |
| `messageData` | **MessageDataObject** | | Any runtime string on messages |
| `expression` | **ExpressionObject** | | Regular expression form for the condition |
| `navigationOptions` | Array**NavigationOptionsObject** | | Navigation parameters on routing URL |
| `message` | **MessageObject** | | |

#### Referenced Objects

The following objects are referenced within this definition.


Unpublished work. © 2025 Siemens

#### ExpressionObject

Regular expression form for the condition

#### MessageDataObject

Any runtime string on messages

#### MessageObject

*No description specified.*

#### MessageTextLinksObject

*No description specified.*

#### NavigationOptionsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `option` | string | | |
| `text` | string | Yes | |
| `action` | string | | |

## paramLocalizationDef Definition (ReusableSchemaDefs)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | Yes | |
| `params` | Arraystring | Yes | |

## propDbValueDef Definition (ReusableSchemaDefs)

This definition must satisfy *one* of the following constraints:

- This value must satisfy *one* of the following constraints: it must be set to a string value, it must be set to a **paramLocalizationDef** value.
- it must be set to a boolean value
- it must be set to a array value

## propDbValueDefForDataMapper Definition (ReusableSchemaDefs)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a boolean value
- it must be set to a array value

## structDef Definition (ReusableSchemaDefs)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `structName` | string | Yes | |
| `structValue` | Array**fieldDef** | Yes | |

## viewModelObjectDef Definition (ReusableSchemaDefs)

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | **ViewModelObjectDefPatternObject** | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Referenced Objects

The following objects are referenced within this definition.

### PropsObject

View Model Properties of the object

### ViewModelObjectDefPatternObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `uid` | string | | Unique ID of the object |
| `props` | PropsObject | | View Model Properties of the object |
| `selected` | boolean | | For radio or check box properties , if true shows selected |
| `type` | string | | Type of the object |

## viewModelPropDef Definition (ReusableSchemaDefs)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `displayName` | DisplayNameAltObject | | |
| `type` | | | It can be set to one of the following values: `STRING`, `BOOLEAN`, `DATETIME`, ``, `String`, `INTEGER`, `DOUBLE`, `DATE`. PreDefine type of the object. Valid type must be from above list |
| `isRequired` | | | It can be set to one of the following values: `true`, `false`. If isRequired is true , it should |

| | | | |
|---|---|---|---|
| | | | be compulsory property to fill by user |
| requiredText | RequiredTextAltObject | | |
| propertyRadioTrueText | PropertyRadioTrueTextAltObject | | |
| propertyRadioFalseText | PropertyRadioFalseTextAltObject | | |
| hasLov | boolean | | If the object has more than one object to hold , this value can be true |
| numberOfLines | integer | | Indicate the number of lines the widget can hold |
| dbValue | propDbValueDef | | |
| maxLength | integer | | maximum length of the object |
| labelPosition | | | It can be set to one of the following values: PROPERTY_LABEL_AT_TOP, PROPERTY_LABEL_AT_SIDE, PROPERTY_LABEL_AT_RIGHT. Decide the postion of the label on the panel |
| isEditable | boolean OR string | | If it is set , user can edit the property |
| isArray | boolean OR string | | the property can hold the multiple data |
| dispValue | DispValueAltObject | | |
| displayValue | DisplayValueAltObject | | |
| defaultValues | Arraystring | | |
| uiValue | propDbValueDef | | |
| vertical | boolean | | Vertical postioning of the radio buttons |
| validationCriteria | ArrayValidationCriteriaObject | | |
| iconName | string | | name of the icon of command |
| tooltip | string | | name of the tooltip of command |
| view | string | | view name of the extended tooltip |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `extendedTooltipContent` | string | | Text content of the tooltip which want ballon popup layout |
| `commandType` | string | | command type of the command |
| `destPanelId` | string | | In navigation , refers to the destination panel id |
| `recreatePanel` | boolean | | In navigation , it decide to recreate the panel after navigation |
| `patterns` | Arraystring | | |
| `preferredPattern` | string | | |
| `isRichText` | boolean | | |
| `isModifiable` | boolean OR string | | true if the user should have the ability to change the property's value. false if the value is read-only. |
| `isAutoAssignable` | boolean OR string | | true if the property's value can/should be assigned automatically by server. flse if the property's value is not normally assigned/controlled by server. |
| `isSelectOnly` | boolean OR string | | 'isSelectOnly' state of ViewModelProperty.This property allows the selected lov entry to be select only, i.e, user cannot type in the widget after selecting the lov entry. This change is to support isSelectOnly in aw-widget used as a list box. |
| `referenceTypeName` | string | | name of typed reference property |
| `oldValue` | string | | |
| `oldValues` | array | | |
| `value` | **propDbValueDef** | | |
| `dataProvider` | string | | |

| | | | |
|---|---|---|---|
| `isEnabled` | boolean OR string | | If isEnabled is true, the property available for modify and default value is true. |
| `maxArraySize` | integer | | maximum array length for that property |
| `renderingHint` | string | | depending on the type, this string indicates some variation in how the property's value should be displayed (e.g. For 'BOOLEAN' type, valid values include 'radiobutton','togglebutton', 'checkbox'. For 'STRING' type, valid values include 'label', 'textbox','textfield', 'textarea', 'longtext') |
| `autofocus` | boolean OR string | | defines whether the widget needs to be autofocused or not |
| `condition` | string | | the name of the condition in validation criteria. |

### Referenced Objects

The following objects are referenced within this definition.

#### DispValueAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **paramLocalizationDef** value

#### DisplayNameAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a string value


Unpublished work. © 2025 Siemens

- it must be set to a **paramLocalizationDef** value

#### DisplayValueAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **paramLocalizationDef** value

#### PropertyRadioFalseTextAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **paramLocalizationDef** value

#### PropertyRadioTrueTextAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **paramLocalizationDef** value

#### RequiredTextAltObject

*No description specified*.

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **paramLocalizationDef** value

### ValidationCriteriaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | Yes | |
| `value` | string | Yes | |

## SecondaryWorkareaTabs Schema

This configuration file can be used to define tabs to be displayed in the Secondary Work Area.

This schema must satisfy *one* of the following constraints:

- it must be set to a **SecondaryWorkareaTabsAltObject1** value
- it must be set to a Array**tabDef** value

### Definitions

This schema includes the following definitions:

- **conditionDef**
- **conditionOrBoolDef**
- **expressionStringOrObjectDef**
- **sourceStringOrObjectDef**
- **tabDef**
- **tabNameDef**

### Referenced Objects

The following objects are referenced within this schema.

### ConditionsObject1

Conditions provide live logic evaluation and they can evaluate to true or false.

You can define properties on this object with a name matching one of the following patterns:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | conditionDef | | |

### SecondaryWorkareaTabsAltObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| schema | string | | The version of the JSONSchema this file complies with |
| $schema | string | | The schema file for which the current file is compliant to. |
| tabs | TabsObject1 | | The definition of a tab. |
| conditions | ConditionsObject1 | | Conditions provide live logic evaluation and they can evaluate to true or false. |

This value must satisfy *one* of the following constraints:

- the following properties must be specified: **schema**, **tabs**
- the following properties must be specified: **$schema**, **tabs**

### TabsObject1

The definition of a tab.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | tabDef | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeSecondaryWorkareaTabsSchema-1.0.0.json",
  "description": "This configuration file can be used to define tabs to be
displayed in the Secondary Work Area.",
  "definitions": {
    "sourceStringOrObjectDef": {
      "oneOf": [
```

```
        {
          "type": "string"
        },
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "$adapt": {
              "type": "string",
              "description": "Indicates that the resulting source should be
adapted before evaluating the query."
            }
          }
        }
      ]
    },
    "expressionStringOrObjectDef": {
      "oneOf": [
        {
          "type": "object",
          "description": "Expression can be defined as object with the
support of below operators.",
          "additionalProperties": false,
          "properties": {
            "$source": {
              "$ref": "#/definitions/sourceStringOrObjectDef",
              "description": "Indicates the reference on data context node
to be used as starting point for evaluation. For ex. 'ctx.selected'"
            },
            "$query": {
              "type": "object",
              "description": "Defines the query to be executed on the
source"
            },
            "$and": {
              "type": "array",
              "description": "Indicates logical 'AND' of each query result"
            },
            "$or": {
              "type": "array",
              "description": "Indicates logical 'OR' of each query result"
            },
            "$all": {
              "type": "array",
              "description": "Indicates logical 'ALL' of each query result"
            },
            "$eventSource": {
              "type": "string",
              "description": "This operator lets user specify the viewId or
viewName or 'self' to write a condition to enable the events to be filtered
out based on whether or not it is triggered from the same viewModel."
            },
            "$not": {
              "type": "object",
              "description": "Indicates logical 'NOT' of a query result"
            }
```


Unpublished work. © 2025 Siemens

```
          }
        },
        {
          "type": "string",
          "description": "Expression as simple string: 'expression':
'ctx.mselected && ctx.mselected.length > 1'"
        }
      ]
    },
    "conditionDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Expressions can be defined as simple strings or
object.",
      "properties": {
        "expression": {
          "$ref": "#/definitions/expressionStringOrObjectDef"
        }
      },
      "required": [
        "expression"
      ]
    },
    "conditionOrBoolDef": {
      "oneOf": [
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "condition": {
              "type": [
                "string",
                "boolean"
              ],
              "description": "Condition expression that evaluates to :
'true','false',true or false"
            }
          }
        },
        {
          "type": "boolean"
        },
        {
          "$ref": "#/definitions/expressionStringOrObjectDef"
        }
      ]
    },
    "tabNameDef": {
      "oneOf": [
        {
          "type": "string",
          "description": "The Name of the tab."
        },
        {
          "type": "object",
          "additionalProperties": false,
```

```
          "properties": {
            "source": {
              "type": "string",
              "description": "The path to an i18n file containing the key
corresponding to the tab name, including the containing i18n folder."
            },
            "key": {
              "type": "string",
              "description": "The key of the string within the i18n message
file to used for the tab name."
            }
          },
          "required": [
            "source",
            "key"
          ]
        }
      ]
    },
    "tabDef": {
      "type": "object",
      "additionalProperties": false,
      "propertyNames": {
        "enum": [
          "name",
          "priority",
          "view",
          "visibleWhen",
          "method",
          "deps",
          "hasChildTabs",
          "pageId",
          "data",
          "tabKey",
          "id"
        ]
      },
      "properties": {
        "name": {
          "description": "The name of the tab to be displayed in the
Secondary Work Area.",
          "$ref": "#/definitions/tabNameDef"
        },
        "priority": {
          "description": "The relative priority order to render tabs in
ascending order.",
          "type": "number"
        },
        "view": {
          "description": "The component name to be rendered in the
Secondary Work Area when the tab is selected.",
          "type": "string"
        },
        "visibleWhen": {
          "description": "The name of the declarative condition which will
determine when the associated template is visible.",
```


Unpublished work. © 2025 Siemens

```
          "$ref": "#/definitions/conditionOrBoolDef"
        },
        "method": {
          "description": "INTERNAL USE ONLY.",
          "type": "string"
        },
        "deps": {
          "description": "INTERNAL USE ONLY.",
          "type": "string"
        },
        "hasChildTabs": {
          "description": "INTERNAL USE ONLY.",
          "type": "boolean"
        },
        "pageId": {
          "description": "The unique ID with which tab will be identified.
This is optional and Secondary Work Area will create pageID with tab
indexes if this is not specified.",
          "type": "string"
        },
        "data": {
          "description": "The view can refer to data and view state,
defined in the data section. The braces '{{  }}' allow declarative binding
to the live data.",
          "type": "object"
        },
        "tabKey": {
          "description": "The key using which active tab will be identified
or stored. If tabKey is not provided then view will be used as tabKey.",
          "type": "string"
        },
        "id": {
          "description": "The unique identifier for the tab.",
          "type": "string"
        }
      },
      "required": [],
      "oneOf": [
        {
          "properties": {
            "hasChildTabs": {
              "type": "boolean",
              "const": true
            }
          },
          "required": [
            "hasChildTabs"
          ]
        },
        {
          "not": {
            "required": [
              "hasChildTabs"
            ]
          },
          "required": [
```

```
            "view"
          ]
        }
      ]
    }
  },
  "oneOf": [
    {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "schema": {
          "description": "The version of the JSONSchema this file complies
with",
          "type": "string"
        },
        "$schema": {
          "description": "The schema file for which the current file is
compliant to.",
          "type": "string"
        },
        "tabs": {
          "type": "object",
          "description": "The definition of a tab.",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "$ref": "#/definitions/tabDef"
            }
          }
        },
        "conditions": {
          "description": "Conditions provide live logic evaluation and they
can evaluate to true or false.",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "$ref": "#/definitions/conditionDef"
            }
          }
        }
      },
      "oneOf": [
        {
          "required": [
            "schema",
            "tabs"
          ]
        },
        {
          "required": [
            "$schema",
            "tabs"
          ]
        }
      ]
    },
    {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "type": "array",
      "items": {
        "type": "object",
        "$ref": "#/definitions/tabDef"
      }
    }
  ],
  "__defs": []
}
```

## conditionDef Definition (SecondaryWorkareaTabs)

Expressions can be defined as simple strings or object.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `expression` | expressionStringOrObjectDef | Yes | |

## conditionOrBoolDef Definition (SecondaryWorkareaTabs)

This definition must satisfy *one* of the following constraints:

- it must be set to a ConditionOrBoolDefObject1 value
- it must be set to a boolean value
- it must be set to a expressionStringOrObjectDef value

### Referenced Objects

The following objects are referenced within this definition.

#### ConditionOrBoolDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string OR boolean | | Condition expression that evaluates to : 'true','false',true or false |

## expressionStringOrObjectDef Definition (SecondaryWorkareaTabs)

This definition must satisfy *one* of the following constraints:

- it must be set to a **ExpressionStringOrObjectDefObject1** value
- it must be set to a string value

### Referenced Objects

The following objects are referenced within this definition.

#### ExpressionStringOrObjectDefObject1

Expression can be defined as object with the support of below operators.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$source` | **sourceStringOrObjectDef** | | Indicates the reference on data context node to be used as starting point for evaluation. For ex. 'ctx.selected' |
| `$query` | **QueryObject** | | Defines the query to be executed on the source |
| `$and` | array | | Indicates logical 'AND' of each query result |
| `$or` | array | | Indicates logical 'OR' of each query result |
| `$all` | array | | Indicates logical 'ALL' of each query result |
| `$eventSource` | string | | This operator lets user specify the viewId or viewName or 'self' to write a condition to enable the events to be filtered out based on whether or not it is triggered from the same viewModel. |
| `$not` | **NotObject** | | Indicates logical 'NOT' of a query result |

#### NotObject

Indicates logical 'NOT' of a query result


Siemens Web Framework
Unpublished work. © 2025 Siemens

#### QueryObject

Defines the query to be executed on the source

## sourceStringOrObjectDef Definition (SecondaryWorkareaTabs)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **SourceStringOrObjectDefObject1** value

#### Referenced Objects

The following objects are referenced within this definition.

#### SourceStringOrObjectDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$adapt` | string | | Indicates that the resulting source should be adapted before evaluating the query. |

## tabDef Definition (SecondaryWorkareaTabs)

This definition must satisfy *one* of the following constraints:

- it must be set to a **TabDefObject1** value

#### Referenced Objects

The following objects are referenced within this definition.

#### TabDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `hasChildTabs` | boolean | Yes | It must be set to `true`. |

### tabNameDef Definition (SecondaryWorkareaTabs)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a TabNameDefObject1 value

#### Referenced Objects

The following objects are referenced within this definition.

#### TabNameDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `source` | string | Yes | The path to an i18n file containing the key corresponding to the tab name, including the containing i18n folder. |
| `key` | string | Yes | The key of the string within the i18n message file to used for the tab name. |

## Site Schema

This is the schema the declarative site definition (in JSON format) must adhere to. The Apollo audit check validates all site JSON files against this schema. For explanation of the keywords of this schema, please visit https://ajv.js.org/keywords.html

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `kits` | Arraystring | | Array of the kits depend on the current site solution. It must contain at least 1 unique item. |
| `name` | string | Yes | Site name |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `solutionName` | string | | Solution name |
| `assetsDir` | string | | path of the assets directory |
| `link` | boolean | | Set to false to skip setting symlink in out directory. Defaults to true. |
| `locale` | Arraystring | | Array of the installed locales |
| `ignoreMissingKits` | boolean | | Set to true to ignore any referenced kits that are missing. |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeSiteSchema-1.0.0.json",
  "description": "This is the schema the declarative site  definition (in
JSON format) must adhere to. The Apollo audit check validates all site JSON
files against this schema. For explanation of the keywords of this schema,
please visit https://ajv.js.org/keywords.html",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "kits": {
      "type": "array",
      "description": "Array of the kits depend on the current site
solution",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "name": {
      "type": "string",
      "description": "Site name"
    },
    "solutionName": {
      "type": "string",
      "description": "Solution name "
    },
    "assetsDir": {
      "type": "string",
      "description": "path of the assets directory"
    },
    "link": {
      "type": "boolean",
      "description": "Set to false to skip setting symlink in out
directory. Defaults to true."
    },
    "locale": {
      "type": "array",
      "description": "Array of the installed locales",
```

```
      "items": {
        "type": "string"
      }
    },
    "ignoreMissingKits": {
      "type": "boolean",
      "description": "Set to true to ignore any referenced kits that are
missing."
    }
  },
  "required": [
    "name"
  ],
  "__defs": []
}
```

## SPLMTableViewModel Schema

This is the schema for validating View Models where SPLM Table is used. The Apollo audit check validates all View Model JSON files against this schema. For explanation of the keywords of this schema, please visit https://ajv.js.org/keywords.html

**Note**

This object *can* have additional properties.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `columnProviders` | ColumnProvidersObject | | |
| `grids` | GridsObject | | |

### Definitions

This schema includes the following definitions:

- columnProviderDef
- gridDef

### Referenced Objects

The following objects are referenced within this schema.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## ColumnProvidersObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| \b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b | columnProviderDef | | |

## GridsObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| \b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b | gridDef | | |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "description": "This is the schema for validating View Models where SPLM
Table is used. The Apollo audit check validates all View Model JSON files
against this schema. For explanation of the keywords of this schema, please
visit https://ajv.js.org/keywords.html",
  "definitions": {
    "columnProviderDef": {
      "additionalProperties": true,
      "type": "object",
      "properties": {
        "frozenColumnIndex": {
          "type": "number"
        },
        "sortCriteria": {
          "type": [
            "array",
            "string"
          ]
        },
        "columns": {
          "additionalProperties": false,
          "type": [
            "array",
            "string"
          ],
```

```
            "items": {
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "cellRenderers": {
                  "type": "object"
                },
                "displayName": {
                  "type": "string"
                },
                "dataType": {
                  "type": "string"
                },
                "enableColumnHiding": {
                  "type": "boolean"
                },
                "enableColumnMenu": {
                  "type": "boolean"
                },
                "enableColumnMoving": {
                  "type": "boolean"
                },
                "enableColumnResizing": {
                  "type": "boolean"
                },
                "enableSorting": {
                  "type": "boolean"
                },
                "headerTooltip": {
                  "type": "boolean"
                },
                "hiddenFlag": {
                  "type": "boolean"
                },
                "isFilteringEnabled": {
                  "type": "boolean"
                },
                "isTableCommand": {
                  "type": "boolean"
                },
                "isTreeNavigation": {
                  "type": "boolean"
                },
                "minWidth": {
                  "type": "number"
                },
                "maxWidth": {
                  "type": "number"
                },
                "name": {
                  "type": "string"
                },
                "pinnedLeft": {
                  "type": "boolean"
                },
                "propertyName": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "type": "string"
              },
              "renderingHint": {
                "type": "string"
              },
              "typeName": {
                "type": "string"
              },
              "visible": {
                "type": "boolean"
              },
              "width": {
                "type": "number"
              },
              "modifiable": {
                "type": "boolean"
              },
              "parentName": {
                "type": "string"
              },
              "columnNumber": {
                "type": "string"
              },
              "columnHeaderTranspose": {
                "type": "boolean"
              }
            }
          }
        }
      }
    },
    "gridDef": {
      "additionalProperties": false,
      "type": "object",
      "required": [
        "dataProvider"
      ],
      "properties": {
        "addIconColumn": {
          "type": "boolean"
        },
        "columnProvider": {
          "type": "string"
        },
        "dataProvider": {
          "type": "string"
        },
        "deps": {
          "type": "string"
        },
        "enableArrangeMenu": {
          "type": "boolean"
        },
        "pinIconColumn": {
          "type": "boolean"
        },
```

```
        "propertyProvider": {
          "type": "string"
        },
        "gridOptions": {
          "additionalProperties": false,
          "type": "object",
          "properties": {
            "enableCellHeaderSelection": {
              "type": "boolean"
            },
            "enableColumnSelection": {
              "type": "boolean"
            },
            "enableExpandAndPaginationInEdit": {
              "type": "boolean"
            },
            "enableExpansionStateCaching": {
              "type": "boolean"
            },
            "enableDragAndDrop": {
              "type": "boolean"
            },
            "enableDynamicRowHeight": {
              "type": "boolean"
            },
            "enableGridMenu": {
              "type": "boolean"
            },
            "enableHeaderIcon": {
              "type": "boolean"
            },
            "enablePinning": {
              "type": [
                "boolean",
                "string"
              ]
            },
            "enableSaveSortCriteria": {
              "type": "boolean"
            },
            "enableSorting": {
              "type": "boolean"
            },
            "forceAutoSave": {
              "type": "boolean"
            },
            "headerHeight": {
              "type": [
                "string",
                "number"
              ]
            },
            "isFilteringEnabled": {
              "type": [
                "boolean",
                "string"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            ]
          },
          "maxRowsToShow": {
            "type": "number"
          },
          "rowHeight": {
            "type": "string"
          },
          "showContextMenu": {
            "type": "boolean"
          },
          "textBundle": {
            "type": "object",
            "properties": {
              "TwistieTooltipCollapsed": {
                "type": "string"
              },
              "TwistieTooltipExpanded": {
                "type": "string"
              }
            }
          },
          "transpose": {
            "type": "boolean"
          },
          "useStaticFirstCol": {
            "type": "boolean"
          },
          "enableHeaderResizing": {
            "type": "boolean"
          },
          "enableColumnNumbering": {
            "type": "boolean"
          },
          "flexLastColumn": {
            "type": "boolean"
          }
        }
      }
    }
  }
},
"type": "object",
"additionalProperties": true,
"properties": {
  "columnProviders": {
    "type": "object",
    "patternProperties": {
      "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
        "type": "object",
        "$ref": "#/definitions/columnProviderDef"
      }
    }
  },
  "grids": {
    "type": "object",
```

```
      "patternProperties": {
        "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
          "type": "object",
          "$ref": "#/definitions/gridDef"
        }
      }
    }
  },
  "__defs": []
}
```

## columnProviderDef Definition (SPLMTableViewModel)

*No description specified.*

**Note**
This object *can* have additional properties.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `frozenColumnIndex` | number | | |
| `sortCriteria` | array OR string | | |
| `columns` | array OR string | | |

## gridDef Definition (SPLMTableViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `addIconColumn` | boolean | | |
| `columnProvider` | string | | |
| `dataProvider` | string | Yes | |
| `deps` | string | | |
| `enableArrangeMenu` | boolean | | |
| `pinIconColumn` | boolean | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `propertyProvider` | string | | |
| `gridOptions` | GridOptionsObject | | |

### Referenced Objects

The following objects are referenced within this definition.

### GridOptionsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `enableCellHeaderSelection` | boolean | | |
| `enableColumnSelection` | boolean | | |
| `enableExpandAndPaginationInEdit` | boolean | | |
| `enableExpansionStateCaching` | boolean | | |
| `enableDragAndDrop` | boolean | | |
| `enableDynamicRowHeight` | boolean | | |
| `enableGridMenu` | boolean | | |
| `enableHeaderIcon` | boolean | | |
| `enablePinning` | boolean OR string | | |
| `enableSaveSortCriteria` | boolean | | |
| `enableSorting` | boolean | | |
| `forceAutoSave` | boolean | | |
| `headerHeight` | string OR number | | |
| `isFilteringEnabled` | boolean OR string | | |
| `maxRowsToShow` | number | | |
| `rowHeight` | string | | |
| `showContextMenu` | boolean | | |
| `textBundle` | TextBundleObject | | |
| `transpose` | boolean | | |

| | | | |
|---|---|---|---|
| `useStaticFirstCol` | boolean | | |
| `enableHeaderResizing` | boolean | | |
| `enableColumnNumbering` | boolean | | |
| `flexLastColumn` | boolean | | |

#### TextBundleObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `TwistieTooltipCollapsed` | string | | |
| `TwistieTooltipExpanded` | string | | |

## State Schema

Actions can be of different actionType (REST, JSFunction, JSFunctionAsync, Event, Command or TcSoaService). All actions have inputData which specifies the JSON input for the action and outputData for mapping the output response.

This schema must satisfy *any* of the following constraints:

- the following properties must be specified: **view**
- the following properties must be specified: **type**
- it must be set to a **StateObject1** value
- the following properties must be specified: **url**
- one of the following must be specified: **view** *or* **type** *or* **resolveActions**

### Definitions

This schema includes the following definitions:

- **conditionDef**
- **expressionStringOrObjectDef**
- **sourceStringOrObjectDef**


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Referenced Objects

The following objects are referenced within this schema.

## StateObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `abstract` | boolean | Yes | It must be set to `true`. |

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeStateSchema-1.0.0.json",
  "description": "Actions can be of different actionType (REST, JSFunction,
JSFunctionAsync, Event, Command or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
  "type": "object",
  "additionalProperties": false,
  "propertyNames": {
    "enum": [
      "abstract",
      "view",
      "type",
      "data",
      "parent",
      "url",
      "resolveActions",
      "params",
      "context",
      "visibleWhen",
      "reloadOnSearch",
      "dynamic"
    ]
  },
  "definitions": {
    "sourceStringOrObjectDef": {
      "oneOf": [
        {
          "type": "string"
        },
        {
          "type": "object",
          "additionalProperties": false,
```

```
          "properties": {
            "$adapt": {
              "type": "string",
              "description": "Indicates that the resulting source should be
adapted before evaluating the query"
            }
          }
        }
      ]
    },
    "expressionStringOrObjectDef": {
      "oneOf": [
        {
          "type": "object",
          "description": "Expression can be defined as object with the
support of below operators.",
          "additionalProperties": false,
          "properties": {
            "$source": {
              "$ref": "#/definitions/sourceStringOrObjectDef",
              "description": "Indicates the reference on data context node
to be used as starting point for evaluation. For ex. 'ctx.selected'"
            },
            "$query": {
              "type": "object",
              "description": "Defines the query to be executed on the
source"
            },
            "$and": {
              "type": "array",
              "description": "Indicates logical 'AND' of each query result"
            },
            "$or": {
              "type": "array",
              "description": "Indicates logical 'OR' of each query result"
            },
            "$all": {
              "type": "array",
              "description": "Indicates logical 'ALL' of each query result"
            },
            "$eventSource": {
              "type": "string",
              "description": "This operator lets user specify the viewId or
viewName or 'self' to write a condition to enable the events to be filtered
out based on whether or not it is triggered from the same viewModel."
            },
            "$not": {
              "type": "object",
              "description": "Indicates logical 'NOT' of a query result"
            }
          }
        },
        {
          "type": "string",
          "description": "Expression as simple string: 'expression':
'ctx.mselected && ctx.mselected.length > 1'"
```


Unpublished work. © 2025 Siemens

```
        }
      ]
    },
    "conditionDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Expressions can be defined as simple strings or
object.",
      "properties": {
        "expression": {
          "$ref": "#/definitions/expressionStringOrObjectDef"
        }
      },
      "required": [
        "expression"
      ]
    }
  },
  "properties": {
    "abstract": {
      "type": "boolean",
      "description": "Whether the state is abstract or not. Abstract states
can never be directly activated but can be used to provide inherited
properties (url, resolve, data, etc.) to child states."
    },
    "view": {
      "type": "string",
      "description": "The name of the component to be rendered when the
state is loaded."
    },
    "type": {
      "type": "string",
      "description": "Identifies a default framework component to be used
instead of the view. This property is deprecated in favor of **view**.",
      "enum": [
        "location",
        "subLocation",
        "parameter"
      ]
    },
    "data": {
      "type": "object",
      "description": "An arbitrary object used to define custom data
associated to the state. You can use expressions within double curly braces
({{ }}) to bind live data."
    },
    "parent": {
      "type": "string",
      "description": "The identifier of the parent state."
    },
    "url": {
      "type": [
        "null",
        "string"
      ],
      "description": "A URL fragment (with optional parameters) which is
```

```
used to match the browser location with this state."
    },
    "resolveActions": {
      "type": "object",
      "description": "The action to be invoked before the state is loaded
and its component is rendered."
    },
    "params": {
      "type": "object",
      "description": "A dictionary used to define optional state parameters
that can be passed via URL.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_-]*$": {
          "anyOf": [
            {
              "type": "object",
              "properties": {
                "type": {
                  "type": "string",
                  "description": "Specifies type of the parameter."
                },
                "value": {
                  "type": [
                    "string",
                    "boolean",
                    "array",
                    "integer"
                  ],
                  "description": "Sets the default value for the parameter."
                },
                "array": {
                  "type": "boolean",
                  "description": "Specifies whether the parameter is
expected to be an array."
                },
                "dynamic": {
                  "type": "boolean",
                  "description": "Specifies whether to reload the page when
parameter value is changed."
                }
              },
              "required": [],
              "additionalProperties": false
            },
            {
              "type": [
                "string",
                "null"
              ]
            }
          ]
        }
      }
    },
    "context": {
      "type": "object",
```


Unpublished work. © 2025 Siemens

```
      "description": "An object containing application contexts that will
be set when the state is loaded."
    },
    "visibleWhen": {
      "description": "The declarative condition which will determine when
the associated view is visible.",
      "oneOf": [
        {
          "$ref": "#/definitions/conditionDef"
        },
        {
          "type": "boolean"
        }
      ]
    },
    "reloadOnSearch": {
      "type": "boolean",
      "description": "Marks all query parameters as dynamic."
    },
    "dynamic": {
      "type": "boolean",
      "description": "When set to true, changes to the parameter values
will not cause the state to be entered/exited, the resolvers will not be
re-fetched, and the view will not be reloaded."
    }
  },
  "required": [],
  "anyOf": [
    {
      "required": [
        "view"
      ]
    },
    {
      "required": [
        "type"
      ]
    },
    {
      "properties": {
        "abstract": {
          "type": "boolean",
          "const": true
        }
      },
      "required": [
        "abstract"
      ]
    },
    {
      "required": [
        "url"
      ],
      "anyOf": [
        {
          "required": [
```

```
            "view"
          ]
        },
        {
          "required": [
            "type"
          ]
        },
        {
          "required": [
            "resolveActions"
          ]
        }
      ]
    }
  ],
  "__defs": []
}
```

## conditionDef Definition (State)

Expressions can be defined as simple strings or object.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `expression` | expressionStringOrObjectDef | Yes | |

## expressionStringOrObjectDef Definition (State)

This definition must satisfy *one* of the following constraints:

- it must be set to a ExpressionStringOrObjectDefObject1 value
- it must be set to a string value

### Referenced Objects

The following objects are referenced within this definition.

### ExpressionStringOrObjectDefObject1

Expression can be defined as object with the support of below operators.


Unpublished work. © 2025 Siemens

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$source` | sourceStringOrObjectDef | | Indicates the reference on data context node to be used as starting point for evaluation. For ex. 'ctx.selected' |
| `$query` | QueryObject | | Defines the query to be executed on the source |
| `$and` | array | | Indicates logical 'AND' of each query result |
| `$or` | array | | Indicates logical 'OR' of each query result |
| `$all` | array | | Indicates logical 'ALL' of each query result |
| `$eventSource` | string | | This operator lets user specify the viewId or viewName or 'self' to write a condition to enable the events to be filtered out based on whether or not it is triggered from the same viewModel. |
| `$not` | NotObject | | Indicates logical 'NOT' of a query result |

#### NotObject

Indicates logical 'NOT' of a query result

#### QueryObject

Defines the query to be executed on the source

## sourceStringOrObjectDef Definition (State)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a SourceStringOrObjectDefObject1 value

#### Referenced Objects

The following objects are referenced within this definition.

### SourceStringOrObjectDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$adapt` | string | | Indicates that the resulting source should be adapted before evaluating the query |

## States Schema

This schema is used to validate states.json files that are used to configure the states (routes) of your application.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | State | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeStatesSchema-1.0.0.json",
  "description": "This schema is used to validate states.json files that
are used to configure the states (routes) of your application.",
  "propertyNames": true,
  "type": "object",
  "minProperties": 1,
  "patternProperties": {
    "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
      "$ref": "declarativeStateSchema-1.0.0.json"
    }
  },
  "__defs": []
}
```

## SyncStrategy Schema

This is the schema the declarative sync strategy (in JSON format) must adhere to. The Apollo audit check validates all syncStrategy View Model JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**


Unpublished work. © 2025 Siemens

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with |
| `syncStrategies` | **SyncStrategiesObject** | Yes | Defines the sync configuration |
| `syncStrategyHandlers` | **SyncStrategyHandlersObject** | Yes | Defines the sync handler |
| `actions` | **ActionsObject** | | In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button. |
| `data` | **DataAltObject** | | The View can refer to data and view state, defined in the View Model data section. The braces '{{ }}' allow declarative binding to the live data. |
| `preferences` | Arraystring | | undefined. It must contain only unique items. |
| `functions` | **functionsDef** | | |
| `messages` | **MessagesObject** | | Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. |

| | | | | |
|---|---|---|---|---|
| | | | They can even leverage view model data and conditions. | |
| `conditions` | ConditionsObject | | Conditions provide live logic evaluation. They can evaluate to true/ false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |
| `onEvent` | ArrayOnEventObject | | The View Model can register to react to events by declaring an onEvent section. It can then conditionally execute a declarative action. | |

## Definitions

This schema includes the following definitions:

- actionDef
- sourceOrTargetDef
- syncHandlerDef
- syncStrategyDef

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the


Unpublished work. © 2025 Siemens

JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | actionDef | | |

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | Condition | | |

## CriteriaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `panelId` | string | | |
| `destPanelId` | string | | |

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | number OR string OR boolean | | |

## DataAltObject

The View can refer to data and view state, defined in the View Model data section. The braces '{{ }}' allow declarative binding to the live data.

This value must satisfy *any* of the following constraints:

- it must be set to a **DataObject1** value
- it must be set to a **DataObject2** value

## DataObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `objects` | viewModelObjectDef | | |

## DataObject2

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | viewModelPropDef | | |

## InputArgsObject

*No description specified.*

## MessagesObject

Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions.

You can define properties on this object with a name matching one of the following patterns:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | messageDef | | |

## OnEventObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| eventId | string | | The id of an event that the viewModel is interested to listen to. |
| criteria | CriteriaObject | | |
| eventSource | string | | (Optional) set it to 'current' to make event listeners only listen to events fired by the current view model |
| action | string | | An action that is defined in the actions section of the viewModel which would be executed only if the specified condition evaluates to true. |
| message | string | | |
| inputArgs | InputArgsObject | | |
| cacheEventData | boolean | | |

## SyncStrategiesObject

Defines the sync configuration

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | syncStrategyDef | | |

## SyncStrategyHandlersObject

Defines the sync handler

You can define properties on this object with a name matching one of the following patterns:


Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | syncHandlerDef | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeSyncStrategySchema-1.0.0.json",
  "description": "This is the schema the declarative sync strategy (in JSON
format) must adhere to. The Apollo audit check validates all syncStrategy
View Model JSON files against this schema. For explanation of the keywords
of this schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "sourceOrTargetDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Source or target between which the syncStrategy is to
be defined.",
      "properties": {
        "view": {
          "type": "string",
          "description": "Name of the view for which the sync Strategy is
to be defined"
        },
        "port": {
          "type": "string",
          "description": "Name of the port for which the sync Strategy is
to be defined"
        }
      },
      "required": [
        "view",
        "port"
      ]
    },
    "syncStrategyDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "The definition of the syncStrategy binding the ports
of the specified Source and Target Views",
      "properties": {
        "source": {
          "$ref": "#/definitions/sourceOrTargetDef"
        },
        "target": {
          "$ref": "#/definitions/sourceOrTargetDef"
        }
      },
      "required": [
        "source",
        "target"
      ]
    },
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
    "syncHandlerDef": {
      "additionalProperties": false,
      "type": "object",
      "properties": {
        "id": {
          "description": "The command Id for which the handler is
associated.",
          "type": "string"
        },
        "action": {
          "description": "The name of the declarative action
(REST,SOA,Event or JSFunction) to be executed.",
          "type": "string"
        },
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when this command handler is active for the associated command."
        }
      },
      "required": [
        "id",
        "activeWhen"
      ]
    },
    "actionDef": {
      "description": "Actions can be of different actionType (REST,
JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
      "oneOf": [
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "TcSoaService"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
TcSoaServiceDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "dataProvider"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
```

```
dataProviderActionDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Event"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
EventDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "RESTService"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
RESTServiceDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Copy"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
CopyDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Navigate"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
NavigateDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "popup"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
popupDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "JSFunctionAsync"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
JSFunctionAsyncDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "JSFunction"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
JSFunctionDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Edit"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
```

```
EditDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "batchJob"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
batchJobDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "GraphQL"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
GraphQLDef"
          },
          "else": false
        }
      ]
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
    },
    "syncStrategies": {
      "type": "object",
      "description": "Defines the sync configuration",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/syncStrategyDef"
        }
      }
    },
    "syncStrategyHandlers": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "type": "object",
      "description": "Defines the sync handler",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/syncHandlerDef"
        }
      }
    },
    "actions": {
      "type": "object",
      "description": "In the View Model, actions are declaratively
defined. Actions can be of different 'actionType' (REST, JSFunction,
JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which
specifies the JSON input for the action and 'outputData' for mapping the
output response. Actions can be used in dataProviders or UI elements such
as an aw-button.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/actionDef"
        }
      }
    },
    "data": {
      "type": "object",
      "description": "The View can refer to data and view state, defined in
the View Model data section. The braces '{{  }}' allow declarative binding
to the live data.",
      "anyOf": [
        {
          "properties": {
            "objects": {
              "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/viewModelObjectDef"
            }
          }
        },
        {
          "type": "object",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/viewModelPropDef"
            }
          }
        }
      ]
    },
    "preferences": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
```

```
    },
    "functions": {
      "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
functionsDef"
    },
    "messages": {
      "type": "object",
      "description": "Messages are declaratively defined in the View Model.
Messages are launched by events and actions. They cover information,
warning, error notifications. They give the user more deeper customization
options. They are localized. They can fire some actions. They can even
leverage view model data and conditions.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
messageDef"
        }
      }
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and
can be used for event handling (onEvent handler). The data value can
be live evaluated by condition. The conditions can refer to view model
state and can update view model state. They can leverage Boolean condition
expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    },
    "onEvent": {
      "type": "array",
      "description": "The View Model can register to react to events
by declaring an onEvent section. It can then conditionally execute a
declarative action.",
      "required": [
        "eventId",
        "action"
      ],
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "eventId": {
            "type": "string",
            "description": "The id of an event that the viewModel is
interested to listen to."
          },
          "criteria": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
```


Unpublished work. © 2025 Siemens

```
                "type": [
                  "number",
                  "string",
                  "boolean"
                ]
              }
            },
            "properties": {
              "panelId": {
                "type": "string"
              },
              "destPanelId": {
                "type": "string"
              }
            }
          },
          "eventSource": {
            "type": "string",
            "description": "(Optional) set it to 'current' to make event
listeners only listen to events fired by the current view model"
          },
          "action": {
            "type": "string",
            "description": "An action that is defined in the actions
section of the viewModel which would be executed only if the specified
condition evaluates to true."
          },
          "message": {
            "type": "string"
          },
          "inputArgs": {
            "type": "object"
          },
          "cacheEventData": {
            "type": "boolean"
          }
        }
      }
    }
  },
  "required": [
    "syncStrategies",
    "syncStrategyHandlers"
  ],
  "__defs": []
}
```

## actionDef Definition (SyncStrategy)

Actions can be of different actionType (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have inputData which specifies the JSON input for the action and outputData for mapping the output response.

This definition must satisfy *one* of the following constraints:

- if **actionType** is set to `TcSoaService` **TcSoaServiceDef**
- if **actionType** is set to `dataProvider` **dataProviderActionDef**
- if **actionType** is set to `Event` **EventDef**
- if **actionType** is set to `RESTService` **RESTServiceDef**
- if **actionType** is set to `Copy` **CopyDef**
- if **actionType** is set to `Navigate` **NavigateDef**
- if **actionType** is set to `popup` **popupDef**
- if **actionType** is set to `JSFunctionAsync` **JSFunctionAsyncDef**
- if **actionType** is set to `JSFunction` **JSFunctionDef**
- if **actionType** is set to `Edit` **EditDef**
- if **actionType** is set to `batchJob` **batchJobDef**
- if **actionType** is set to `GraphQL` **GraphQLDef**

## sourceOrTargetDef Definition (SyncStrategy)

Source or target between which the syncStrategy is to be defined.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `view` | string | Yes | Name of the view for which the sync Strategy is to be defined |
| `port` | string | Yes | Name of the port for which the sync Strategy is to be defined |

## syncHandlerDef Definition (SyncStrategy)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | The command Id for which the handler is associated. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `action` | string | | The name of the declarative action (REST,SOA,Event or JSFunction) to be executed. |
| `activeWhen` | conditionOrBoolDef | Yes | The name of the declarative condition which will determine when this command handler is active for the associated command. |

### syncStrategyDef Definition (SyncStrategy)

The definition of the syncStrategy binding the ports of the specified Source and Target Views

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `source` | sourceOrTargetDef | Yes | |
| `target` | sourceOrTargetDef | Yes | |

## TabsViewModel Schema

This is the schema the tabs View Model (in JSON format) must adhere to. The Apollo audit check validates all tabs View Model JSON files against this schema. For explanation of the keywords of this schema, please visit https://ajv.js.org/keywords.html

This object can contain the following properties:

| Name | Type | Required | Description | |
|---|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with | |
| `conditions` | ConditionsObject | | Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |
| `i18n` | I18nObject | | All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON | |

| | | | |
|---|---|---|---|
| | | | bundle which holds the localized text to be shown. |
| `tabs` | **TabsObject** | | Definition of a tab |
| `tabHandlers` | **TabHandlersObject** | | A tab can have many tab Handlers. However at any given time, there may at most be only one active tabHandler for a tab. A declarative 'activeWhen' condition which evaluates a boolean expression controls whether a handler is currently active. If more than one handler for a given tab evaluates to true, then the more 'specific' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the tab handler will determine tab visibility and What a tab will do when executed. |
| `tabPlacements` | **TabPlacementsObject** | | A tab can have many placements. A placement is the actual visual rendering of the display of the tab. |
| `actions` | **ActionsObject** | | In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in UI elements such as an aw-button. |
| `messages` | **MessagesObject** | | Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions. |
| `onEvent` | Array**OnEventObject** | | The View Model can register to react to events by declaring an onEvent section. It can then conditionally execute a declarative action. |

## Definitions

This schema includes the following definitions:


Unpublished work. © 2025 Siemens

- actionDef
- stringOrBoolDef
- tabDef
- tabHandlersDef
- tabPlacementsDef

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in UI elements such as an aw-button.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | actionDef | | |

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | Condition | | |

## CriteriaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| panelId | string | | |
| destPanelId | string | | |

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | number OR string OR boolean | | |

### EventDataObject

*No description specified.*

### I18nObject

All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | array OR string | | |

### InputArgsObject

*No description specified.*

### MessagesObject

Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions.

You can define properties on this object with a name matching one of the following patterns:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | messageDef | | |

## OnEventObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| eventId | string | | The id of an event that the viewModel is interested to listen to. |
| criteria | CriteriaObject | | |
| eventSource | string | | (Optional) set it to 'current' to make event listeners only listen to events fired by the current view model |
| action | string | | An action that is defined in the actions section of the viewModel which would be executed only if the specified condition evaluates to true. |
| message | string | | |
| inputArgs | InputArgsObject | | |
| cacheEventData | boolean | | |
| condition | string | | |
| eventData | EventDataObject | | |

## TabHandlersObject

A tab can have many tab Handlers. However at any given time, there may at most be only one active tabHandler for a tab. A declarative 'activeWhen' condition which evaluates a boolean expression controls whether a handler is currently active. If more than one handler for a given tab evaluates to true, then the more 'specific' condition (i.e. the condition with longer expression) is chosen to be active. By being active, the tab handler will determine tab visibility and What a tab will do when executed.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | tabHandlersDef | | |

### TabPlacementsObject

A tab can have many placements. A placement is the actual visual rendering of the display of the tab.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | tabPlacementsDef | | |

### TabsObject

Definition of a tab

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | tabDef | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeTabsViewModelSchema-1.0.0.json",
  "description": "This is the schema the tabs View Model (in JSON format)
must adhere to. The Apollo audit check validates all tabs View Model JSON
files against this schema. For explanation of the keywords of this schema,
please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "stringOrBoolDef": {
      "oneOf": [
        {
          "type": "string",
          "description": "A condition statement which would evaluate to
boolean.",
          "additionalProperties": false
        },
        {
          "type": "boolean"
        }
      ]
    },
    "tabDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "iconId": {
          "description": "The id of the tab's icon. It refers to the SVG
icon that is stored in the 'src/image' folder in the site location",
          "type": "string"
        },
        "title": {
          "oneOf": [
            {
              "description": "The title to display on the tooltip of the
tab. It should be either an i18n key or as just a non-18n string. Even
though schema doesn't enforces it, title is required, which could be either
string or object, so that functionality can work properly",
              "type": "string"
            },
            {
              "description": "The title to display on the tooltip of the
tab. It should be parametrized i18n key along with its params",
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "text": {
                  "type": "string"
                },
                "params": {
                  "type": "array",
                  "items": {
                    "type": "string"
                  }
                }
              }
            }
          ]
        },
        "description": {
          "oneOf": [
            {
              "description": "The description of the tab to be specified
here. It should be either an i18n key or as just a non-18n string. Even
though schema doesn't enforces it, title is required, which could be either
string or object, so that functionality can work properly",
              "type": "string"
            },
            {
              "description": "The description of the tab to be specified
here. It should be parametrized i18n key along with its params",
              "type": "object",
              "additionalProperties": false,
              "properties": {
                "text": {
                  "type": "string"
                },
                "params": {
                  "type": "array",
                  "items": {
                    "type": "string"
                  }
```

```
                }
              }
            }
          ]
        },
        "displayName": {
          "description": "Display name of the tab to b displayed along with
the Title",
          "type": "string"
        },
        "extendedTooltip": {
          "oneOf": [
            {
              "description": "The advanced title to display on the tooltip
of the tab. It will have the view as key where it will have the template
that can be shown in the tooltip box.",
              "type": "object",
              "properties": {
                "view": {
                  "type": "string"
                }
              }
            }
          ]
        },
        "indicators": {
          "description": "Icon indicators are like icons on the tab. These
are optional.",
          "type": "array",
          "items": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "iconId": {
                "description": "The id of the tab's icon. It refers to the
SVG icon that is stored in the 'src/image' folder in the site location",
                "type": "string"
              },
              "position": {
                "description": "Determines where the icon is to be
displayed. It's value can be either left or right",
                "type": "string"
              },
              "condition": {
                "$ref": "#/definitions/stringOrBoolDef",
                "description": "The name of the declarative condition which
will determine when this tab handler is active for the associated tab."
              }
            },
            "required": []
          }
        }
      },
      "required": []
    },
    "tabHandlersDef": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "description": "The tab Id for which the handler is associated.",
          "type": "string"
        },
        "view": {
          "description": "The view to be displayed when the tab is
selected.",
          "type": "string"
        },
        "activeWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which will
determine when this tab handler is active for the associated tab."
        },
        "visibleWhen": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
          "description": "The name of the declarative condition which
will determine when the associated tab isVisible. This condition is only
evaluated if this tab handler is active."
        },
        "actions": {
          "description": "The name of the declarative action
(REST,SOA,Event or JSFunction) to be executed.",
          "type": "object"
        },
        "subPanelContext": {
          "description": "Custom data specified through the handler.",
          "type": "object"
        }
      },
      "required": [
        "id",
        "activeWhen"
      ]
    },
    "tabPlacementsDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "description": "The tab Id for which the placement is
associated.",
          "type": "string"
        },
        "uiAnchor": {
          "description": "A well known name for a given element. Anchors
could be created where needed by Solutions.",
          "type": "string"
        },
        "priority": {
          "description": "The relative priority order to render tabs in
```

```
ascending order unless the tab bar has reverse alignment. Even though
schema doesn't enforces it, title is required so that functionality can
work properly.",
          "type": "number"
        }
      },
      "required": [
        "id"
      ]
    },
    "actionDef": {
      "description": "Actions can be of different actionType (REST,
JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have
inputData which specifies the JSON input for the action and outputData for
mapping the output response.",
      "oneOf": [
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Event"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
EventDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "RESTService"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
RESTServiceDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "Navigate"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
NavigateDef"
          },
```


Unpublished work. © 2025 Siemens

```
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "JSFunctionAsync"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
JSFunctionAsyncDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "JSFunction"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
JSFunctionDef"
          },
          "else": false
        },
        {
          "if": {
            "properties": {
              "actionType": {
                "const": "batchJob"
              }
            }
          },
          "then": {
            "$ref": "declarativeActionSchema-1.0.0.json#/definitions/
batchJobDef"
          },
          "else": false
        }
      ]
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
```

```
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses. The
conditions can refer to view model state and can update view model state.
They can leverage Boolean condition expressions like '&&', '||', '==', '!
='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    },
    "i18n": {
      "type": "object",
      "description": "All user facing text should be localized unless
a web client solution supports only one language. The View refers to
localizations defined in the 'i18n' section of the View Model. You can
specify the actual string to be displayed (however, it would not be
localized) OR the name of JSON bundle which holds the localized text to
be shown.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": [
            "array",
            "string"
          ],
          "items": {
            "type": "string",
            "description": "Name of the i18n bundle file to be specified in
array"
          }
        }
      }
    },
    "tabs": {
      "type": "object",
      "description": "Definition of a tab",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/tabDef"
        }
      }
    },
    "tabHandlers": {
      "type": "object",
      "description": "A tab can have many tab Handlers. However at any
given time, there may at most be only one active tabHandler for a tab.
A declarative 'activeWhen' condition which evaluates a boolean expression
controls whether a handler is currently active.  If more than one handler
for a given tab evaluates to true, then the more 'specific' condition (i.e.
the condition with longer expression) is chosen to be active.  By being
active, the tab handler will determine tab visibility and What a tab will
do when executed.",
      "patternProperties": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/tabHandlersDef"
        }
      }
    },
    "tabPlacements": {
      "type": "object",
      "description": "A tab can have many placements. A placement is the
actual visual rendering of the display of the tab.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/tabPlacementsDef"
        }
      }
    },
    "actions": {
      "type": "object",
      "description": "In the View Model, actions are declaratively
defined. Actions can be of different 'actionType' (REST, JSFunction,
JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which
specifies the JSON input for the action and 'outputData' for mapping the
output response. Actions can be used in UI elements such as an aw-button.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/actionDef"
        }
      }
    },
    "messages": {
      "type": "object",
      "description": "Messages are declaratively defined in the View Model.
Messages are launched by events and actions. They cover information,
warning, error notifications. They give the user more deeper customization
options. They are localized. They can fire some actions. They can even
leverage view model data and conditions.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
messageDef"
        }
      }
    },
    "onEvent": {
      "type": "array",
      "description": "The View Model can register to react to events
by declaring an onEvent section. It can then conditionally execute a
declarative action.",
      "required": [
        "eventId",
        "action"
      ],
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
```


Unpublished work. © 2025 Siemens

```
          "eventId": {
            "type": "string",
            "description": "The id of an event that the viewModel is
interested to listen to."
          },
          "criteria": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
                "type": [
                  "number",
                  "string",
                  "boolean"
                ]
              }
            },
            "properties": {
              "panelId": {
                "type": "string"
              },
              "destPanelId": {
                "type": "string"
              }
            }
          },
          "eventSource": {
            "type": "string",
            "description": "(Optional) set it to 'current' to make event
listeners only listen to events fired by the current view model"
          },
          "action": {
            "type": "string",
            "description": "An action that is defined in the actions
section of the viewModel which would be executed only if the specified
condition evaluates to true."
          },
          "message": {
            "type": "string"
          },
          "inputArgs": {
            "type": "object"
          },
          "cacheEventData": {
            "type": "boolean"
          },
          "condition": {
            "type": "string"
          },
          "eventData": {
            "type": "object"
          }
        }
      }
    }
  },
```



Unpublished work. © 2025 Siemens

```
  "__defs": []
}
```

## actionDef Definition (TabsViewModel)

Actions can be of different actionType (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have inputData which specifies the JSON input for the action and outputData for mapping the output response.

This definition must satisfy *one* of the following constraints:

- if **actionType** is set to `Event` **EventDef**
- if **actionType** is set to `RESTService` **RESTServiceDef**
- if **actionType** is set to `Navigate` **NavigateDef**
- if **actionType** is set to `JSFunctionAsync` **JSFunctionAsyncDef**
- if **actionType** is set to `JSFunction` **JSFunctionDef**
- if **actionType** is set to `batchJob` **batchJobDef**

## stringOrBoolDef Definition (TabsViewModel)

This definition must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a boolean value

## tabDef Definition (TabsViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `iconId` | string | | The id of the tab's icon. It refers to the SVG icon that is stored in the 'src/image' folder in the site location |
| `title` | **TitleAltObject** | | |
| `description` | **DescriptionAltObject** | | |

| | | | |
|---|---|---|---|
| `displayName` | string | | Display name of the tab to b displayed along with the Title |
| `extendedTooltip` | ExtendedTooltipAltObject | | |
| `indicators` | ArrayIndicatorsObject | | Icon indicators are like icons on the tab. These are optional. |

### Referenced Objects

The following objects are referenced within this definition.

#### DescriptionAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a DescriptionObject1 value

#### DescriptionObject1

The description of the tab to be specified here. It should be parametrized i18n key along with its params

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | |
| `params` | Arraystring | | |

#### ExtendedTooltipAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a ExtendedTooltipObject1 value


Siemens Web Framework
Unpublished work. © 2025 Siemens

### ExtendedTooltipObject1

The advanced title to display on the tooltip of the tab. It will have the view as key where it will have the template that can be shown in the tooltip box.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `view` | string | | |

### IndicatorsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `iconId` | string | | The id of the tab's icon. It refers to the SVG icon that is stored in the 'src/image' folder in the site location |
| `position` | string | | Determines where the icon is to be displayed. It's value can be either left or right |
| `condition` | **stringOrBoolDef** | | The name of the declarative condition which will determine when this tab handler is active for the associated tab. |

### TitleAltObject

*No description specified.*

This value must satisfy *one* of the following constraints:

- it must be set to a string value
- it must be set to a **TitleObject1** value

### TitleObject1

The title to display on the tooltip of the tab. It should be parametrized i18n key along with its params

This object can contain the following properties:


Unpublished work. © 2025 Siemens

| Name | Type | Required | Description |
|---|---|---|---|
| `text` | string | | |
| `params` | Arraystring | | |

## tabHandlersDef Definition (TabsViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | The tab Id for which the handler is associated. |
| `view` | string | | The view to be displayed when the tab is selected. |
| `activeWhen` | conditionOrBoolDef | Yes | The name of the declarative condition which will determine when this tab handler is active for the associated tab. |
| `visibleWhen` | conditionOrBoolDef | | The name of the declarative condition which will determine when the associated tab isVisible. This condition is only evaluated if this tab handler is active. |
| `actions` | ActionsObject | | The name of the declarative action (REST,SOA,Event or JSFunction) to be executed. |
| `subPanelContext` | SubPanelContextObject | | Custom data specified through the handler. |

### Referenced Objects

The following objects are referenced within this definition.

#### ActionsObject

The name of the declarative action (REST,SOA,Event or JSFunction) to be executed.

#### SubPanelContextObject

Custom data specified through the handler.


Unpublished work. © 2025 Siemens

### tabPlacementsDef Definition (TabsViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | The tab Id for which the placement is associated. |
| `uiAnchor` | string | | A well known name for a given element. Anchors could be created where needed by Solutions. |
| `priority` | number | | The relative priority order to render tabs in ascending order unless the tab bar has reverse alignment. Even though schema doesn't enforces it, title is required so that functionality can work properly. |

## TypeIconsRegistry Schema

This schema provides a way to configure which type icon to display depending on specific conditions typically based on properties present in View Model Objects.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `$schema` | string | Yes | The schema against which the JSON file is validated. It should be the same as the file name. For example - 'declarativeIndicatorsSchema-1.0.0.json' is valid, 'declarativeIndicatorsSchema-1.0.0' is not. |
| `typeIconsRegistry` | TypeIconsRegistryObject | Yes | The collection of typeIconsRegistry objects. |

### Referenced Objects

The following objects are referenced within this schema.

### TypeIconsRegistryObject

The collection of typeIconsRegistry objects.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | TypeIconsRegistryObject | | Name of the unique typeIconRegistry Object. |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeTypeIconsRegistrySchema-1.0.0.json",
  "description": "This schema provides a way to configure which type icon
to display depending on specific conditions typically based on properties
present in View Model Objects.",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "$schema": {
      "type": "string",
      "description": "The schema against which the JSON file
is validated. It should be the same as the file name.
For example - 'declarativeIndicatorsSchema-1.0.0.json' is valid,
'declarativeIndicatorsSchema-1.0.0' is not."
    },
    "typeIconsRegistry": {
      "type": "object",
      "description": "The collection of typeIconsRegistry objects.",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "description": "Name of the unique typeIconRegistry Object.",
          "$ref": "declarativeTypeIconsRegistryObjectSchema-1.0.0.json"
        }
      }
    }
  },
  "required": [
    "$schema",
    "typeIconsRegistry"
  ],
  "__defs": []
}
```

## TypeIconsRegistryObject Schema

This schema points to the typeIconsRegistryObjDef definition.


Siemens Web Framework
Unpublished work. © 2025 Siemens

## Definitions

This schema includes the following definitions:

- nameDef
- typeDef
- typeIconsRegistryObjDef

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeTypeIconsRegistryObjectSchema-1.0.0.json",
  "description": "This schema provides a way to configure which type icon
to display depending on specific conditions typically based on properties
present in View Model Objects.This is the schema for declarative indicators
(in JSON format). The SWF audit check validates all indicators JSON files
against this schema. For explanation of the keywords of this schema, please
visit https://ajv.js.org/keywords.html .",
  "definitions": {
    "nameDef": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "typeDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "names": {
          "description": "List of ViewModelObject types on which the
indicators need to be evaluated.",
          "$ref": "#/definitions/nameDef"
        },
        "prop": {
          "description": "An object containing a list of properties for the
type this Icon is being evaluated on",
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "names": {
              "description": "The name of the VMP of the object specified
in type section",
              "$ref": "#/definitions/nameDef"
            },
            "condition": {
              "description": "The condition under which the typeIcon needs
to be applied. It is a standard declarative condition which can be either
string based or object based.",
              "$ref": "declarativeConditionSchema-1.0.0.json#/definitions/
expressionStringOrObjectDef"
```

```
            },
            "type": {
              "description": "The type name of the ViewModelObject
referenced by the ViewModelProperty. This is optional - needed if a
referenced type needs to be traversed.",
              "$ref": "#/definitions/typeDef"
            }
          },
          "oneOf": [
            {
              "required": [
                "names",
                "type"
              ]
            },
            {
              "required": [
                "names",
                "condition"
              ]
            }
          ]
        },
        "condition": {
          "description": "Condition on which Icon to be made visible",
          "$ref": "declarativeConditionSchema-1.0.0.json#/definitions/
expressionStringOrObjectDef"
        }
      },
      "required": [
        "names"
      ]
    },
    "typeIconsRegistryObjDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "iconId": {
          "description": "The name of the icon/typeIcon to display, with or
without size suffix and without extension. For example - 'typeSMWDiagram48'
is valid, 'typeSMWDiagram48.svg' is not.",
          "type": "string"
        },
        "type": {
          "description": "The type name of the View Model Object referenced
by the View Model Property",
          "$ref": "#/definitions/typeDef"
        },
        "thumbnail": {
          "description": "The name of the thumbnail of the View Model
Object referenced by the View Model Property",
          "$ref": "#/definitions/typeDef"
        },
        "priority": {
          "description": "The priority of the typeIconsRegistry object.
Value should be a number. If not provided, the priority is considered 1.",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "type": "integer"
        },
        "isClientViewModelObject": {
          "description": "Boolean value indicating if the context VMO is
not a represetation of underlying database Model. Defaults to false.",
          "type": "boolean"
        }
      },
      "oneOf": [
        {
          "required": [
            "type"
          ]
        },
        {
          "required": [
            "thumbnail"
          ]
        }
      ]
    }
  },
  "$ref": "#/definitions/typeIconsRegistryObjDef",
  "__defs": []
}
```

## nameDef Definition (TypeIconsRegistryObject)

This value is an array of **string** objects.

## typeDef Definition (TypeIconsRegistryObject)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| names | nameDef | Yes | List of ViewModelObject types on which the indicators need to be evaluated. |
| prop | PropAltObject | | An object containing a list of properties for the type this Icon is being evaluated on |
| condition | expressionStringOrObjectDef | | Condition on which Icon to be made visible |

#### Referenced Objects

The following objects are referenced within this definition.

#### PropAltObject

An object containing a list of properties for the type this Icon is being evaluated on

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `names` | nameDef | | The name of the VMP of the object specified in type section |
| `condition` | expressionStringOrObjectDef | | The condition under which the typeIcon needs to be applied. It is a standard declarative condition which can be either string based or object based. |
| `type` | typeDef | | The type name of the ViewModelObject referenced by the ViewModelProperty. This is optional - needed if a referenced type needs to be traversed. |

This value must satisfy *one* of the following constraints:

- the following properties must be specified: **names**, **type**
- the following properties must be specified: **names**, **condition**

### typeIconsRegistryObjDef Definition (TypeIconsRegistryObject)

This definition must satisfy *one* of the following constraints:

- the following properties must be specified: **type**
- the following properties must be specified: **thumbnail**

## ViewModel Schema

This is the schema the declarative View Model (in JSON format) must adhere to. The Apollo audit check validates all View Model JSON files against this schema. For explanation of the keywords of this schema, please visit https://ajv.js.org/keywords.html


Unpublished work. © 2025 Siemens

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with |
| `imports` | Arraystring | | **This property is deprecated** (This is no longer used in @**swf/core@6.0.0**.) Imports are used in the View Model to indicate the UI elements we want to use in our View. The Apollo declarative runtime demand loads the needed elements. It must contain only unique items. |
| `actions` | **ActionsObject** | | In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button. |
| `data` | **DataAltObject** | | The View can refer to data and view state, defined in the View Model data section. The braces '{{ }}' allow declarative binding to the live data. |
| `preferences` | Arraystring | | undefined. It must contain only unique items. |
| `functions` | **functionsDef** | | |
| `messages` | **MessagesObject** | | Messages are declaratively defined in the View Model. Messages are launched by events and actions. |

| | | | | |
|---|---|---|---|---|
| | | | They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions. | |
| `conditions` | **ConditionsObject** | | Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', ' | ', '==', '!=' |
| `i18n` | **i18nDef** | | | |
| `editContext` | string | | Specifying editContext will create an edit handler with the given editContext for this View Model. | |
| `onEvent` | Array**OnEventObject** | | The View Model can register to react to events by declaring an onEvent section. It can then conditionally execute a declarative action. | |
| `dataParseDefinitions` | **DataParseDefinitionsObject** | | It contains the parser definitions. Each parser definition must define a 'props'. Each prop, which is of a specific JSON structure, is converted to a ViewModelProperty. | |
| `dataProviders` | **DataProvidersObject** | | Data Provider, as the name suggests, is a single placeholder wherein we configure different ways | |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | to fetch/paginate data and pass the same data to underlying component. It is basically an abstraction layer for components which fulfils component's demand to load/paginate data from the server. In Apollo, data-provider is used with 'aw-list' & 'aw-table (including tree)' to meet their data requirements. |
| `columnProviders` | **ColumnProvidersObject** | | |
| `chartProviders` | **ChartProvidersObject** | | Chart Provider, as the name suggests, is a single placeholder wherein we configure different ways to fetch data and pass the same data to underlying chart component. It is basically an abstraction layer for components which fulfils component's demand to load chart data from the server. In Apollo, chart-provider is used with 'aw-chart' to meet their data requirements. |
| `grids` | **GridsObject** | | |
| `editConfig` | **EditConfigObject** | | |
| `mock` | **MockObject** | | The Mock section of the View Model |
| `ports` | **PortsObject** | | |
| `lifecycleHooks` | **LifecycleHooksObject** | | |
| `props` | **PropsObject** | | |
| `ctx` | **CtxObject** | | |
| `ctxFields` | array | | |
| `selectionModels` | **SelectionModelsObject** | | |

| | | | |
|---|---|---|---|
| editHandlers | EditHandlersObject | | |
| doc | DocObject | | Metadata used to document a component. |

## Definitions

This schema includes the following definitions:

- chartPointsConfigDef
- chartProviderDef
- commandDef
- dataParseDef
- dataProvidersDef
- editConfigDef
- gridDef
- i18nDef
- portsDef
- workspaceDef

## Referenced Objects

The following objects are referenced within this schema.

## ActionsObject

In the View Model, actions are declaratively defined. Actions can be of different 'actionType' (REST, JSFunction, JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which specifies the JSON input for the action and 'outputData' for mapping the output response. Actions can be used in dataProviders or UI elements such as an aw-button.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | Action | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## ChartProvidersObject

Chart Provider, as the name suggests, is a single placeholder wherein we configure different ways to fetch data and pass the same data to underlying chart component. It is basically an abstraction layer for components which fulfils component's demand to load chart data from the server. In Apollo, chart-provider is used with 'aw-chart' to meet their data requirements.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | chartProviderDef | | |

## ColumnProvidersObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | dataProvidersDef | | |

## ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | Condition | | |

## CriteriaObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `panelId` | string | | |
| `destPanelId` | string | | |

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b` | number OR string OR boolean | | |

### CtxObject

*No description specified.*

### DataAltObject

The View can refer to data and view state, defined in the View Model data section. The braces '{{ }}' allow declarative binding to the live data.

This value must satisfy *any* of the following constraints:

- it must be set to a DataObject1 value
- it must be set to a DataObject2 value

### DataObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `objects` | viewModelObjectDef | | |

### DataObject2

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:


Siemens Web Framework
Unpublished work. © 2025 Siemens

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | viewModelPropDef | | |

## DataParseDefinitionsObject

It contains the parser definitions. Each parser definition must define a 'props'. Each prop, which is of a specific JSON structure, is converted to a ViewModelProperty.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | dataParseDef | | |

## DataProvidersObject

Data Provider, as the name suggests, is a single placeholder wherein we configure different ways to fetch/paginate data and pass the same data to underlying component. It is basically an abstraction layer for components which fulfils component's demand to load/paginate data from the server. In Apollo, data-provider is used with 'aw-list' & 'aw-table (including tree)' to meet their data requirements.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| ^[a-zA-Z_]+[a-zA-Z0-9_]*$ | dataProvidersDef | | |

## DocObject

Metadata used to document a component.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| access | string | | It can be set to one of the following values: private, public. Specifies whether the component is public or private. |
| description | string | | A brief description of the component and its purpose. |

| | | | |
|---|---|---|---|
| `deprecated` | string | | It must match the pattern `/\d+\.\d+\.\d+ - .+/`. If the component is deprecated, set this field to a string in the following format: " - ". |
| `since` | string | | It must match the pattern `/\d+\.\d+\.\d+/`. The version in which the component was introduced. |
| `moreInfo` | string | | A relative path to a markdown file containing additional information and examples related to the component. |
| `controls` | string | | A relative path to the controls file which defines the controls corresponding to the component. |

### EditConfigObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | editConfigDef | | |

### EditHandlersObject

*No description specified.*

### EventDataObject

*No description specified.*

### GridsObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b` | gridDef | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## InputArgsObject

*No description specified.*

## LifecycleHooksObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `onInit` | string | | Name of the action to be called when the ViewModel is constructed |
| `onMount` | string | | Name of the action to be called when the ViewModel is mounted in DOM |
| `onUnmount` | string | | Name of the action to be called when the ViewModel is unmounted from DOM |
| `render` | string | | Name of the action to be called when ViewModel is rendered |
| `onUpdate` | LifecycleHooksPatternObject | | Name of the action to be called when ViewModel is updated |

## LifecycleHooksPatternObject

Name of the action to be called when ViewModel is updated

## MessagesObject

Messages are declaratively defined in the View Model. Messages are launched by events and actions. They cover information, warning, error notifications. They give the user more deeper customization options. They are localized. They can fire some actions. They can even leverage view model data and conditions.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | messageDef | | |

## MockObject

The Mock section of the View Model

## OnEventObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `eventId` | string | | The id of an event that the viewModel is interested to listen to. |
| `criteria` | CriteriaObject | | |
| `eventSource` | string | | (Optional) set it to 'current' to make event listeners only listen to events fired by the current view model instance |
| `action` | string | | An action that is defined in the actions section of the viewModel which would be executed only if the specified condition evaluates to true. |
| `message` | string | | |
| `inputArgs` | InputArgsObject | | |
| `cacheEventData` | boolean | | |
| `condition` | string | | |
| `eventData` | EventDataObject | | |

## PortsObject

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | portsDef | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

## PropsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| doc | DocObject | | Metadata used to document a prop of a component. |

## SelectionModelsObject

*No description specified.*

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeViewModelSchema-1.0.0.json",
  "description": "This is the schema the declarative View Model (in JSON
format) must adhere to. The Apollo audit check validates all View Model
JSON files against this schema. For explanation of the keywords of this
schema, please visit https://ajv.js.org/keywords.html",
  "definitions": {
    "chartPointsConfigDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "seriesType": {
          "type": "string",
          "description": "The type of series, for example line or column or
pie or scatter or spline, etc"
        },
        "seriesMarker": {
          "type": "object",
          "description": "Options for the point markers of series.
Properties like radius, symbol, lineColor and lineWidth define the visual
appearance of the markers."
        },
        "seriesTooltip": {
          "type": "object",
          "description": "Options for the tooltip of series. Properties
like headerFormat, pointFormat define the visual appearance of the tooltip
for markers."
        },
        "yAxis": {
          "type": "number",
          "description": "When using dual or multiple y axes, this number
defines which yAxis the particular series is connected to."
        },
```

```
        "dataLabels": {
          "type": "object",
          "description": "Options for the series data labels, appearing
next to each data point."
        },
        "showInLegend": {
          "type": "boolean",
          "description": "Show series name in the legend."
        },
        "isDataLabelOnChartEnabled": {
          "type": "boolean",
          "description": "show labels on chart."
        },
        "seriesColorClass": {
          "type": "string",
          "description": "Specify class which has custom colors."
        },
        "dashStyle": {
          "type": "string",
          "description": "Represents the line style of the series for line
charts."
        }
      }
    },
    "dataProvidersDef": {
      "anyOf": [
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "action": {
              "type": "string"
            },
            "initializeAction": {
              "type": "string"
            },
            "nextAction": {
              "type": "string"
            },
            "previousAction": {
              "type": "string"
            },
            "focusAction": {
              "type": "string"
            },
            "expandAction": {
              "type": "string"
            },
            "commands": {
              "type": "object",
              "items": {
                "$ref": "#/definitions/commandDef"
              }
            },
            "enablePropEdit": {
              "type": [
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
                "string",
                "boolean"
              ],
              "description": "Enables the ability to edit table cells
without having to go through the Edit Command. Click twice on any editable
table cell to edit the cell. The possible values of this option are true,
false or a declarative condition. False, will disable this behavior. True
will enable this behavior (default). A declarative condtion will allow True/
False to be set based on the conditions resolved value. If the \"Autosave
Tables\" toggle is enabled in the user settings panel, changes to the table
cells will be saved instantly on blur. If the toggle is disabled, the user
has to explicitly save the edits using the save command."
            },
            "editContext": {
              "type": "string",
              "description": "Specifying editContext will create an edit
handler with the given editContext for this Data Provider."
            },
            "response": {
              "type": "string"
            },
            "totalFound": {
              "type": "string"
            },
            "totalLoaded": {
              "type": "string"
            },
            "uidInResponse": {
              "type": "string"
            },
            "preSelection": {
              "type": "boolean"
            },
            "selectionModelMode": {
              "type": "string",
              "enum": [
                "multiple",
                "single"
              ]
            }
          },
          "required": [
            "action"
          ]
        },
        {
          "type": "object",
          "items": {
            "type": "array",
            "additionalProperties": false,
            "properties": {
              "name": {
                "type": "string"
              },
              "displayName": {
                "type": "string"
```

```
              },
              "minWidth": {
                "type": "integer"
              },
              "maxWidth": {
                "type": "integer"
              },
              "width": {
                "type": "integer"
              },
              "enableColumnMoving": {
                "type": "boolean"
              },
              "enableColumnMenu": {
                "type": "boolean"
              },
              "enableColumnResizing": {
                "type": "boolean"
              },
              "enableFiltering": {
                "type": "boolean"
              },
              "isFilteringEnabled": {
                "type": "boolean"
              },
              "enableHiding": {
                "type": "boolean"
              },
              "enablePinning": {
                "type": "boolean"
              },
              "enableSorting": {
                "type": "boolean"
              },
              "headerTooltip": {
                "type": "boolean"
              },
              "pinnedLeft": {
                "type": "boolean"
              }
            }
          }
        }
      ]
    },
    "commandDef": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "type": "string"
        },
        "title": {
          "type": "string"
        },
        "areas": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "type": "array"
        },
        "priorities": {
          "type": "array"
        },
        "action": {
          "type": "string"
        },
        "dependencies": {
          "type": "array"
        },
        "displayOption": {
          "type": "string",
          "enum": [
            "ON_HOVER",
            "ON_SELECTION",
            "ON_HOVER_AND_SELECTION",
            "ALWAYS"
          ]
        },
        "position": {
          "type": "string",
          "enum": [
            "TOP_RIGHT",
            "MIDDLE_RIGHT",
            "BOTTOM_RIGHT",
            "TOP_LEFT",
            "MIDDLE_LEFT",
            "BOTTOM_LEFT"
          ]
        }
      }
    },
    "gridDef": {
      "additionalProperties": false,
      "type": "object",
      "required": [
        "dataProvider"
      ],
      "properties": {
        "addIconColumn": {
          "type": "boolean"
        },
        "columnProvider": {
          "type": "string"
        },
        "dataProvider": {
          "type": "string"
        },
        "deps": {
          "type": "string"
        },
        "enableArrangeMenu": {
          "type": "boolean"
        },
        "gridOptions": {
```

```
            "type": "object",
            "properties": {
              "enableGridMenu": {
                "type": "boolean"
              },
              "enableFiltering": {
                "type": "boolean"
              },
              "isFilteringEnabled": {
                "type": [
                  "boolean",
                  "string"
                ]
              },
              "enableHeaderIcon": {
                "type": "boolean"
              },
              "enablePinning": {
                "type": [
                  "boolean",
                  "string"
                ]
              },
              "enableSorting": {
                "type": "boolean"
              },
              "headerHeight": {
                "type": [
                  "string",
                  "number"
                ]
              },
              "rowHeight": {
                "type": [
                  "number",
                  "string"
                ]
              },
              "textBundle": {
                "type": "object",
                "properties": {
                  "TwistieTooltipCollapsed": {
                    "type": "string"
                  },
                  "TwistieTooltipExpanded": {
                    "type": "string"
                  }
                }
              },
              "transpose": {
                "type": "boolean"
              },
              "showContextMenu": {
                "type": "boolean"
              }
            }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
        },
        "pinIconColumn": {
          "type": "boolean"
        },
        "propertyProvider": {
          "type": "string"
        }
      }
    },
    "chartProviderDef": {
      "additionalProperties": false,
      "type": "object",
      "required": [
        "chartType"
      ],
      "properties": {
        "title": {
          "type": "string"
        },
        "chartColorOverrideClass": {
          "type": "string"
        },
        "chartType": {
          "type": "string",
          "enum": [
            "column",
            "line",
            "pie",
            "heatmap",
            "scatter",
            "spline"
          ]
        },
        "loadDataAction": {
          "type": "string"
        },
        "selectAction": {
          "type": "string"
        },
        "unSelectAction": {
          "type": "string"
        },
        "chartConfig": {
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "isChartZoomable": {
              "type": "boolean"
            },
            "isYAxisLinearOrLogarithmic": {
              "type": "string",
              "enum": [
                "linear",
                "logarithmic"
              ]
            },
```

```
            "size": {
              "type": "string"
            },
            "xAxisLabel": {
              "type": "string"
            },
            "yAxisLabel": {
              "type": "string"
            },
            "isDataLabelOnChartEnabled": {
              "type": "boolean"
            },
            "spacingLeft": {
              "type": "number",
              "description": "Number of pixels to offset chart to the left.
Used by highchart and ignored by echarts."
            },
            "showAxes": {
              "type": "boolean",
              "description": "Hides axes when no data for highchart,
ignored by echarts."
            },
            "tooltip": {
              "type": "object",
              "description": "tooltip data is a JSON object. It uses the
same format as the highchart/echarts chart config tooltip option."
            },
            "xAxis": {
              "type": "object",
              "description": "xAxis data is a JSON object. It uses the same
format as the highchart/echarts chart config xAxis option."
            },
            "yAxis": {
              "type": "object",
              "description": "yAxis data is a JSON object. It uses the same
format as the highchart/echarts config yAxis option."
            },
            "plotOptions": {
              "type": "object",
              "description": "plotOptions data is a JSON object. It is the
same as highchart config plotOptions object. Not used for echart."
            },
            "legend": {
              "type": "object",
              "description": "legend data is a JSON object. It uses the
same format as the highchart/echarts chart config legend option."
            },
            "dataLabels": {
              "type": "object",
              "description": "Options for the series data labels, appearing
next to each data point."
            }
          }
        },
        "chartPointsConfig": {
          "type": "array",
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
          "items": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
                "$ref": "#/definitions/chartPointsConfigDef"
              }
            }
          },
          "minItems": 0,
          "uniqueItems": true
        }
      }
    },
    "workspaceDef": {
      "additionalProperties": false,
      "type": "object",
      "required": [
        "workspaceId",
        "workspaceName",
        "workspaceType",
        "defaultPage"
      ],
      "properties": {
        "schemaVersion": {
          "description": "Version of the schema it complies with",
          "type": [
            "string",
            "integer"
          ]
        },
        "workspaceId": {
          "type": "string"
        },
        "workspaceName": {
          "type": [
            "string",
            "object"
          ],
          "properties": {
            "source": {
              "type": "string"
            },
            "key": {
              "type": "string"
            }
          }
        },
        "workspaceType": {
          "type": "string"
        },
        "scope": {
          "type": "string"
        },
        "defaultPage": {
          "type": "string"
```

```
        },
        "availablePages": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "uniqueItems": true
        }
      }
    },
    "editConfigDef": {
      "additionalProperties": false,
      "type": "string",
      "required": [
        "startEditAction",
        "saveEditAction",
        "identifier"
      ],
      "properties": {
        "startEditAction": {
          "type": "string"
        },
        "saveEditAction": {
          "type": "string"
        },
        "identifier": {
          "type": "string"
        }
      }
    },
    "portsDef": {
      "additionalProperties": false,
      "type": "object",
      "description": "Observables defined in the viewModels.",
      "properties": {
        "onChangeAction": {
          "type": "string",
          "description": "Name of the action which will be auto-invoked
when the data associated with the port is changed"
        },
        "syncObject": {
          "type": "object",
          "description": "Data passed from source to target"
        }
      },
      "required": [
        "syncObject"
      ]
    },
    "dataParseDef": {
      "oneOf": [
        {
          "type": "object",
          "additionalProperties": false,
          "properties": {
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            "path": {
              "type": "string"
            },
            "dataInput": {
              "type": "string"
            },
            "outputFormatType": {
              "type": "string"
            },
            "operationType": {
              "type": "string"
            },
            "dataFilter": {
              "type": "array",
              "items": {
                "type": "object",
                "additionalProperties": false,
                "properties": {
                  "condition": {
                    "type": "string"
                  }
                },
                "required": [
                  "condition"
                ]
              }
            },
            "viewModelPropValues": {
              "type": "object",
              "properties": {
                "type": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "additionalProperties": false,
                    "properties": {
                      "condition": {
                        "type": "string"
                      },
                      "value": {
                        "type": "string"
                      }
                    },
                    "required": [
                      "condition",
                      "value"
                    ]
                  }
                },
                "displayName": {
                  "type": "string"
                },
                "dbValue": {
                  "type": "array",
                  "items": {
                    "type": "object",
```

```
                    "additionalProperties": false,
                    "properties": {
                      "condition": {
                        "type": "string"
                      },
                      "value": {
                        "type": "string"
                      }
                    },
                    "required": [
                      "condition",
                      "value"
                    ]
                  }
                },
                "propMisc": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "additionalProperties": false,
                    "properties": {
                      "propName": {
                        "type": "string"
                      },
                      "value": {
                        "type": "string"
                      }
                    },
                    "required": [
                      "propName",
                      "value"
                    ]
                  }
                }
              },
              "required": [
                "type",
                "displayName",
                "dbValue",
                "propMisc"
              ]
            },
            "outputFormat": {
              "type": "array",
              "items": {
                "anyOf": [
                  {
                    "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/fieldDef"
                  },
                  {
                    "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/structDef"
                  }
                ]
              }
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
            },
            "filterOtherData": {
              "type": "object",
              "properties": {
                "path": {
                  "type": "string"
                },
                "dataFilter": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "additionalProperties": false,
                    "properties": {
                      "condition": {
                        "type": "string"
                      }
                    },
                    "required": [
                      "condition"
                    ]
                  }
                }
              },
              "required": [
                "path",
                "dataFilter"
              ]
            }
          }
        },
        {
          "type": "object",
          "additionalProperties": true,
          "properties": {
            "props": {
              "oneOf": [
                {
                  "type": "object",
                  "additionalProperties": false,
                  "patternProperties": {
                    "^[a-zA-Z0-9_]*$": {
                      "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/dataMapperEachPropDef"
                    }
                  }
                },
                {
                  "type": "string"
                }
              ]
            }
          },
          "required": [
            "props"
          ]
        }
```

```
      ]
    },
    "i18nDef": {
      "type": "object",
      "description": "All user facing text should be localized unless
a web client solution supports only one language. The View refers to
localizations defined in the 'i18n' section of the View Model. You can
specify the actual string to be displayed (however, it would not be
localized) OR the name of JSON bundle which holds the localized text to
be shown.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": [
            "array",
            "string"
          ],
          "items": {
            "type": "string",
            "description": "Name of the i18n bundle file to be specified in
array"
          }
        }
      }
    }
  },
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "description": "Version of the schema it complies with",
      "type": [
        "string",
        "integer"
      ]
    },
    "imports": {
      "type": "array",
      "description": "Imports are used in the View Model to indicate the UI
elements we want to use in our View. The Apollo declarative runtime demand
loads the needed elements.",
      "deprecated": "This is no longer used in @swf/core@6.0.0.",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "actions": {
      "type": "object",
      "description": "In the View Model, actions are declaratively
defined. Actions can be of different 'actionType' (REST, JSFunction,
JSFunctionAsync, Event or TcSoaService). All actions have 'inputData' which
specifies the JSON input for the action and 'outputData' for mapping the
output response. Actions can be used in dataProviders or UI elements such
as an aw-button.",
      "additionalProperties": false,
```


Unpublished work. © 2025 Siemens

```
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeActionSchema-1.0.0.json"
        }
      }
    },
    "data": {
      "type": "object",
      "description": "The View can refer to data and view state, defined in
the View Model data section. The braces '{{  }}' allow declarative binding
to the live data.",
      "anyOf": [
        {
          "properties": {
            "objects": {
              "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/viewModelObjectDef"
            }
          }
        },
        {
          "type": "object",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/
definitions/viewModelPropDef"
            }
          }
        }
      ]
    },
    "preferences": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "functions": {
      "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
functionsDef"
    },
    "messages": {
      "type": "object",
      "description": "Messages are declaratively defined in the View Model.
Messages are launched by events and actions. They cover information,
warning, error notifications. They give the user more deeper customization
options. They are localized. They can fire some actions. They can even
leverage view model data and conditions.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
messageDef"
        }
      }
```

```
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and
can be used for event handling (onEvent handler). The data value can
be live evaluated by condition. The conditions can refer to view model
state and can update view model state. They can leverage Boolean condition
expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    },
    "i18n": {
      "$ref": "#/definitions/i18nDef"
    },
    "editContext": {
      "type": "string",
      "description": "Specifying editContext will create an edit handler
with the given editContext for this View Model."
    },
    "onEvent": {
      "type": "array",
      "description": "The View Model can register to react to events
by declaring an onEvent section. It can then conditionally execute a
declarative action.",
      "required": [
        "eventId",
        "action"
      ],
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "eventId": {
            "type": "string",
            "description": "The id of an event that the viewModel is
interested to listen to."
          },
          "criteria": {
            "type": "object",
            "additionalProperties": false,
            "patternProperties": {
              "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
                "type": [
                  "number",
                  "string",
                  "boolean"
                ]
              }
            },
            "properties": {
              "panelId": {
                "type": "string"
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
              },
              "destPanelId": {
                "type": "string"
              }
            }
          },
          "eventSource": {
            "type": "string",
            "description": "(Optional) set it to 'current' to make event
listeners only listen to events fired by the current view model instance"
          },
          "action": {
            "type": "string",
            "description": "An action that is defined in the actions
section of the viewModel which would be executed only if the specified
condition evaluates to true."
          },
          "message": {
            "type": "string"
          },
          "inputArgs": {
            "type": "object"
          },
          "cacheEventData": {
            "type": "boolean"
          },
          "condition": {
            "type": "string"
          },
          "eventData": {
            "type": "object"
          }
        }
      }
    },
    "dataParseDefinitions": {
      "type": "object",
      "description": "It contains the parser definitions. Each parser
definition must define a 'props'. Each prop, which is of a specific JSON
structure, is converted to a ViewModelProperty.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/dataParseDef"
        }
      }
    },
    "dataProviders": {
      "type": "object",
      "description": "Data Provider, as the name suggests, is a single
placeholder wherein we configure different ways to fetch/paginate data and
pass the same data to underlying component. It is basically an abstraction
layer for components which fulfils component's demand to load/paginate data
from the server. In Apollo, data-provider is used with 'aw-list' & 'aw-
table (including tree)' to meet their data requirements.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
```

```
          "type": "object",
          "$ref": "#/definitions/dataProvidersDef"
        }
      }
    },
    "columnProviders": {
      "type": "object",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": "object",
          "$ref": "#/definitions/dataProvidersDef"
        }
      }
    },
    "chartProviders": {
      "type": "object",
      "description": "Chart Provider, as the name suggests, is a single
placeholder wherein we configure different ways to fetch data and pass the
same data to underlying chart component. It is basically an abstraction
layer for components which fulfils component's demand to load chart data
from the server. In Apollo, chart-provider is used with 'aw-chart' to meet
their data requirements.",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": "object",
          "$ref": "#/definitions/chartProviderDef"
        }
      }
    },
    "grids": {
      "type": "object",
      "patternProperties": {
        "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
          "type": "object",
          "$ref": "#/definitions/gridDef"
        }
      }
    },
    "editConfig": {
      "type": "object",
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "type": "string",
          "$ref": "#/definitions/editConfigDef"
        }
      }
    },
    "mock": {
      "type": "object",
      "description": "The Mock section of the View Model",
      "items": {
        "dataProviders": {
          "type": "object",
          "description": "Data Provider, as the name suggests, is a single
placeholder wherein we configure different ways to fetch/paginate data and
pass the same data to underlying component. It is basically an abstraction
```


Unpublished work. © 2025 Siemens

```
layer for components which fulfils component's demand to load/paginate data
from the server. In Apollo, data-provider is used with 'aw-list' & 'aw-
table (including tree)' to meet their data requirements.",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "type": "object",
              "$ref": "#/definitions/dataProvidersDef"
            }
          }
        },
        "columnProviders": {
          "type": "object",
          "patternProperties": {
            "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
              "type": "object",
              "$ref": "#/definitions/dataProvidersDef"
            }
          }
        }
      }
    },
    "ports": {
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
        "^[a-zA-Z_]+[a-zA-Z0-9_]*$": {
          "$ref": "#/definitions/portsDef"
        }
      }
    },
    "lifecycleHooks": {
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
        "onInit": {
          "type": "string",
          "description": "Name of the action to be called when the
ViewModel is constructed"
        },
        "onMount": {
          "type": "string",
          "description": "Name of the action to be called when the
ViewModel is mounted in DOM"
        },
        "onUnmount": {
          "type": "string",
          "description": "Name of the action to be called when the
ViewModel is unmounted from DOM"
        },
        "render": {
          "type": "string",
          "description": "Name of the action to be called when ViewModel is
rendered"
        },
        "onUpdate": {
          "type": [
```

```
            "object",
            "array"
          ],
          "description": "Name of the action to be called when ViewModel is
updated"
        }
      }
    },
    "props": {
      "type": "object",
      "properties": {
        "doc": {
          "type": "object",
          "description": "Metadata used to document a prop of a component.",
          "properties": {
            "access": {
              "type": "string",
              "enum": [
                "private",
                "public"
              ],
              "description": "Specifies whether the prop is public or
private."
            },
            "description": {
              "type": "string",
              "description": "A brief description of the prop and its
purpose."
            },
            "deprecated": {
              "type": "string",
              "pattern": "\\d+\\.\\d+\\.\\d+ - .+",
              "description": "If the prop is deprecated, set this field
to a string in the following format: \"<version number> - <text listing
alternatives>\"."
            },
            "since": {
              "type": "string",
              "pattern": "\\d+\\.\\d+\\.\\d+",
              "description": "The version in which the prop was introduced."
            }
          },
          "additionalProperties": false
        }
      }
    },
    "ctx": {
      "type": "object"
    },
    "ctxFields": {
      "type": "array"
    },
    "selectionModels": {
      "type": "object"
    },
    "editHandlers": {
```


Unpublished work. © 2025 Siemens

```
      "type": "object"
    },
    "doc": {
      "type": "object",
      "description": "Metadata used to document a component.",
      "properties": {
        "access": {
          "type": "string",
          "enum": [
            "private",
            "public"
          ],
          "description": "Specifies whether the component is public or
private."
        },
        "description": {
          "type": "string",
          "description": "A brief description of the component and its
purpose."
        },
        "deprecated": {
          "type": "string",
          "pattern": "\\d+\\.\\d+\\.\\d+ - .+",
          "description": "If the component is deprecated, set this field
to a string in the following format: \"<version number> - <text listing
alternatives>\"."
        },
        "since": {
          "type": "string",
          "pattern": "\\d+\\.\\d+\\.\\d+",
          "description": "The version in which the component was
introduced."
        },
        "moreInfo": {
          "type": "string",
          "description": "A relative path to a markdown file containing
additional information and examples related to the component."
        },
        "controls": {
          "type": "string",
          "description": "A relative path to the controls file which
defines the controls corresponding to the component."
        }
      },
      "additionalProperties": false
    }
  },
  "__defs": []
}
```


## chartPointsConfigDef Definition (ViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `seriesType` | string | | The type of series, for example line or column or pie or scatter or spline, etc |
| `seriesMarker` | SeriesMarkerObject | | Options for the point markers of series. Properties like radius, symbol, lineColor and lineWidth define the visual appearance of the markers. |
| `seriesTooltip` | SeriesTooltipObject | | Options for the tooltip of series. Properties like headerFormat, pointFormat define the visual appearance of the tooltip for markers. |
| `yAxis` | number | | When using dual or multiple y axes, this number defines which yAxis the particular series is connected to. |
| `dataLabels` | DataLabelsObject | | Options for the series data labels, appearing next to each data point. |
| `showInLegend` | boolean | | Show series name in the legend. |
| `isDataLabelOnChartEnabled` | boolean | | show labels on chart. |
| `seriesColorClass` | string | | Specify class which has custom colors. |
| `dashStyle` | string | | Represents the line style of the series for line charts. |

### Referenced Objects

The following objects are referenced within this definition.

### DataLabelsObject

Options for the series data labels, appearing next to each data point.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### SeriesMarkerObject

Options for the point markers of series. Properties like radius, symbol, lineColor and lineWidth define the visual appearance of the markers.

### SeriesTooltipObject

Options for the tooltip of series. Properties like headerFormat, pointFormat define the visual appearance of the tooltip for markers.

## chartProviderDef Definition (ViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `title` | string | | |
| `chartColorOverrideClass` | string | | |
| `chartType` | string | Yes | It can be set to one of the following values: `column`, `line`, `pie`, `heatmap`, `scatter`, `spline`. |
| `loadDataAction` | string | | |
| `selectAction` | string | | |
| `unSelectAction` | string | | |
| `chartConfig` | ChartConfigObject | | |
| `chartPointsConfig` | ArrayChartPointsConfigObject | | undefined. It must contain only unique items. |

### Referenced Objects

The following objects are referenced within this definition.

### ChartConfigObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `isChartZoomable` | boolean | | |
| `isYAxisLinearOrLogarithmic` | string | | It can be set to one of the following values: `linear`, `logarithmic`. |
| `size` | string | | |
| `xAxisLabel` | string | | |
| `yAxisLabel` | string | | |
| `isDataLabelOnChartEnabled` | boolean | | |
| `spacingLeft` | number | | Number of pixels to offset chart to the left. Used by highchart and ignored by echarts. |
| `showAxes` | boolean | | Hides axes when no data for highchart, ignored by echarts. |
| `tooltip` | **TooltipObject** | | tooltip data is a JSON object. It uses the same format as the highchart/echarts chart config tooltip option. |
| `xAxis` | **XAxisObject** | | xAxis data is a JSON object. It uses the same format as the highchart/ echarts chart config xAxis option. |
| `yAxis` | **YAxisObject** | | yAxis data is a JSON object. It uses the same format as the highchart/ echarts config yAxis option. |
| `plotOptions` | **PlotOptionsObject** | | plotOptions data is a JSON object. It is the same as highchart config plotOptions object. Not used for echart. |
| `legend` | **LegendObject** | | legend data is a JSON object. It uses the same format as the highchart/echarts chart config legend option. |
| `dataLabels` | **DataLabelsObject** | | Options for the series data labels, appearing next to each data point. |


Unpublished work. © 2025 Siemens

### ChartPointsConfigObject

*No description specified*.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | chartPointsConfigDef | | |

### DataLabelsObject

Options for the series data labels, appearing next to each data point.

### LegendObject

legend data is a JSON object. It uses the same format as the highchart/echarts chart config legend option.

### PlotOptionsObject

plotOptions data is a JSON object. It is the same as highchart config plotOptions object. Not used for echart.

### TooltipObject

tooltip data is a JSON object. It uses the same format as the highchart/echarts chart config tooltip option.

### XAxisObject

xAxis data is a JSON object. It uses the same format as the highchart/echarts chart config xAxis option.

### YAxisObject

yAxis data is a JSON object. It uses the same format as the highchart/echarts config yAxis option.

## commandDef Definition (ViewModel)

*No description specified*.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `id` | string | | |
| `title` | string | | |
| `areas` | array | | |
| `priorities` | array | | |
| `action` | string | | |
| `dependencies` | array | | |
| `displayOption` | string | | It can be set to one of the following values: `ON_HOVER`, `ON_SELECTION`, `ON_HOVER_AND_SELECTION`, `ALWAYS`. |
| `position` | string | | It can be set to one of the following values: `TOP_RIGHT`, `MIDDLE_RIGHT`, `BOTTOM_RIGHT`, `TOP_LEFT`, `MIDDLE_LEFT`, `BOTTOM_LEFT`. |

## dataParseDef Definition (ViewModel)

This definition must satisfy *one* of the following constraints:

- it must be set to a **DataParseDefObject1** value
- it must be set to a **DataParseDefObject2** value

### Referenced Objects

The following objects are referenced within this definition.

#### DataFilterObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | Yes | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### DataParseDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `path` | string | | |
| `dataInput` | string | | |
| `outputFormatType` | string | | |
| `operationType` | string | | |
| `dataFilter` | Array**DataFilterObject** | | |
| `viewModelPropValues` | **ViewModelPropValuesObject** | | |
| `outputFormat` | Array**OutputFormatAltObject** | | |
| `filterOtherData` | **FilterOtherDataObject** | | |

### DataParseDefObject2

*No description specified.*

**Note**
This object *can* have additional properties.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `props` | **PropsAltObject** | Yes | |

### DbValueObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | Yes | |
| `value` | string | Yes | |

### FilterOtherDataObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `path` | string | Yes | |
| `dataFilter` | Array**DataFilterObject** | Yes | |

### OutputFormatAltObject

*No description specified.*

This value must satisfy *any* of the following constraints:

- it must be set to a **fieldDef** value
- it must be set to a **structDef** value

### PropMiscObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `propName` | string | Yes | |
| `value` | string | Yes | |

### PropsAltObject

*No description specified.*


Siemens Web Framework
Unpublished work. © 2025 Siemens

This value must satisfy *one* of the following constraints:

- it must be set to a **PropsObject1** value
- it must be set to a string value

### PropsObject1

*No description specified.*

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z0-9_]*$` | **dataMapperEachPropDef** | | |

### TypeObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `condition` | string | Yes | |
| `value` | string | Yes | |

### ViewModelPropValuesObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `type` | Array**TypeObject** | Yes | |
| `displayName` | string | Yes | |
| `dbValue` | Array**DbValueObject** | Yes | |
| `propMisc` | Array**PropMiscObject** | Yes | |

## dataProvidersDef Definition (ViewModel)

This definition must satisfy *any* of the following constraints:

- it must be set to a **DataProvidersDefObject1** value
- it must be set to a **DataProvidersDefObject2** value

### Referenced Objects

The following objects are referenced within this definition.

### CommandsObject

*No description specified.*

### DataProvidersDefObject1

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `action` | string | Yes | |
| `initializeAction` | string | | |
| `nextAction` | string | | |
| `previousAction` | string | | |
| `focusAction` | string | | |
| `expandAction` | string | | |
| `commands` | **CommandsObject** | | |
| `enablePropEdit` | string OR boolean | | Enables the ability to edit table cells without having to go through the Edit Command. Click twice on any editable table cell to edit the cell. The possible values of this option are true, false or a declarative condition. False, will disable this behavior. True will enable this behavior (default). A declarative condtion will allow True/False to be set based on the conditions resolved value. If the "Autosave Tables" toggle is enabled in the user settings |


Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | panel, changes to the table cells will be saved instantly on blur. If the toggle is disabled, the user has to explicitly save the edits using the save command. |
| `editContext` | string | | Specifying editContext will create an edit handler with the given editContext for this Data Provider. |
| `response` | string | | |
| `totalFound` | string | | |
| `totalLoaded` | string | | |
| `uidInResponse` | string | | |
| `preSelection` | boolean | | |
| `selectionModelMode` | string | | It can be set to one of the following values: `multiple`, `single`. |

#### DataProvidersDefObject2

*No description specified*.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `name` | string | | |
| `displayName` | string | | |
| `minWidth` | integer | | |
| `maxWidth` | integer | | |
| `width` | integer | | |
| `enableColumnMoving` | boolean | | |
| `enableColumnMenu` | boolean | | |
| `enableColumnResizing` | boolean | | |
| `enableFiltering` | boolean | | |
| `isFilteringEnabled` | boolean | | |
| `enableHiding` | boolean | | |
| `enablePinning` | boolean | | |

| | | | |
|---|---|---|---|
| `enableSorting` | boolean | | |
| `headerTooltip` | boolean | | |
| `pinnedLeft` | boolean | | |

## editConfigDef Definition (ViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `startEditAction` | string | Yes | |
| `saveEditAction` | string | Yes | |
| `identifier` | string | Yes | |

## i18nDef Definition (ViewModel)

All user facing text should be localized unless a web client solution supports only one language. The View refers to localizations defined in the 'i18n' section of the View Model. You can specify the actual string to be displayed (however, it would not be localized) OR the name of JSON bundle which holds the localized text to be shown.

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `^[a-zA-Z_]+[a-zA-Z0-9_]*$` | array OR string | | |

## gridDef Definition (ViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `addIconColumn` | boolean | | |
| `columnProvider` | string | | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `dataProvider` | string | Yes | |
| `deps` | string | | |
| `enableArrangeMenu` | boolean | | |
| `gridOptions` | GridOptionsObject | | |
| `pinIconColumn` | boolean | | |
| `propertyProvider` | string | | |

### Referenced Objects

The following objects are referenced within this definition.

#### GridOptionsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `enableGridMenu` | boolean | | |
| `enableFiltering` | boolean | | |
| `isFilteringEnabled` | boolean OR string | | |
| `enableHeaderIcon` | boolean | | |
| `enablePinning` | boolean OR string | | |
| `enableSorting` | boolean | | |
| `headerHeight` | string OR number | | |
| `rowHeight` | number OR string | | |
| `textBundle` | TextBundleObject | | |
| `transpose` | boolean | | |
| `showContextMenu` | boolean | | |

#### TextBundleObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `TwistieTooltipCollapsed` | string | | |
| `TwistieTooltipExpanded` | string | | |

## portsDef Definition (ViewModel)

Observables defined in the viewModels.

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `onChangeAction` | string | | Name of the action which will be auto-invoked when the data associated with the port is changed |
| `syncObject` | SyncObjectObject | Yes | Data passed from source to target |

### Referenced Objects

The following objects are referenced within this definition.

#### SyncObjectObject

Data passed from source to target

## workspaceDef Definition (ViewModel)

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | | Version of the schema it complies with |
| `workspaceId` | string | Yes | |
| `workspaceName` | WorkspaceNameObject | Yes | |
| `workspaceType` | string | Yes | |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| `scope` | string | | |
| `defaultPage` | string | Yes | |
| `availablePages` | Arraystring | | undefined. It must contain at least 1 unique item. |

#### Referenced Objects

The following objects are referenced within this definition.

#### WorkspaceNameObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `source` | string | | |
| `key` | string | | |

## Workspace Schema

This is the schema the declarative workspace (in JSON format) must adhere to. The Apollo audit check validates all workspace JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | Yes | Version of the schema complies with |
| `workspaceId` | string | Yes | A unique identifier strin the workspace |
| `workspaceName` | **WorkspaceNameObject** | Yes | |
| `workspaceType` | string | Yes | It can be set to one of t following values: `Incl` `Exclusive`. A string v specifying the type of |

| | | | |
|---|---|---|---|
| | | | workspace.The allowed are - 'Inclusive','Exclusiv |
| `defaultPage` | string | Yes | The page which will be by default when user lo changes workspace |
| `includeInSolutions` | Arraystring | | undefined. It must cont least 1 unique item. |
| `availablePages` | Arraystring | | undefined. It must cont least 1 unique item. |
| `includedCommands` | Arraystring | | undefined. It must cont only unique items. |
| `excludedCommands` | Arraystring | | undefined. It must cont only unique items. |
| `availableNavigations` | Array**AvailableNavigationsObject** | | An array of json object defining configurable b for actions. It must con least 1 unique item. |
| `availableContextConfigurations` | Arraystring | | It's an Optional parame and it's an array of strin where each value is the contextId for each of lin which would show-up i context control while w in a specific workspace value is the state define in contextConfiguration availableContextConfig is not defined, then all of the contextIds/ contextConfiguration a in the kit would show u must contain at least 1 item. |
| `scope` | string | | scope of the workspace |
| `conditions` | **ConditionsObject** | | Conditions provide live evaluation. They can ev to true/false. They can in visible-when clauses can be used for event h (onEvent handler). The value can be live evalua condition. The conditio refer to view model sta |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | can update view model<br>They can leverage Bool<br>condition expressions l<br>' |
| `settings` | SettingsObject | | Additional workspace s<br>which can be supplied<br>and read at runtime. |

## Referenced Objects

The following objects are referenced within this schema.

### AvailableNavigationsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `key` | string | | action id |
| `page` | string | | The id of the page to navigate to |
| `params` | ParamsObject | | The set of parameters that are added to URL when user is navigated to designated page |
| `activeWhen` | conditionOrBoolDef | | The name of the declarative condition which decides which navigation configuration to pick |

### ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b` | Condition | | |

### ParamsObject

The set of parameters that are added to URL when user is navigated to designated page

### SettingsObject

Additional workspace settings which can be supplied to ctx and read at runtime.

### WorkspaceNameObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| source | string | | |
| key | string | | |

### JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeWorkspaceSchema-1.0.0.json",
  "description": "This is the schema the declarative workspace (in JSON
format) must adhere to. The Apollo audit check validates all workspace JSON
files against this schema. For explanation of the keywords of this schema,
please visit https://ajv.js.org/keywords.html",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "type": [
        "string",
        "integer"
      ],
      "description": "Version of the schema it complies with"
    },
    "workspaceId": {
      "type": "string",
      "description": "A unique identifier string for the workspace"
    },
    "workspaceName": {
      "type": [
        "string",
        "object"
      ],
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
      "additionalProperties": false,
      "properties": {
        "source": {
          "type": "string"
        },
        "key": {
          "type": "string"
        }
      }
    },
    "workspaceType": {
      "type": "string",
      "description": "A string value specifying the type of workspace.The
allowed values are - 'Inclusive','Exclusive' ",
      "enum": [
        "Inclusive",
        "Exclusive"
      ]
    },
    "defaultPage": {
      "type": "string",
      "description": "The page which will be shown by default when user
logs in / changes workspace"
    },
    "includeInSolutions": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "availablePages": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "includedCommands": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "excludedCommands": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
```


Unpublished work. © 2025 Siemens

```
    "availableNavigations": {
      "type": "array",
      "description": "An array of json object defining configurable
behavior for actions",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "key": {
            "type": "string",
            "description": "action id"
          },
          "page": {
            "type": "string",
            "description": "The id of the page to navigate to"
          },
          "params": {
            "type": "object",
            "description": "The set of parameters that are added to URL
when user is navigated to designated page"
          },
          "activeWhen": {
            "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
            "description": "The name of the declarative condition which
decides which navigation configuration to pick"
          }
        }
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "availableContextConfigurations": {
      "type": "array",
      "description": "It's an Optional parameter and it's an array
of strings where each value is the contextId for each of link
which would show-up in context control while working in a specific
workspace. Each value is the state defined in contextConfiguration.json.If
availableContextConfigurations is not defined, then all of the contextIds/
contextConfiguration available in the kit would show up.",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "scope": {
      "type": "string",
      "description": "scope of the workspace"
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and
can be used for event handling (onEvent handler). The data value can
be live evaluated by condition. The conditions can refer to view model
```


Siemens Web Framework
Unpublished work. © 2025 Siemens

```
state and can update view model state. They can leverage Boolean condition
expressions like '&&', '||', '==', '!='",
      "patternProperties": {
        "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    },
    "settings": {
      "type": "object",
      "description": "Additional workspace settings which can be supplied
to ctx and read at runtime."
    }
  },
  "required": [
    "schemaVersion",
    "workspaceId",
    "workspaceName",
    "workspaceType",
    "defaultPage"
  ],
  "__defs": []
}
```

## WorkspaceContribution Schema

This is the schema the declarative workspace (in JSON format) must adhere to. The Apollo audit check validates all workspace JSON files against this schema. For explanation of the keywords of this schema, please visit **https://ajv.js.org/keywords.html**

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `schemaVersion` | string OR integer | Yes | Version of the schema complies with |
| `workspaceId` | string | Yes | A unique identifier strin the workspace |
| `availablePages` | Arraystring | | An array of string value where each value is the page name. Supports u `kit::kitName` forma include all pages from t It must contain only un items. |
| `includedCommands` | Arraystring | | An array of commands each value is the comm |

| | | | |
|---|---|---|---|
| | | | which needs to be seer the given workspace. S using `kit::kitName` to include all commanc that kit. It must contair unique items. |
| `excludedCommands` | Arraystring | | An array of commands each value is the comm Id which the user can exclude inside the give workspace. Supports u `kit::kitName` forma include all commands f that kit. It must contair unique items. |
| `availableNavigations` | Array**AvailableNavigationsObject** | | An array of json object defining configurable b for actions. It must con least 1 unique item. |
| `availableContextConfigurations` | Arraystring | | It's an Optional parame and it's an array of strir where each value is the contextId for each of li which would show-up i context control while w in a specific workspace value is the state define in contextConfiguration availableContextConfig is not defined, then all of the contextIds/ contextConfiguration a in the kit would show u must contain at least 1 item. |
| `conditions` | **ConditionsObject** | | Conditions provide live evaluation. They can ev to true/false. They can in visible-when clauses can be used for event h (onEvent handler). The value can be live evalua condition. The conditio refer to view model sta can update view model They can leverage Bool |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | | |
|---|---|---|---|
| | | | condition expressions l<br>' |

## Referenced Objects

The following objects are referenced within this schema.

### AvailableNavigationsObject

*No description specified.*

This object can contain the following properties:

| Name | Type | Required | Description |
|---|---|---|---|
| `key` | string | | action id |
| `page` | string | | The id of the page to navigate to |
| `params` | ParamsObject | | The set of parameters that are added to URL when user is navigated to designated page |
| `activeWhen` | conditionOrBoolDef | | The name of the declarative condition which decides which navigation configuration to pick |

### ConditionsObject

Conditions provide live logic evaluation. They can evaluate to true/false. They can be used in visible-when clauses and can be used for event handling (onEvent handler). The data value can be live evaluated by condition. The conditions can refer to view model state and can update view model state. They can leverage Boolean condition expressions like '&&', '||', '==', '!='

You can define properties on this object with a name matching one of the following patterns:

| Pattern | Type | Required | Description |
|---|---|---|---|
| `\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\b` | Condition | | |

### ParamsObject

The set of parameters that are added to URL when user is navigated to designated page

## JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "declarativeWorkspaceContributionSchema-1.0.0.json",
  "description": "This is the schema the declarative workspace (in JSON
format) must adhere to. The Apollo audit check validates all workspace JSON
files against this schema. For explanation of the keywords of this schema,
please visit https://ajv.js.org/keywords.html",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "schemaVersion": {
      "type": [
        "string",
        "integer"
      ],
      "description": "Version of the schema it complies with"
    },
    "workspaceId": {
      "type": "string",
      "description": "A unique identifier string for the workspace"
    },
    "availablePages": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true,
      "description": "An array of string values where each value is the
page name. Supports using `kit::kitName` format to include all pages from
that kit."
    },
    "includedCommands": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true,
      "description": "An array of commands where each value is the command
Id which needs to be seen inside the given workspace. Supports using
`kit::kitName` format to include all commands from that kit."
    },
    "excludedCommands": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true,
      "description": "An array of commands where each value is the command
Id which the user can exclude inside the given workspace. Supports using
`kit::kitName` format to include all commands from that kit."
```


Unpublished work. © 2025 Siemens

```
    },
    "availableNavigations": {
      "type": "array",
      "description": "An array of json object defining configurable
behavior for actions",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
          "key": {
            "type": "string",
            "description": "action id"
          },
          "page": {
            "type": "string",
            "description": "The id of the page to navigate to"
          },
          "params": {
            "type": "object",
            "description": "The set of parameters that are added to URL
when user is navigated to designated page"
          },
          "activeWhen": {
            "$ref": "declarativeReusableSchemaDefs-1.0.0.json#/definitions/
conditionOrBoolDef",
            "description": "The name of the declarative condition which
decides which navigation configuration to pick"
          }
        }
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "availableContextConfigurations": {
      "type": "array",
      "description": "It's an Optional parameter and it's an array
of strings where each value is the contextId for each of link
which would show-up in context control while working in a specific
workspace. Each value is the state defined in contextConfiguration.json.If
availableContextConfigurations is not defined, then all of the contextIds/
contextConfiguration available in the kit would show up.",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "conditions": {
      "type": "object",
      "description": "Conditions provide live logic evaluation. They can
evaluate to true/false. They can be used in visible-when clauses and
can be used for event handling (onEvent handler). The data value can
be live evaluated by condition. The conditions can refer to view model
state and can update view model state. They can leverage Boolean condition
expressions like '&&', '||', '==', '!='",
      "patternProperties": {
```


Unpublished work. © 2025 Siemens

```
        "\\b[a-z]+[A-Z]*([A-Z]*[a-z]*[0-9]*)*\\b": {
          "$ref": "declarativeConditionSchema-1.0.0.json"
        }
      }
    }
  },
  "required": [
    "schemaVersion",
    "workspaceId"
  ],
  "__defs": []
}
```

# Services

This section contains the reference documentation of all the public Siemens Web Framework services.

## js/appCtxService

Package: **@swf/engine**

This service provides helpful APIs to register/unregister/update context variables used to hold application state.

### Properties

| Name | Type | Description |
|---|---|---|
| ctx | object | Object holding the entire Application Context. |

### Methods

#### registerCtx(name: *string*, value: *object*)

Register application context variable

### Parameters

| Name | Type | Description |
|---|---|---|
| name | string | The name of context variable |
| value | object | The value of context variable |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### unRegisterCtx(path: *string*)

Unregister application context variable

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | string | The name of context variable |

### updateCtx(path: *string*, value: *object*)

Update application context and Announce app context update by publishing an event 'appCtx.update' with eventData as {"name": ctxVariableName, "value": ctxVariableValue}

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | string | The name of context variable |
| value | object | The value of context variable |

### getCtx(path: *string*) → *object*

Get application context variable value

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | string | Path to the context |

#### Returns

object Value (if any) at the indicated context path location.

### updateCtxFromObject(ctxObject: *object*)

**Note**
6.1.0 - Use the `updateCtx()` method instead.

Update app context from object

### Parameters

| Name | Type | Description |
|---|---|---|
| ctxObject | object | The source context object containing properties that need to be update on global ctx |

## js/awHttpService

Package: **@swf/engine**

This service provides a way to facilitate communication with the remote HTTP servers.

### Properties

| Name | Type | Description |
|---|---|---|
| instance | awHttpInstance | Object providing access to http apis. |

## js/awParseService

Package: **@swf/engine**

This service accepts an expression string and returns a function that can be used to parse and evaluate the expression.

### Properties

| Name | Type | Description |
|---|---|---|
| instance | awParseServiceInstance | A callback that accepts the expression to parse. |

## js/awStateService

Package: **@swf/engine**

This service provides apis related to state routing.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Properties

| Name | Type | Description |
|---|---|---|
| instance | AwStateInstance | Object providing access to the state routing apis. |

## js/bannerService

Package: **@swf/commonframe**

This service provides the ability to show banners at the top of every page of your application.

### Methods

#### showBanner(bannerContext: *Array.BannerContext*)

Processes the list of banners, validates each banner and shows valid banners.

### Parameters

| Name | Type | Description |
|---|---|---|
| bannerContext | Array.BannerContext | A list of banner context to process, each defining a banner to display. |

## js/browserUtils

Package: **@swf/engine**

This module provides reusable functions related to handling URL manipulation.

### Properties

| Name | Type | Description |
|---|---|---|
| cachedBaseURL | object | Cached Base URL for the current application |

### Methods

### getBaseURL() → *string*

Returns Base URL for the current application

### Returns

string Base URL for the current applications root document without any query or location attributes and (if otherwise valid) with a trailing / assured

### updateBrowserUrl(newLocale: *string*)

Updates the URL with locale information

### Parameters

| Name | Type | Description |
|---|---|---|
| newLocale | string | The locale value which needs to be set e.g. 'en_US' |

### updateBrowserUrlAttribute(key: *string*, value: *string*)

Updates the URL with given attribute

### Parameters

| Name | Type | Description |
|---|---|---|
| key | string | The key which needs to be set e.g. 'locale' |
| value | string | The value which needs to be set e.g. 'en_US' |

### removeUrlAttribute(attrToRemove: *string*)

Remove the given named attribute from the 'search' area (i.e. URL attributes before the '#') of the browser's current URL.

### Parameters

| Name | Type | Description |
|---|---|---|
| attrToRemove | string | Name of the attribute to remove e.g. 'locale'. |

### getBrowserType() → *String*


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Returns

string - string containing the browser and version i.e. Chrome 112 https://stackoverflow.com/questions/2400935/browser-detection-in-javascript

## js/clipboardService

Package: **@swf/engine**

Please refer Clipboard

### Properties

| Name | Type | Description |
|---|---|---|
| delegateService | object | The clipboard service instance as configured in SolutionDef |

### Methods

#### setContents(contentsToSet: *array*)

Sets the current contents of the clipboard.

### Parameters

| Name | Type | Description |
|---|---|---|
| contentsToSet | array | Array of Objects to set as the current clipboard contents |

#### getCachableObjects() → *array*

Return the content of the clipboard that is cached.

### Returns

array Array of current Objects that is cached.

#### copyHyperlinkToClipboard(content: *array*) → *boolean*

Copies hyperlink to OS clipboard

### Parameters

| Name | Type | Description |
|---|---|---|
| content | array | Array of selected object whose hyperlink is created and copied to os clipboard |

### Returns

boolean successful Whether the content was successfully copied to the clipboard or not

## js/conditionService

Package: **@swf/engine**

This service provides the APIs to evaluate the declarative condition expression using given viewmodel context

### Methods

#### evaluateCondition(declViewModel: *DeclarativeViewModel*, expression: *string*, evaluationEnv: *object*, depModuleObj: *object*) → *boolean*

Evaluate condition expression and ensure a boolean is returned

### Parameters

| Name | Type | Description |
|---|---|---|
| declViewModel | DeclarativeViewModel | The 'declViewModel' context to evaluate the condition within. |
| expression | string | Expression |
| evaluationEnv | object | The data environment for expression evaluation. |
| depModuleObj | object | The module object who's functions can be used in expression evaluation. |

### Returns

boolean The evaluated condition result.


Siemens Web Framework
Unpublished work. © 2025 Siemens

# js/configurationService

Package: **@swf/engine**

Please refer **Extensions and Configuration Points**

## Methods

### notifyConfigChange(path: *String*, updateLocalStorage: *Boolean*)

Notify that a configuration piece has changed and the local cache should be cleared

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | string | Name of the Configuration (e.g. 'solutionDef') |
| updateLocalStorage | boolean | Whether to update local storage. Defaults to true. |

### getCfg(path: *string*, noCache: *boolean*) → *Promise.object*

Get configuration data for specified configuration path.

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | string | Name of the Configuration (e.g. 'solutionDef') |
| noCache | boolean | Do not use cache |

#### Returns

Promise. promise This would resolve to configuration json

# js/dateTimeService

Package: **@swf/engine**

This service provides useful APIs for the date and time formatting. It allows to set the default date and time format for the session.

## Methods

### formatDate(dateToFormat: *number*) → *string*

This method formats the given Date object or UNIX epoch value to a date string in the default d-M-Y format.

### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents MS since UNIX epoch. |

### Returns

string The formatted date of the given Date object formatted according to current user session format.

### formatDateTime(dateToFormat: *number*) → *string*

This method formats given date object or UNIX epoch value to a datetime string in thecurrent user session format.

### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents MS since UNIX epoch. |

### Returns

string The formatted date time of the given Date object formatted according to current user session format.

### formatTime(dateToFormat: *number*) → *string*

This method formats given date object or UNIX epoch value to a time string in thecurrent user session format.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents MS since UNIX epoch. |

### Returns

string The formatted time of the given Date object formatted according to current user session format.

## formatSessionDateTime(dateToFormat: *number*) → *string*

This method formats given date object or UNIX epoch value to a date and time string according to the current user session format.

### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents the milliseconds since UNIX epoch. |

### Returns

string The date time of the given Date object formatted according to current user session format.

## formatSessionDate(dateToFormat: *number*) → *string*

This method formats given date object or UNIX epoch value to a date string according to the current user session format.

### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents milliseconds since UNIX epoch. |

### Returns

string The date time of the given Date object formatted according to current user session format.


Unpublished work. © 2025 Siemens

### formatSessionTime(dateToFormat: *number*) → *string*

This method formats given date object or UNIX epoch value to a time string according to the current user session format.

#### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents milliseconds since UNIX epoch. |

#### Returns

string The time portion of the given Date object formatted according to current user session format.

### formatUTC(dateToFormat: *number*) → *string*

This method returns the formatted date and time of the given date object or UNIX epoch value according to UTC format.

#### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents MS since UNIX epoch. |

#### Returns

string The formatted date and time of the given Date object formatted according to UTC format.

### formatNonStandardDate(dateToFormat: *number*, dateFormat: *string*) → *string*

This method returns the date & time of the given date object or UNIX epoch value formatted according to the input dateFormat.

#### Parameters

| Name | Type | Description |
|---|---|---|
| dateToFormat | number | A number or Date or any other object that its 'valueOf' will result in a number that represents MS since UNIX epoch. |
| dateFormat | string | This is a EWI data collection object time stamp format. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Returns

string The date and time of the given Date object formatted according to the input dateFormat.

## getDateFormat() → *string*

Returns the format compatible with the 'dateFormat' property.

### Returns

string Date entry format for use in the datepicker API.

## getDateFormatPlaceholder() → *string*

Returns the user-oriented text used to indicate the required format of the date entry.

### Returns

string User-oriented text used to indicate the required format of the date entry (i.e. yy.mm.dd, day-month-year, etc.).

## getJSDate(dateToTest: *object*) → *Date*

Returns a new Date object based on the given Date object or value. This method handles some corner cases found in (at least) the Firefox browser. A *null date* did not always create a valid Date object.

### Parameters

| Name | Type | Description |
|---|---|---|
| dateToTest | object | Object that MAY be a JS Date or MAY be the number of milliseconds since 'epoch'. |

### Returns

Date A new Date object based on the given object (or null date if no other JS date possible).

## getNullDate() → *Date*

Returns the cached Date object set with the same date/time as the special 'null' or 'not yet set' value.

### Returns

Date The cached Date object set with the same date/time as the special null or not yet set value.

### getSessionDateTimeFormat() → *string*

Returns a session datetime format to display a date & time entry in.

#### Returns

string A session datetime format to display a date time entry in.

### getSessionDateFormat() → *string*

Returns a session date format to display a date entry in.

#### Returns

string A session date format to display a date entry in.

### getSessionTimeFormat() → *string*

Returns a session time format to display a time entry in.

#### Returns

string A session time format to display a time entry in.

### isNullDate(dateToTest: *object*) → *boolean*

Determines whether the specified date is null or not yet set.

#### Parameters

| Name | Type | Description |
|---|---|---|
| dateToTest | object | A number or Date or any other object that its 'valueOf' will result in a number that represents MS since UNIX epoch. |

#### Returns

boolean TRUE if the given test object represents the same date/time as the special null or not yet set value.

### compare(date1: *Date*, date2: *Date*) → *number*

Compares two dates.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| date1 | Date | First date to compare. |
| date2 | Date | Second date to compare. |

### Returns

number The value 0 if the date2 is equal to date1; a value less than 0 if date1 is less than date2; and a value greater than 0 if date1 is greater than date2.

### getMsTimeFromUiValue(uiValue: *) → *number*

### Parameters

| Name | Type | Description |
|---|---|---|
| uiValue | * | The uiValue from the ViewModelProperty. |

### Returns

number Time in milliseconds

## js/dialogService

Package: **@swf/essentials**

This service provides the methods to manage the dialogs.

### Methods

### closeDialog(editContext: *object*, popupId: *string*)

Closes a dialog.


Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| editContext | object | the editContext if any |
| popupId | string | the unique PopupId to be closed |

### updateDialogDimensions(popupId: *object*, newOptions: *string*)

updates dimensions.

### Parameters

| Name | Type | Description |
|---|---|---|
| popupId | object | the unique PopupId to be updated |
| newOptions | string | set of new dimensions i.e width, height |

## js/eventBus

Package: **@swf/engine**

This service manages the subscription and publishing of the events. It is based on the 'PostalJS' API.

### Methods

### subscribe(topic: *string*, callbackFn: *EventBusEventCallback*, subId: *string*) → *object*

Subscribe to the given 'topic' on the event channel. When the event is published, the given function will be invoked and passed the 'eventData'.

### Parameters

| Name | Type | Description |
|---|---|---|
| topic | string | Topic to subscribe to. A '#' character is interpreted as a wildcard |
| callbackFn | EventBusEventCallback | Function to be invoked and passed 'eventData' when the event is published |
| subId | string | An optional ID to associate with this subscription |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Returns

object A PostalJS SubscriptionDefinition object. Needed while unsuscribing the event topic using unsubscribe

### publish(topic: *string*, eventData: *object*, isCustomEvent: *boolean*)

Publish the given 'topic' on the event channel. The given 'eventData' will be passed to each subscribers callback function.

### Parameters

| Name | Type | Description |
|---|---|---|
| topic | string | Topic to publish |
| eventData | object | Optional data to pass to the subscribed callback functions |
| isCustomEvent | boolean | True if event stated in *viewModel.json |

### unsubscribe(subDef: *object*)

Removes the given subscription from the event channel so that the callback will no longer be invoked.

### Parameters

| Name | Type | Description |
|---|---|---|
| subDef | object | A 'PostalJS' SubscriptionDefinition object returned by subscribe |

## js/iconService

Package: **@swf/engine**

Please refer **Solution Configuration**

### Methods

### getIcon(iconName: *string*) → *string*

Returns the HTML SVG string for the given icon name **without** any 'class' attribute being set.


Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| iconName | string | The icon name to get an icon for |

### Returns

string SVG definition string for the icon (or null if the icon name has not been registered as an alias in a module.json or the SVG file was not found during war the build)

## js/localStorage

Package: **@swf/engine**

This module defines and provides a set of utility APIs used to access the 'localStorage' APIs of the browser the client is running within.

### Methods

#### subscribe(topic: *string*, cb: *LocalStorageEventCallback*)

Subscribes to 'storage' event for given storage topic (key)

### Parameters

| Name | Type | Description |
|---|---|---|
| topic | string | Local storage topic (key) |
| cb | LocalStorageEventCallback | Event handler |

#### publish(topic: *string*, data: *string*)

Add data to local storage for given storage topic (key)

### Parameters

| Name | Type | Description |
|---|---|---|
| topic | string | Local storage topic (key) |
| data | string | Data to add to local storage |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### get(topic: *string*) → *string*

Get the value of gven local storage topic (key)

#### Parameters

| Name | Type | Description |
|---|---|---|
| topic | string | Local storage topic (key) |

#### Returns

string value Value of local storage (or NULL if the topic is not in the local storage)

### removeItem(topic: *string*)

Remove the given local storage topic (key)

#### Parameters

| Name | Type | Description |
|---|---|---|
| topic | string | The local storage topic (key) to remove |

## js/localeService

Package: **@swf/engine**

This service provides helpful APIs to work with the localization of the text strings.

#### Methods

### getLocale() → *string*

Returns the i18n code for the current user language *with* any country or region code. Note: This value set up-to-date once login is complete. Until then, it will return a default locale code of 'en_US'.

#### Returns

string Current Locale value e.g. en_US


Unpublished work. © 2025 Siemens

### getLocalizedText(resource: *string*, key: *string*, useNative: *boolean*, noCache: *boolean*, localeCode: *string*) → *Promise.string*

Returns localized text from the given resource with the given key

#### Parameters

| Name | Type | Description |
|---|---|---|
| resource | string | Name of the country-neutral (i.e. w/o country-code or extension) i18n resource to load (or null to return the 'BaseMessages' resource) |
| key | string | The text key for the text which need to be localized |
| useNative | boolean | Use browser native API to execute. Don't use it in return before beyond angular is done |
| noCache | boolean | Control, if we want use the cached Locale value or not |
| localeCode | string | Custom locale code (optional), if you need a localized message bundle/value for some custom locale |

#### Returns

Promise. Promise containing the localized text

### getLocalizedTextFromKey(resource: *string*, localeCode: *string*) → *Promise.string*

Returns localized text from the given resource with the given key

#### Parameters

| Name | Type | Description |
|---|---|---|
| resource | string | Name of the country-neutral (i.e. w/o country-code or extension) i18n resource.key to load |
| localeCode | string | Custom locale code (optional), if you need a localized message bundle/value for some custom locale |

#### Returns

Promise. Promise containing the localized text

### getLoadedText(resource: *object*) → *object*

Returns the cached textBundle resource (or NULL if the bundle has not been cached yet).


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| resource | object | Name of the textBundle to return |

### Returns

object Cached textBundle resource (or NULL if the bundle has not been cached yet).

### getLoadedTextFromKey(resource: *object*) → *string*

Returns the cached textBundle resource (or NULL if the bundle has not been cached yet).

### Parameters

| Name | Type | Description |
|---|---|---|
| resource | object | Name of the textBundle.key to return |

### Returns

string Cached localized text from textBundle

### isLovAutoCompleteBlackList() → *boolean*

### Returns

boolean true if the locale is in the blacklist

## js/logger

Package: **@swf/engine**

This service is used to announce various level logging events to the console.

Expected URL patterns:

- **http://localhost:3000?logLevel=ERROR**
- **http://localhost:3000?logLevel=ERROR&locale=fr**
- **http://localhost:8080/?logLevel=ERROR#/Declarative%20Building%20Blocks/Elements/aw-autofocus**

### Methods

#### error()

Similar to info() but also includes a stack trace from where the method was called.

#### warn()

This method is like info() but also displays a yellow warning icon along with the logged message.

#### info()

This method is identical to debug() except for log level = 'CONFIG'.

#### success()

#### debug()

Displays a message in the console. You pass one or more objects to this method, each of which are evaluated and concatenated into a space-delimited string. The first parameter you pass may contain format specifiers, a string token composed of the percent sign (%) followed by a letter that indicates the formatting to be applied.

#### trace()

Prints a stack trace from the point where the method was called, including links to the specific lines in the JavaScript source.

## js/messagingService

Package: **@swf/engine**

This module is part of declarative UI framework and provides service related to displaying notifications.

### Methods

#### showError(message: *string*, messageData: *string*, messageDefn: *MessageDefn*, buttonsArr: *object*)

Show error message in notification.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| message | string | Error message to show |
| messageData | string | Dynamic data to pass along with message |
| messageDefn | MessageDefn | Message definition |
| buttonsArr | object | Array of buttons as user options |

## showInfo(message: *string*, messageData: *string*, messageDefn: *MessageDefn*, buttonsArr: *object*)

Show informational message in notification.

### Parameters

| Name | Type | Description |
|---|---|---|
| message | string | Informational message to show |
| messageData | string | Dynamic data to pass along with message |
| messageDefn | MessageDefn | Message definition |
| buttonsArr | object | Array of buttons as user options |

## showWarning(message: *string*, buttonsArr: *string*, messageData: *string*, messageDefn: *MessageDefn*)

Show warning message in notification.

### Parameters

| Name | Type | Description |
|---|---|---|
| message | string | Warning message to show |
| buttonsArr | string | – Array of buttons (like: Cancel, Ok) |
| messageData | string | Dynamic data to pass along with message |
| messageDefn | MessageDefn | Message definition |

## setTimeout(messageType: *MessageTypeEnum*, timeoutValue: *number*)

Setting custom notification timeout for a specified message type.

### Parameters

| Name | Type | Description |
|---|---|---|
| messageType | MessageTypeEnum | The type of message to apply the timeout to. |
| timeoutValue | number | The timeout value in milliseconds. |

### setMessageVisibility(messageType: *MessageTypeEnum*, value: *boolean*)

Sets the visibility of a specific type of messages. This method can be used to prevent certain types of messages from being displayed to the user (by default, all messages are displayed).

### Parameters

| Name | Type | Description |
|---|---|---|
| messageType | MessageTypeEnum | The type of message for which the visibility will be set. |
| value | boolean | If set to `false`, messages of the specified type will not be displayed to the user. |

## js/modelPropertyService

Package: **@swf/engine**

This module provides methods related to creation and updation of ViewModelProperty.

### Methods

### updateProperty(parentPath: *string*, attrHolderPropName: *string*, sourceObject: *ViewModelObject*, targetObject: *ViewModelObject*)

Update the property in the 'target' object with the same value as the 'source' object for a specified 'path'.

### Parameters

| Name | Type | Description |
|---|---|---|
| parentPath | string | path to the property in the 'source' & 'target' objects. |
| attrHolderPropName | string | name of the property to be updated. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | |
|---|---|---|
| sourceObject | ViewModelObject | the object to be referred to read the value(s) from. |
| targetObject | ViewModelObject | the 'target' object to be updated. |

### createViewModelProperty(propAttrHolder: *ViewModelPropertyAttributeHolder*) → *ViewModelProperty*

Create a new ViewModelProperty.

#### Parameters

| Name | Type | Description |
|---|---|---|
| propAttrHolder | ViewModelPropertyAttributeHolder | An object having the necessary attributes to construct a ViewModelProperty. |

#### Returns

ViewModelProperty newly created ViewModelProperty object

## js/navigationService

Package: **@swf/engine**

This service provides the methods to allow navigation to the specified url.

#### Methods

### navigate(action: *NavigateAction*, navigationParams: *object*) → *string*

Create link to navigate to the specified object

#### Parameters

| Name | Type | Description |
|---|---|---|
| action | NavigateAction | The declarative action object of actionType `Navigate`. |
| navigationParams | object | The 'navigationParams' object specific for `Navigate` action type. |

#### Returns

string The link generated for navigation.

# js/pageLayoutService

Package: **@swf/pagecomponents**

This module provides helpful APIs for page layouting related services.

## Methods

### changeLayout(sublocationState: *object*, layoutId: *string*)

This method is used to change current page layout (e.g. Awp0SinglePaneLayout, Awp0TwoLeftOneRightLayout )

#### Parameters

| Name | Type | Description |
|---|---|---|
| sublocationState | object | Current layout information like layoutContext, viewModeContext |
| layoutId | string | Unique identifier for layout |

### changeViewMode(viewTupleState: *object*, viewTuple: *object*, sublocationState: *object*, title: *string*)

This method is used to change current viewMode (e.g. list, table, summary)

#### Parameters

| Name | Type | Description |
|---|---|---|
| viewTupleState | object | Current viewMode in tuple format |
| viewTuple | object | Desired viewMode in tuple format |
| sublocationState | object | State of the sublocation |
| title | string | Title of the selected view |

### changeLayoutContext(sublocationState: *object*, layoutContext: *string*)

This method is used to update workarea layout context


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| sublocationState | object | Current layout information like layoutContext, viewModeContext |
| layoutContext | string | Expected layout on workarea |

layoutContext = { layout: { column: [ { pwaViewMode:'list' } ] }, layoutId: 'Awp0SinglePaneLayout' }; |

## js/pasteService

Package: **@swf/engine**

This service is used to manage the configuration of the paste operation.

Please refer to **Solution Configuration** for more information.

### Methods

#### execute(targetObject: *object*, sourceObjects: *array*, relationType: *string*, customPasteContext: *object*) → *Promise.Array.object*

To Paste objects with single relationType

### Parameters

| Name | Type | Description |
|---|---|---|
| targetObject | object | The 'target' Object for the paste |
| sourceObjects | array | Array of 'source' Objects to paste onto the 'target' Object |
| relationType | string | Relation type name |
| customPasteContext | object | Custom paste context |

### Returns

Promise.<Array.object> This promise will be resolved or rejected when the service is invoked and its response data is available.

# js/popupService

Package: **@swf/engine**

This module provides methods related to popup.

## Methods

### hide(popupEl: *object*, excludeId: *string*, force: *boolean*) → *Promise*

Close the popup.

#### Parameters

| Name | Type | Description |
|---|---|---|
| popupEl | object | the popupElement to close. accept native element, css selector, empty string (all popups and dialogs), or 'allPopups'(exclude dialogs) |
| excludeId | string | the popupEl to exclude |
| force | boolean | force to close all popups on current page |

#### Returns

Promise promise with the close result, true or false

### show(options: *PopupOptions*) → *Promise*

Display the popup.

#### Parameters

| Name | Type | Description |
|---|---|---|
| options | PopupOptions | Represents an object to configure a popup. |

#### Returns

Promise promise which is resolved after the popup is opened.

### update(popupEl: *object*) → *Promise*

Schedules an update for the popup. It will run on the next UI update available.


Siemens Web Framework
Unpublished work. © 2025 Siemens

### Parameters

| Name | Type | Description |
|---|---|---|
| popupEl | object | the popupEl/popupRef to update. |

### Returns

Promise promise resolved with updated popup reference.

## js/splmTablePublishedService

Package: **@swf/table**

This module provides helpful APIs for table and tree related services.

### Methods

#### createElement(column: *Column*, ViewModelObject: *ViewModelObject*, tableElem: *DOMElement*, rowElem: *DOMElement*)

Creates and returns a DOMElement for the propertyCell of the passed in view model object (vmo) which defines the row and the given column (columnInfo )

### Parameters

| Name | Type | Description |
|---|---|---|
| column | Column | Declarative columnInfo object |
| ViewModelObject | ViewModelObject | Declarative view model object (e.g. row) |
| tableElem | DOMElement | Table DOMElement |
| rowElem | DOMElement | Row DOMElement |

#### createTreeCellCommandElement(column: *Column*, ViewModelObject: *ViewModelObject*, tableElem: *DOMElement*)

This method is used for creating tree cell commands for PL table

### Parameters

| Name | Type | Description |
|---|---|---|
| column | Column | Declarative columnInfo object |
| ViewModelObject | ViewModelObject | Declarative view model object (e.g. row) |
| tableElem | DOMElement | Table DOMElement |

## createCellCommandElement(column: *Column*, vmo: *ViewModelObject*)

This method is used for creating cell commands for PL Table

### Parameters

| Name | Type | Description |
|---|---|---|
| column | Column | column Definition |
| vmo | ViewModelObject | View model object |

## getTableScrollBar(tableElement: *DOMElement*)

Get table scroll bar

### Parameters

| Name | Type | Description |
|---|---|---|
| tableElement | DOMElement | Table DOMElement |

## addHighlights(displayValue: *string*) → *string*

This method is used for adding highlight

### Parameters

| Name | Type | Description |
|---|---|---|
| displayValue | string | |

### Returns

string displayValue


Siemens Web Framework
Unpublished work. © 2025 Siemens

### createFilter(operation: *string*, columnName: *string*, values: *Array*)

Create a filter based on the operation and values. for example RANGE, GREATER, GREATER_EQUALS, LESS, LESS_EQUALS EQUALS, CASE_SENSITIVE_EQUALS, NOT_EQUALS, CASE_SENSITIVE_NOT_EQUALS, CONTAINS, NOT_CONTAINS, STARTS_WITH, ENDS_WITH

#### Parameters

| Name | Type | Description |
|---|---|---|
| operation | string | operation name of the filter |
| columnName | string | column name of the grid |
| values | array | values of filter input |

### closeColumnMenu(popupRef: *)

Closes the passed in column menu popup

#### Parameters

| Name | Type | Description |
|---|---|---|
| popupRef | * | |

### createColumnInfo(columnProperties: *object*) → *Column*

#### Parameters

| Name | Type | Description |
|---|---|---|
| columnProperties | object | (Optional) An object who's properties will overwrite the default values of the |
| new AwTableColumnInfo object. | | |

#### Returns

**Column** Newly created AwTableColumnInfo object.

### createColumnProvider(declViewModel: *DeclarativeViewModel*, $scope: *object*, commands: *Array*, gridId: *string*, commandsAnchor: *string*) → *ColumnProviderConfiguration*

Create the column provider

### Parameters

| Name | Type | Description |
|---|---|---|
| declViewModel | DeclarativeViewModel | The 'declViewModel' with the properties to use. |
| $scope | object | The AngularJS data context node. |
| commands | array | The list of associated commands |
| gridId | string | The ID of the associated 'declGrid'. |
| commandsAnchor | string | commandAnchor. |

### Returns

ColumnProviderConfiguration Newly created AwTableColumnProvider object.

## setPinRows(dataProvider: *DataProvider*, vmoObj: *ViewModelObject*, position: *Boolean*)

### Parameters

| Name | Type | Description |
|---|---|---|
| dataProvider | DataProvider | The 'dataProvider' with the properties of the view. |
| vmoObj | ViewModelObject | The vmoObj represting the data/contents of the pinned row. |
| position | boolean | The position of the pinned row. If it is set to true or the value is undefined, the pinned row is set in the bottom of the table contents |
| and if it is set to false the pinned row is set on the Top of table contents. | | |

## setContainerHeight(tableElement: *DOMElement*, height: *Number*)

Set/Update the table container height.

### Parameters

| Name | Type | Description |
|---|---|---|
| tableElement | DOMElement | Table DOM Element |
| height | number | The desired height to update the table container. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### reloadTableWithDataProvider(dataProvider: *DataProvider*)

Reloads the data provider and clears json first page.

#### Parameters

| Name | Type | Description |
|---|---|---|
| dataProvider | DataProvider | The data provider to reload. |

## js/splmTablePublishedTreeService

Package: **@swf/table**

This module provides helpful APIs for tree related services.

### Methods

### buildTreeLoadResult(treeLoadInput: *TreeLoadInput*, childNodesIn: *Array.ViewModelTreeNode*, simplePage: *boolean*, startReached: *boolean*, endReached: *boolean*, newTopNode: *ViewModelTreeNode*)

Resolve the row data for the 'next' page of 'children' nodes of the given 'parent'.

#### Parameters

| Name | Type | Description |
|---|---|---|
| treeLoadInput | TreeLoadInput | The original input to the load operation. |
| childNodesIn | Array.ViewModelTreeNode | The 'child' nodes just loaded. |
| simplePage | boolean | TRUE if the 'childNodesIn' represents a 'page' (and NOT an array of all 'child' nodes). |

If TRUE, the nodes will be simply passed along as the result. If FALSE, the page of children will be extracted from this array. | | startReached | boolean | TRUE if the first page of the results has been reached. | | endReached | boolean | TRUE if the last page of the results has been reached. | | newTopNode | ViewModelTreeNode | (Optional) The node to be considered the new top-most 'parent' node as a result of this loading operation. |

### createViewModelTreeNode(nodeId: *string*, type: *string*, displayName: *string*, levelNdx: *number*, childNdx: *number*, iconURL: *string*, svgString: *string*)

### Parameters

| Name | Type | Description |
|---|---|---|
| nodeId | string | Unique ID for this node within the tree-table. 'DocumentRevision', etc.). |
| type | string | The type of model object represented by this tree node (i.e. 'Item' |
| displayName | string | The name to display in the 'navigation' column of the tree-table. |
| levelNdx | number | The # of levels down from the 'root' of the tree-table. |
| childNdx | number | The index to this 'child' within the immediate 'parent'.This information is meant to help in when only a partial (or sparse) range of children have been loaded. |
| iconURL | string | The URL to the icon associated with the display of this node in the 'navigation' column of the tree-table. |
| svgString | string | The String svg icon associated with the display of this node in the 'navigation' column of the tree-table. |

## createPropertyLoadResult(childNodes: *Array.ViewModelTreeNode*)

### Parameters

| Name | Type | Description |
|---|---|---|
| childNodes | Array.ViewModelTreeNode | Array of nodes who's ViewModelProperties have been loaded/ updated. |

## createTreeLoadInput(parentNode: *ViewModelTreeNode*, startChildNdx: *number*, startChildId: *string*, cursorNodeId: *string*, pageSize: *number*, addAfter: *boolean*, rootNode: *ViewModelTreeNode*) → *TreeLoadInput*

### Parameters

| Name | Type | Description |
|---|---|---|
| parentNode | ViewModelTreeNode | The 'parent' ViewModelTreeNode used to access any 'child' nodes in the next level down in the hierarchy. |
| startChildNdx | number | The 'childNdx' to start the 'next' page loading at. |
| startChildId | string | (Optional) The 'id' of the node to start the 'next' page loading at. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

| | | |
|---|---|---|
| cursorNodeId | string | (Optional) ID of an existing node in the 'parent' (and, presumably a ViewModelCollection) to insert any new nodes after (or before) depending on value of 'addAfter'. |
| pageSize | number | (Optional) The maximum # of 'child' nodes to return in any single paged access (Default: See _defaultPageSize). |
| addAfter | boolean | (Optional) TRUE if any new children should be added AFTER the optional 'cursorNodeId' (Default: TRUE) |
| rootNode | ViewModelTreeNode | The 'root' ViewModelTreeNode used to access any 'child' nodes in the next level down in the hierarchy. |

### Returns

TreeLoadInput Newly created wrapper initialized with properties from the given inputs.

## createPropertyLoadInput(owningNodes: *Array.ViewModelTreeNode*, propertyNames: *Array*) → *PropertyLoadInput*

### Parameters

| Name | Type | Description |
|---|---|---|
| owningNodes | Array.ViewModelTreeNode | Array of nodes to load ViewModelProperty objects for. |
| propertyNames | array | Array property names to request. |

### Returns

PropertyLoadInput Newly created wrapper initialized with properties from the given inputs.

## getTreeTableState(declViewModel: *DeclarativeViewModel*, gridId: *string*) → *object*

### Parameters

| Name | Type | Description |
|---|---|---|
| declViewModel | DeclarativeViewModel | The DeclViewModel where the table is defined. |
| gridId | string | The ID of the declGrid used to define the table. |

### Returns

object An object holding the table state for the given declModel/declGrid.

### clearAllStates(declViewModel: *DeclarativeViewModel*, gridId: *string*)

#### Parameters

| Name | Type | Description |
|---|---|---|
| declViewModel | DeclarativeViewModel | The DeclViewModel where the table is defined. |
| gridId | string | The ID of the declGrid used to define the table. |

### isNodeExpanded(ttState: *object*, nodeToTest: *Array.ViewModelTreeNode*) → *boolean*

#### Parameters

| Name | Type | Description |
|---|---|---|
| ttState | object | An object holding the table state for a given declModel/ declGrid. |
| nodeToTest | Array.ViewModelTreeNode | The nodes to test status for. |

#### Returns

boolean TRUE if the given node is currently expanded.

### saveRowCollapsed(declViewModel: *DeclarativeViewModel*, gridId: *string*, targetNode: *ViewModelTreeNode*)

Clear the persisted expansion state of the given node . This is called when we collapse a tree node this clears the ExpansionState for a node

#### Parameters

| Name | Type | Description |
|---|---|---|
| declViewModel | DeclarativeViewModel | The DeclViewModel where the table is defined. |
| gridId | string | The ID of the declGrid used to define the table. |
| targetNode | ViewModelTreeNode | The node just collapsed. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

### saveRowExpanded(declViewModel: *DeclarativeViewModel*, gridId: *string*, targetNode: *ViewModelTreeNode*)

Update the persisted expansion state of the given node .

#### Parameters

| Name | Type | Description |
|---|---|---|
| declViewModel | DeclarativeViewModel | The DeclViewModel where the table is defined. |
| gridId | string | The ID of the declGrid used to define the table. |
| targetNode | ViewModelTreeNode | The node just expanded. |

### isViewModelTreeNode(objectToTest: *object*) → *boolean*

Test if the given object 'is-a' ViewModelTreeNode created by this service.

#### Parameters

| Name | Type | Description |
|---|---|---|
| objectToTest | object | Object to check prototype history of. |

#### Returns

boolean TRUE if the given object is a ViewModelTreeNode.

### createTableLoadResult(totalFound: *number*) → *TableLoadResult*

#### Parameters

| Name | Type | Description |
|---|---|---|
| totalFound | number | Total # of rows in the overall table. |

#### Returns

TableLoadResult Newly created wrapper initialized with properties from the given inputs.

### selectParent(dataProvider: *DataProvider*, selectedNode: *ViewModelTreeNode*)

Select the parent of the given node.

### Parameters

| Name | Type | Description |
| --- | --- | --- |
| dataProvider | DataProvider | The dataProvider instance of the tree. |
| selectedNode | ViewModelTreeNode | The viewModelTreeNode whose parent needs to be selected. |


Siemens Web Framework
Unpublished work. © 2025 Siemens

# 12. Glossary

| Term | Definition |
|---|---|
| Actions | **Actions** are payloads of information that send data from your application to your store/back-end. Siemens Web Framework let developers define Actions declaratively inside the view model and supports different type of actions (REST, JSFunction, JSFunctionAsync, Event or TcSoaService) through attribute `actionType`. |
| Adapters | While working on declarative UI, at times the object being displayed in DUI panel or the object on which command is being executed could be a proxy object. In such cases, the object needs to be adapted to a different object before performing any action on it. Declarative UI provides mechanism to adapt to a different object. For more information visit **adapters**. |
| Application context | The **Application context** is simply a singleton JavaScript object that can hold onto state and is made available for multiple view models to access in their conditions, actions inputData or outputData mapping. The appCtx will announce changes performed through its API using the EventBus changes: "appCtx.register" and "appCtx.update". |
| Audit | The **static analysis** command line tool for verifying and validating the declarative artifacts and source code for Siemens Web Framework and solutions built using Siemens Web Framework. |
| Authentication | **Authentication** is the process by which users securely access a server or remote computer by exchanging a Digital Certificate. |
| Bandwidth | Bit rate available for data transfer. |
| Batch action | **Batch Action** is group of actions executed in sequence. The batch actions executes in the exact sequence defined in the configuration and could be configured to pass data from one action to another. |
| Browser cache | Temporary storage location on your computer for files downloaded by browser to display a site. Files that are cached locally include any documents that make up a site, such as html files, CSS, JavaScript, as well as images and other multimedia content. |
| Bundle | **Bundling** is a unit of resources that have been amortized to be demand loaded at the same point in time. |
| Cache-control | HTTP header which is used to control how the resource should be cached. Used to control how the browser manages the resource. |
| Commands | The **commands** encapsulates actions as objects and provides a graphical means for the user to work with their product, process, or other data. |

| | |
|---|---|
| Command decorators | **Command decorators** are used to provide additional information in commands. |
| Command handlers | **Command handlers** handles the command logic. It is used to handle the visible/invisible and enable/disable of commands and the actions of command. |
| Command placement | A **placement** is the actual visual rendering of the display of the command and it has properties such as: *id*, *uiAnchor*, *priority* and *relativeTo*. |
| Common Frame | Common Application Frame consists of header and side navigation and it provides single web presence across all PL Web offerings. |
| Compression | Data compression techniques provide a way to convert data into alternate format which use less. For a site this reduces bandwidth demand for transmission of site content over the network. |
| Conditions | **Conditions** help to evaluate data at run time and can be used with *visible-when* clause and is also used for event handling. |
| Custom actions | Through **custom actions** developers can write their new actions on the top of basic Siemens Web Framework's actions or their own pre-defined custom actions. |
| Data Parser | **Data parser** maps raw JSON output from action response to view model property objects and it contains properties such as *id* and *responseObjs*. |
| Data Provider | **Data Provider**, is a single placeholder wherein we configure different ways to fetch/paginate data and pass the same data to underlying component. It is basically an abstraction layer for components which fulfils component's demand to load/paginate data from the server. |
| Declarative Chrome Extension | It is a tool to debug declarative UI. This enables you to view the declarative view model and application context live in the browser during debugging. |
| Declarative UI | **Declarative UI** is a capability provided by the Siemens Web Framework that allows for a concise, codeless and prescriptive definition of the UI view content, its layout, routing and behaviours. Actions, messages, events, service calls and their inputs/outputs can be mapped and described. It provides an abstracted means of declaratively defining a panel and its behaviours. How it is implemented is hidden/abstracted. |
| Events | **Events** are means communication between view models or UI elements. |
| Extended Tooltip | **Extended tooltip** contains descriptive messages and images that are displayed near the cursor when it hovers over a toolbar or ribbon panel button or any UI element from the screen. |


Unpublished work. © 2025 Siemens

| | | |
|---|---|---|
| Functions | **Function** is a construct of view model consisting of *functionName*, *parameters* and *deps*. Functions can be used in *inputData*, *outputData* and *dataParseDefinition*. | |
| Group commands | **Group commands** are used for displaying multiple varients of commands. | |
| i18n | User facing text is localized. The **localization** resources that provide the translations for the user facing text are JSON format. | |
| Kernel | Kernel is low level internals of the Siemens Web Framework framework. Services for logging, event bus, icons, local storage, paste, etc. | |
| Latency | The delay in data communication over a network. Over a WAN this can be in the order of ~300ms between continents. | |
| Layout | A **layout** defines the structure for a user interface in your application. Layout elemnts consists of *aw-row*, *aw-column*, *aw-break*, *aw-splitter* and *global navigation*. | |
| Max-age | Field in the Cache-control which controls how long a resource can be cached. Used to control how the browser manages the resource. | |
| Messages | **Messages** are declaratively defined in the view model and are launched by events and actions. It can be used for cover information, warning and error notifications. | |
| Minification | Process of removing unnecessary characters from source code (i.e. Javascript) without changing its functionality. Typically this is the removal of white space as a means of reducing the size of resources for transfer over the network. | |
| Module | **Modules** are highly self-contained with distinct functionality, allowing them to be shuffled, removed, or added as necessary, without disrupting the system as a whole. | |
| Page | Pages describe one screen of the user experience. Pages have a URL and are the entire presentation in the browser window. Pages follow **layout** and content patterns and are declaratively defined. They can be made up from several panels. | |
| Panel | Panels describe a region of the page and are declaratively defined. | |
| Routing | When navigating through pages/subpages of a solution and changing URL, the process of associating URL fragments to a given page or subpage content that is demand loaded given its state as part of a solution's navigation as provided by the Siemens Web Framework. | For more information visit **routing**. |
| Schema | The declarative view model is JSON format and adheres to its schema and Siemens Web Framework audit checking validates view models against that schema. The view model has a mandatory "schemaVersion" field that specifies the version of the schema it adheres to. | |

| | |
|---|---|
| Storybook | The **Siemens Web Framework Storybook** helps you to create an innovative app interface and provides the guidelines for developers, partners, and customers. It describes how Siemens Web Framework works and thinks and what we are trying to accomplish. Prior to Siemens Web Framework 7.0, a custom Showcase application was also available. |
| Site | **Site** is the name given to the deployment package for the solution which is deployed to an app server. |
| Solution | A **solution** is a product that has been built on Siemens Web Framework and it is made up from a set of applicable install Packages. |
| Solution generator | It is minimal 'declarative' code that creates basic working structure for a solution. |
| States | **State** definitions specify the URL routing for the set of pages and subpages that form the solution. This controls how a given URL fragment is associated to a given page or subpage's view and view model definition. In Siemens Web Framework, each page and subpage can be defined a unique state. These states get stored in a states.json file on a per module basis. Each states.json file lives as a peer to the module.json. When constructing a "site" the Siemens Web Framework tooling pulls together the state definitions. |
| Tools | Siemens Web Framework provides **tools** for building your solution to create a deployment package (known as a *site*). Commands are provided to deploy the *site* to a live-server file server for development iteration. |
| UI element | A **UI element** is an HTML5 custom element that forms part of the vocabulary for the view. UI elements is how we create declarative building blocks and how we teach HTML new tricks. Custom Elements is one part of the W3C Web Components. In Declarative UI the UI elements do not expose CSS classes, divs or details such as which web framework was used for its internal implementation. |
| Validation criteria | **Validation criteria** is a declarative way to validate property value on the client side. It helps ensure user fills out each form field with correct format and value. |
| View | **View** is simplified HTML markup. It is responsible for specifying view hierarchy of a page or panel. It is populated with UI elements to create the desired UI and layout. |
| View model | **View model** is JSON text. It is responsible for view state for its corresponding view. It declaratively describes data, actions, conditions, events, messages, localizations and imports for the UI elements used in its corresponding view. |
| Workspace | **Workspaces** configure some UI aspects of the application. Within the same solution we can partition capabilities in order to tailor for given |


Unpublished work. © 2025 Siemens

| | |
|---|---|
| | categories of user. Solution provider determines which Workspaces are provided and which is default. |


Unpublished work. © 2025 Siemens