

webMethods CAF and OpenUI Development Help

Version 10.15

October 2022

This document applies to webMethods Composite Application Framework 10.15 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2007-2022 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <https://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

Document ID: CAF-OLH-1015-20221015

Table of Contents

About this Guide.....	9
Document Conventions.....	10
Online Information and Support.....	10
Data Protection.....	11
 1 CAF and webMethods OpenUI Development Prerequisites and Resources.....	 13
Technologies Required for CAF and webMethods OpenUI Development.....	14
Specifying an External Web Browser.....	14
 2 Composite Application Framework Concepts.....	 15
CAF and webMethods OpenUI Development Requirements.....	16
About JavaServer Faces.....	16
About Ajax.....	17
About Portlet Applications.....	17
About Portlets.....	17
About Views.....	18
About CAF Controls.....	18
About Managed Beans.....	19
About CAF File Names and Path Lengths.....	20
 3 Developing CAF Applications.....	 21
Setting CAF Development Preferences.....	22
About Component Libraries.....	23
About Web Application Project Templates.....	24
Cascading Style Sheets (CSS).....	25
Importing a View into a VDL File.....	27
Working with Page Templates.....	27
Navigation in Web Applications.....	29
Setting Application Initialization Parameters.....	33
Setting Environment Variables.....	34
Finding the Home Page ID.....	35
Managing Text Overflow in Property Lines.....	35
Placing Modal Dialog and Popup Panel Controls.....	38
Importing CAF Projects.....	40
Validation Support for xmlImport Files.....	40
Specifying a Target Runtime Server.....	41
Debugging CAF Applications Remotely.....	42
 4 Developing webMethods OpenUI Applications.....	 43
About webMethods OpenUI.....	44
About webMethods OpenUI Shell.....	44
About JSF 2.x Facelets Support.....	45
Additional Resources for Developing OpenUI Applications.....	45

Security in OpenUI Applications.....	46
5 Security in CAF Applications.....	47
Security Roles in Web and Portlet Applications.....	48
Mitigating Portlet Security Vulnerabilities.....	50
6 Getting Started with Web Application Development.....	53
About Web Applications.....	54
Creating a Web Application Project.....	54
Adding a View to a Web Application.....	55
Adding a Control to a Web Application View.....	57
Preparing a Web Application for Localization.....	57
7 Getting Started with Portlet Application Development.....	61
About Portlet Application Projects.....	62
Creating a Portlet Application Project.....	62
Creating a Portlet.....	63
About Portlet Types.....	63
About Portlet Modes.....	64
Adding a View to a Portlet.....	65
Adding a Control to a Portlet View.....	66
Finding Information about CAF Controls.....	67
Adding a Portlet Application Project to My webMethods Server.....	68
Importing My webMethods Server Assets into a Portlet Application.....	68
Publishing a Portlet Application to My webMethods Server.....	69
Adding a Portlet to a My webMethods Server Page.....	69
Uninstalling Portlet Applications from My webMethods Server.....	70
Troubleshooting Portlet Applications.....	70
Repairing a Portlet Application Project.....	71
Opening the Portlet Application Configuration Editor.....	71
8 Getting Started with webMethods OpenUI Shell Page Development.....	73
About webMethods OpenUI Shell Page Templates.....	74
Creating an OpenUI Shell Page Project.....	74
Modifying an OpenUI Shell Custom Application Page.....	75
Modifying an OpenUI Shell Login Page.....	77
Modifying an OpenUI Shell Maintenance Page.....	78
Adding an OpenUI Shell Page to an Existing Project.....	80
About Creating Custom Templates with webMethods OpenUI.....	80
Creating a Custom OpenUI Page Template.....	80
Creating a Custom OpenUI Project Template.....	82
Editing a Custom Page or Project Template.....	83
Exporting a Custom Page or Project Template.....	83
Importing a Custom Page or Project Template in Designer.....	84
Deleting a Custom Page or Project Template.....	84
Default webMethods OpenUI Page Templates.....	84
Variable Conversion When Applying a Page Template.....	85
Handling of Existing Files When Applying a Page Template.....	87

9 Working with Facelets.....	89
About Facelets in CAF.....	90
Migrating CAF Projects and .view Files to JSF Facelets Format.....	90
Working with the CAF Control Tag Library.....	93
Working with Page Navigation in JSF Facelets Applications.....	93
Embedding Portal Resources in a Page File.....	94
Example of Inserting a Form Into a Portlet.....	95
Java Annotations in CAF and webMethods OpenUI Applications.....	97
 10 Working with Portlet Preferences.....	 109
About Portlet Preferences.....	110
Creating a Portlet Preference.....	110
Modifying a Portlet Preference.....	111
Specifying a Portlet Preference Validator.....	112
Wiring Portlets with Preferences.....	112
Exposing Portlet Preferences in My webMethods Server.....	113
Enabling the Show All Managed Beans Toolbar Button.....	114
About Preferences for the Default Portlet Types.....	115
 11 Working with Legacy Portlets.....	 117
About Legacy Portlets.....	118
Enabling Legacy Portlet Support.....	118
Importing Legacy Portlets.....	119
Changing Legacy Portlet Preferences.....	119
 12 Working with CAF Events and Notifications.....	 121
About CAF Events.....	122
Opening the CAF Events Editor and Enabling CAF Events.....	123
Creating a Custom CAF Event.....	124
Creating an Event Handler.....	125
Creating a Notification.....	131
Creating a Subscription.....	133
Configuring Simple Conditions.....	135
Configuring Advanced Condition and Action Expressions.....	138
 13 E-form Support in Portlet Applications.....	 141
Adding E-form Support to a CAF Portlet Application.....	142
Configuring a CAF Portlet Application Project for E-form Support.....	142
Adding an IS Document Type for E-form Support.....	142
Adding an E-form Template Type to a CAF Portlet View.....	143
Adding an E-form Template from a File System or Web Server.....	143
Adding an E-form Template from a Repository.....	145
Adding a Basic Default E-form Download or Upload User Interface.....	145
Adding an Advanced Default E-Form Download or Upload User Interface.....	146
 14 Using a Simple Email Deliverer.....	 147

About the CAF Simple Email Deliverer.....	148
Adding a Simple Email Deliverer.....	148
Specifying the Mail Session.....	149
Specifying Data Binding for a Simple Email Deliverer.....	149
15 Working with the Bindings View.....	151
About the Bindings View.....	152
Binding a Managed Bean Property to Another Bean.....	153
Binding Data to a Control.....	154
Binding a User Attribute to an Output Text Control.....	154
Binding a Security Role to a Control.....	155
Binding Security Roles in an Access Control Panel.....	155
Creating a New Action Method.....	156
16 Working with My webMethods Server.....	159
Adding a My webMethods Server Instance in Designer.....	160
Editing a My webMethods Server Instance.....	161
Starting a My webMethods Server Instance.....	161
Stopping a My webMethods Server Instance.....	161
Running My webMethods Server in Debug Mode.....	162
Specifying a Default Application Server.....	162
Publishing an Application to My webMethods Server.....	162
Manually Deploying a Web Application to My webMethods Server.....	163
Deploying a Jar File to My webMethods Server.....	164
Exporting from My webMethods Server.....	164
17 My webMethods Server Runtime Assets.....	167
About Runtime Asset Extraction.....	168
Extracting Runtime Assets.....	172
Creating a Deployable Component.....	173
18 Performance Monitoring.....	175
Measuring Application Performance.....	176
Creating Custom Tokens for Performance Monitoring.....	177
19 Using Attachments Providers.....	179
About Attachments Providers.....	180
Adding an Attachments Provider Property.....	180
Adding a Shared Managed Bean and Attachments Provider Reference.....	181
20 Working with JCR Providers.....	183
About JCR Client Providers.....	184
Enabling the JCR Client for a CAF Application.....	184
Creating a JCR Client Attachments Provider.....	185
Creating a Node Children Table Provider.....	185
Adding Additional Properties to the JCR Provider.....	186
Creating a JCR Client Node Provider.....	187

Creating a Search Result Table Provider.....	188
Creating a JCR Sub-Folder Attachments Provider.....	188
Creating a JCR Client Temp Attachments Provider.....	190
21 Working with Web Services.....	193
About Web Services in CAF Applications.....	194
Connecting to webMethods Integration Server.....	194
Specifying a Web Service Connector.....	195
Setting Web Service Connector Preferences.....	199
Specifying a Web Service Connector Socket Timeout.....	200
Migrating a Web Service from Glue to WS-Stack.....	200
Configuring wsclient-socketTimeout at Run Time.....	202
Configuring Global Runtime Timeout.....	203
Web Service Connector Preferences.....	203
About XSD Schema Choice Declarations.....	204
22 Using the MWS Admin View.....	207
About the MWS Admin View.....	208
Creating a MWS Data Provider.....	209
Changing the Connection Properties for a Server Instance.....	210
Opening the MWS Administration Dashboard.....	210
23 Views in the Composite Application Framework.....	211
The UI Development Perspective.....	213
The Bindings View.....	215
Data Source Explorer View.....	215
Connecting to a Database.....	215
Creating a Database Connector.....	216
Outline View.....	217
Package Navigator View.....	218
Properties View.....	218
Properties View Toolbar.....	218
Snippets View.....	219
Customizing the Snippets View.....	219
E-forms Snippets Drawer.....	221
CAF Dialog Patterns.....	221
Solutions View.....	221
Database Connectors in the Solutions View.....	221
Creating a Portlet on the Solutions View.....	222
Using the Design Canvas.....	223
The Palette View.....	227
CAF Events Editor.....	230
Using Snippets.....	236
Customizing UI Controls.....	241
24 User Interface Controls Concepts.....	243
Control ID Reference.....	244
Hideable Controls.....	246

Toggle Controls.....	248
Scriptaculous Effects.....	249
Image URLs.....	253
Skinning.....	253
Client-Side Libraries.....	254
25 Understanding the Client-side Model.....	263
About the Client-Side Model.....	264
CAF.Model.....	264
CAF.Output.Model.....	264
CAF.Link.Model.....	265
CAF.Command.Model.....	265
CAF.Input.Model.....	266
CAF.Checkbox.Model.....	267
CAF.Select.Model.....	267
CAF.Table.Model.....	269
CAF.Tree.Model.....	271
Script Placement in the CAF Model.....	272
26 Using Converters and Validators.....	275
About Using Converters and Validators with CAF Controls.....	276
Displaying Converters and Validators.....	276
Adding a Converter to a Control.....	276
Adding a Validator to a Control.....	277
Creating a Custom Converter.....	277
Creating a Custom Validator.....	278

About this Guide

- Document Conventions 10
- Online Information and Support 10
- Data Protection 11

This guide provides information about creating user interfaces with Composite Application Framework (CAF) and webMethods OpenUI.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the symbol.
[]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at <https://documentation.softwareag.com>.

In addition, you can also access the cloud product documentation via <https://www.softwareag.cloud>. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at <https://knowledge.softwareag.com>.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at <https://techcommunity.softwareag.com>. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at <https://github.com/softwareag> and <https://hub.docker.com/publishers/softwareag> and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at <https://empower.softwareag.com>. Many services on this portal require that you have an account. If you do not yet have one, you can request it at <https://empower.softwareag.com/register>. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 CAF and webMethods OpenUI Development

Prerequisites and Resources

■ Technologies Required for CAF and webMethods OpenUI Development	14
■ Specifying an External Web Browser	14

Technologies Required for CAF and webMethods OpenUI Development

Composite Application Framework (CAF) supports the standards of the OpenAjax Alliance. For more information, see [OpenAjax Alliance](#).

The following technologies and version levels are required for the development and run-time environments for Composite Application Framework and webMethods OpenUI.

- Java 1.8
- Servlet 3.1
- JSF 2.2
- Portlet 1.0
- JCR 2.0
- Jetty 9.2

Specifying an External Web Browser

Composite Application Framework uses an internal web browser to display a preview of the application and its user interface (views) when you run it on My webMethods Server. The internal browser is part of the Eclipse framework on which Composite Application Framework was built. However, you can use an external web browser to preview your web and portlet views by specifying an external web browser in **Preferences**.

➤ To specify the use of an external web browser

1. In Software AG Designer, select **Window > Preferences**.
2. In the Preferences window, expand **General**, and click **Web Browser**.
3. In the Web Browser panel, select **Use external web browser**.
4. From the **External Web Browsers** list, select a browser to use, click **Apply**, and then click **OK**.

2 Composite Application Framework Concepts

■ CAF and webMethods OpenUI Development Requirements	16
■ About JavaServer Faces	16
■ About Ajax	17
■ About Portlet Applications	17
■ About Portlets	17
■ About Views	18
■ About CAF Controls	18
■ About Managed Beans	19
■ About CAF File Names and Path Lengths	20

CAF and webMethods OpenUI Development Requirements

Before you develop Composite Application Framework (CAF) and webMethods OpenUI solutions, you are advised to have a working understanding of the following:

- The Eclipse development environment, including the use of perspectives, views, and editors.
- Web development, including development of web and portlet applications, web services, and design requirements for web application client-side user interface development, portlets, and Ajax.
- The Java Platform, Enterprise Edition, formerly known as Java 2 Platform, Enterprise Edition/J2EE.
- JavaServer Faces (JSF) technology and XHTML.
- Concepts of service-oriented architecture (SOA).

In addition, you should have a good understanding of My webMethods Server and the resources available through it. For more information about My webMethods Server, see *Administering My webMethods Server*.

You should also be familiar with the webMethods Product Suite and its integrated approach for developing, deploying, and running web-based applications.

For specific information related to the Java classes used in CAF web and portlet applications, see *webMethods CAF and My webMethods Server Java API Reference*.

About JavaServer Faces

Portlets created in the Composite Application Framework (CAF) support the JavaServer Faces (JSF) standard. You can create new projects as JSF applications with portlets that use View Declaration Language (VDL) pages implemented either as .view files (deprecated), which is a file structure with a proprietary Software AG XML schema, or as JSF Facelets .xhtml files (default).

By default, Software AG Designer creates portlet applications and new pages using the .xhtml file format. To create legacy view pages, you can specify a .view file name extension when creating a view page.

Facelets support provides you with the ability to use:

- XHTML to create web pages.
- Facelets tag libraries to augment JavaServer Faces and JSTL tag libraries.
- The Java Unified Expression Language (UEL).
- Templates for components and pages.

In addition, you can migrate CAF applications developed with .view files to JSF Facelets applications. For more information about migrating CAF projects and .view files to .xhtml files, see [“Migrating CAF Projects and .view Files to JSF Facelets Format” on page 90](#).

For more information about the Facelets technology, see the Oracle Java documentation website.

About Ajax

Composite Application Framework (CAF) uses asynchronous JavaScript and XML (Ajax) for creating interactive web applications. In the Ajax model, an Ajax engine acts as an intermediary between the user and the server. When the user interacts with a web form, the web page communicates asynchronously with the server, enabling the user to make multiple changes in a web form without the need for a page reload. This independent communication behavior provides a rapid response to the user's input into the web application.

Many of the controls available on the Palette view of the UI Development perspective use Ajax techniques, making it possible for you to incorporate Ajax into your portlet applications. Controls with Ajax capabilities are identified as such in the Controls Reference.

About Portlet Applications

A portlet application project is a container for managing one or more portlets. If you want to create a portlet, you must first create a portlet application project to contain it.

After you create a portlet, you can then create the portlet views that you want to implement within the portlet. Portlet applications are deployable only to My webMethods Server.

A portlet can also include a dynamic web application in a portlet view. You can review the user interface of the portlet using the **Preview** tab of the editor to view a static HTML version of the view.

Concerning JSF support:

- Prior to CAF version 9.6, portlet applications supported JSF by implementing views using View Declaration Language (VDL) files created as .view files, a file structure with a proprietary Software AG XML schema.
- With CAF version 9.6 and later, .view files are still supported, but new portlet applications are created as .xhtml files with default JSF Facelets support, enabling the use of Facelets functionality. You can create legacy view pages by specifying a .view file name extension when creating a view page.
- CAF applications developed with .view files in earlier versions of Composite Application Framework can be migrated into JSF Facelets applications at the project level. Individual .view files can also be migrated to .xhtml files. For more information about migrating CAF projects and .view files to .xhtml files, see “[Migrating CAF Projects and .view Files to JSF Facelets Format](#)” on page 90.

About Portlets

A portlet is a server-side component that can be reused in multiple portlet applications. Typically, a portlet presents or provides access to various web resources, data, external applications, and other components. In the Composite Application Framework (CAF) environment, portlets are published to and run on My webMethods Server

Portlets created in CAF are based on the JSR-168 portlet standard. Portlet configuration data is stored as preferences, information that is persisted within a portlet for each user.

CAF adds extensions to standard preferences with simple data types, such as Boolean and integer, as well as additional scopes. These extended scopes are valid only on My webMethods Server. Because of these extensions, you should analyze your portlets carefully if you are considering exporting a portlet to a third-party portal.

When you create a Generic portlet with the New Portlet wizard, the portlet is created with a single default View Declaration Language (VDL) file.

About Views

Within each portlet or web application you create, you can create one or more *views* that define the user interface for the portlet. You define the views as View Declaration Language (VDL) files. Composite Application Framework (CAF) supports two types of VDL files:

- A VDL file implemented as a .view file structured with a proprietary Software AG XML schema. By default, Software AG Designer creates new view pages using the .view format (deprecated). You can create legacy view pages by specifying a .view file name extension when creating a view page.
- VDL files in .xhtml format for JSF Facelets projects (default).

A view represents a page in an application, defining the structure of the controls for the user interface. Typically, CAF has associated a single page managed bean as the primary server-side backing model for every page. However, this is not required. A VDL file can reference any managed bean, and it may be easier for some use cases to not use a page managed bean at all.

You can open a VDL file in the CAF editor, enabling you to modify the view to your custom requirements. You can:

- Drag CAF controls from the Palette view directly on to the design canvas.
- Specify properties for controls, for example, isVisible, isRendered, isDisabled.
- Bind controls to managed beans.
- Import portal resources into your view.

About CAF Controls

To develop the user interface for an application, you add CAF controls to View Declaration Language (VDL) files. Composite Application Framework (CAF) provides two sets of its CAF controls:

- Controls for JSF applications implemented with .view files
- Controls for JSF Facelets applications in .xhtml files

Both sets of CAF controls are available in the Palette view, and are essentially identical, providing the same behavior and control attributes in each case. When you add a control to a VDL file, it is

added using the appropriate code for the container file, either .view or .xhtml. In both cases, you can view and modify the control attributes through the Properties view, or directly in the code if you prefer.

You can find descriptions of the CAF controls and their attributes in *webMethods CAF Tag Library Reference*. For more information, see [“Finding Information about CAF Controls” on page 67](#).

About Managed Beans

A *managed bean* is a Java class that represents a portlet resource, including a description of its attributes, operations, notifications, its unique name within a portlet application, a bean type, and a scope. The scope determines how long the bean exists within the application.

Managed beans are found in the Bindings view. By default, the Bindings view displays only managed beans associated with the page bean currently displayed on the design canvas. You can optionally display all managed beans in the Bindings view. For more information, see [“Enabling the Show All Managed Beans Toolbar Button” on page 114](#).

The following table lists the managed bean scope values and their definitions.

Scope	Description
Application	Facilitates portlet-to-portlet communication. The managed bean is started once and does not expire until the server shuts down or the portlet application is re-published to the server.
Session	Default. Limits the life of the managed bean to the user session. Portlets using this scope cannot share data with other portlets. The managed bean expires when the user session ends.
Request	Expires when the response to the request is sent back to the client. The managed bean life span is that of a single request.
None	Creates the bean every time it is requested.

Setting the Managed Bean Expiration Policy

If you create custom pages in the Fabric Tasks area of My webMethods Server, it is possible to specify the managed bean expiration policy for a Fabric folder (and for workspace templates as well) in My webMethods Server. By default, a page’s managed bean is destroyed when the user navigates to another page. When the user returns to the original page, it must be completely reloaded.

To avoid this, you can use the Bean Expire Policy property for Fabric folders and workspace templates to specify that the managed bean does not expire. For more information, see *Administering My webMethods Server*.

About CAF File Names and Path Lengths

As you develop applications and pages in Composite Application Framework (CAF), be aware of the following file name and path limitations:

Special Characters in File Names

In CAF web and portlet applications, names cannot contain spaces or any of the following characters:

- asterisk (*)
- Pipe (|)
- Back slash (\)
- Forward slash (/)
- Colon (:)
- Double quotes (")
- Less than (<)
- Greater than (>)
- Period (.)
- Question mark (?)

File Name and Path Lengths

Although file names can use up to 255 characters, you should create file names that are as brief as possible to ensure that CAF paths are not too long to parse during runtime. Paths that are too long are a problem for applications deployed in a Windows environment.

For Java classes inside a CAF application, the limit is less than 255 characters because you must subtract the length of the path to where the CAF web archive (WAR) file is first unpacked when it is deployed. The following example shows the possible path length when a CAF application is deployed in a Windows environment:

```
C:\SoftwareAG\MWS\server\default\deploy\wm_yourappname\WEB-INF\classes
```

3 Developing CAF Applications

■ Setting CAF Development Preferences	22
■ About Component Libraries	23
■ About Web Application Project Templates	24
■ Cascading Style Sheets (CSS)	25
■ Importing a View into a VDL File	27
■ Working with Page Templates	27
■ Navigation in Web Applications	29
■ Setting Application Initialization Parameters	33
■ Setting Environment Variables	34
■ Finding the Home Page ID	35
■ Managing Text Overflow in Property Lines	35
■ Placing Modal Dialog and Popup Panel Controls	38
■ Importing CAF Projects	40
■ Validation Support for xmlImport Files	40
■ Specifying a Target Runtime Server	41
■ Debugging CAF Applications Remotely	42

Setting CAF Development Preferences

You can use Composite Application Framework (CAF) development preferences to customize the behavior of your development environment.

➤ To set CAF development preferences

1. In Designer: **Window > Preferences**.
2. In the Preferences dialog box, expand the **Software AG** node, and then expand the **UI Development** node.
3. Click the **UI Development** node or one of its child nodes to set preferences. For more information about the available preferences, see [“About CAF Development Preferences” on page 22](#).
4. Click **Apply** to apply your changes and remain in the Preferences dialog box, or click **OK** to apply your changes and exit the Preferences dialog box.

About CAF Development Preferences

The following table list the Composite Application Framework development preferences that you can set for the UI Development perspective:

Preference Node	Preference	Description
UI Development	Default Java Package	Not specified by default.
	Use Ajax Controls	Selected by default.
	Prefer Declaring Managed Beans via Java Annotations	Selected by default. Select this option to declare new managed beans using Java annotations. Do not select this option to declare them in the WEB-INF/faces-config.xml file.
UI Development > Bindings View	Display All Managed Beans toolbar button	Not selected by default. The Bindings view displays only managed beans associated with the page bean currently displayed on the design canvas. Select this option to display all managed beans in the Bindings view.
UI Development > View Editor	Delete components without confirmation	Not selected by default.

Preference Node	Preference	Description
UI Development > Web Service Connector	Silently create Web Service Connector on Drag-and-Drop	Selected by default.
	Show warning for creating a Web Service Connector from a deprecated source	Selected by default.
	Use wsdl schema namespace values for generating java package names	Not selected by default.
	Automatically initialize objects for optional input fields	Selected by default.
	Default SOAP library	WS-Stack Client is selected by default.

About Component Libraries

The Composite Application Framework (CAF) component libraries contain controls that are displayed in the Palette view of the UI Development perspective. You can also associate other component libraries that do not appear by default in the Palette view.

When you create a web or portlet application project, CAF includes the CAF Base Controls, CAF Base Runtime, and CAF Java Logging component libraries. When creating a new web or portlet application project, you can add other component libraries such as CAF JDBC Drivers, CAF JCR Client and CAF Portlet Controls. For more information, see [“Views in the Composite Application Framework” on page 211](#).

You can also make these component libraries available on the server by including them in the web archive (WAR) file for the application.

Removing a Component Library from an Application

Note:

If you remove a component library from a Composite Application Framework application, and a control from that library is used in the web or portlet application, an error might occur.

➤ To remove a component library from an application

1. In the UI Development perspective, click the Project Explorer view, right-click the node of the application project, and then click **Properties**.
2. In the Properties dialog box, click **Java Build Path**, and then click the **Libraries** tab.

If a component library is listed, that library is associated with the application.

3. Select one or more component libraries to remove from the application, click **Remove**, and click **OK**.
4. Save the project.

Manually Adding a Component Library to an Application

Composite Application Framework automatically adds component libraries to a web and portlet application projects. For example, if you add a web service client to a view, Composite Application Framework adds a web service library to the application. If a component library has not been added previously, you can add it manually.

> To manually add a component library to an application

1. In the UI Development perspective, click the Project Explorer view, right-click the node of the application project, and then click **Properties**.
2. In the Properties dialog box, click **Java Build Path**, click the **Libraries** tab, and then click **Add Library**.
3. In **Add Library**, select a library, click **Next**, and then click **Finish**.
4. Save the project.

About Web Application Project Templates

You specify a project template at the time you create a new web application project.

A project template is a web application that provides you with a predefined set of components from which you can develop a full application. While a project template is not intended as a sample, it can give you an idea of how a web application goes together. When you create a web application, you can choose to begin with an empty project template, or with a Starter Web Application project template that contains the following:

■ default view

Represents the home page for the web application. This page imports the header view file

■ header view

Represents the view that contains navigation and other elements that should appear on each page in a web application. This view is imported into the default view and into any new views created for the web application.

■ login_error view

Represents the page users are redirected to if a login fails authentication.

■ login view

Represents the login page users are redirected to when attempting to open the default view file.

The Starter Web Application project template also includes a `styles.css` file. For more information about CSS files, see [“Cascading Style Sheets \(CSS\)” on page 25](#).

Cascading Style Sheets (CSS)

When you create a web or portlet application, you must also define the appearance of the page. You can specify the appearance of individual controls and views using a Cascading Style Sheet (CSS). In the Composite Application Framework, you can define CSS styles for individual controls in the UI Development perspective using the Properties view, in the CSS Style Definition editor, or by attaching an external CSS to use in your project.

Apply CSS style values to an individual controls using the Properties view in the Composite Application Framework UI Development perspective. CSS styles defined for individual controls have precedence over a CSS applied to the entire page.

If you are not experienced in defining CSS styles, you can use the CSS Style Definition editor to define how a control looks in the view. However, it is a good idea to learn as much about CSS properties and attributes as possible before you begin to define styles for individual controls.

In the CSS Style Definition editor, you can define HTML style properties and attributes such as font family, font size, style, case, weight, and other attributes. The CSS style definitions are not limited to fonts, you can also define attributes for the text block, border, background, and list style.

You can also attach an external CSS with the Include Stylesheet control. For more information about the Include Stylesheet control, see *webMethods CAF Tag Library Reference*. For more information about how to use *webMethods CAF Tag Library Reference*, see [“Finding Information about CAF Controls” on page 67](#).

Changing the CSS Values for a Control



You can edit CSS style values for controls in a web application or portlet application project using the Composite Application Framework UI Development perspective. CSS property values and attributes applied to individual controls have precedence over a CSS applied to the entire page. For more information, see [“Applying CSS to a View” on page 26](#).

➤ To edit the CSS values for a control

1. On the UI Development perspective, open the View Declaration Language (VDL) file for the view you want to work with in the editor.
2. Select the control you want to modify.

Tip:

For the following steps, click the  in the Properties view tool bar to view the **Display** tab.

3. In the Properties view, click the **Display** tab.
4. For the **CSS Style** property, do any of the following:
 - Place the cursor inside the **CSS Style** text field and start typing your CSS definition.
 - Click  to open the CSS editor. In the **Edit CSS** dialog box, define the CSS property and attribute values to apply to the control, or edit any existing text. Click **OK**.
 - Click the expression binding browse button  if you want to add a binding expression to your style definition.

Important:

Doing so will overwrite all contents in the **CSS Style** field. A selected binding expression is added to **CSS Style** field and can be further edited in the CSS editor.

5. Click **Preview** to view the run-time appearance of your CSS Style property change.
6. Save the project.

Applying CSS to a View

You can specify an external style sheet for use by all controls in a view. If you apply the style sheet to a header, or to another view that is imported into all pages of a web application, you can provide a common style to all pages in the application. For more information, see [“Importing a View into a VDL File” on page 27](#).

To view an example, create a web project with the Starter Web Application project template. By default, the login.xhtml includes the external style sheet styles.css.

You can locate an external CSS at a URL to make it accessible by the application server at runtime, or you can include the CSS in the web application project.

➤ To apply an external style sheet to a view

1. In the UI Development perspective, browse to the location of the external CSS you want to add to the project. Right-click the file and click **Copy**.
2. Click the Navigator view, then expand the project's **WebContent** node.
3. Right-click the **WebContent** node and click **Paste**.
4. Open the VDL file for the view you want to work with in the editor.
5. In the **WebContent** node, drag the CSS to the design canvas and drop it into the VDL file (typically at the top).

6. In the Drop Candidates dialog, click **Create “Include Stylesheet” Control** and click **OK**.

The Composite Application Framework automatically adds an Include Stylesheet control to the view and binds it to the CSS file.

7. Click the **Preview** tab to check the result of adding the external CSS to the view.
8. Save the project.

Importing a View into a VDL File

The Import Template View control enables you to import one view into another view. You can use the imported view as a template to create a reusable component, such as a header, that can be used on all pages in an application.

- **For a web application:** you can see a typical implementation example in the Starter Web Application project template that you can create as a new web application project. By default, the header view is imported into the default view.
- **For a portlet application:** you can use the Import Template View control as described below, but you can also use JSF Facelets functionality to incorporate an XHTML-based view as a template. For more information, see “[Embedding Portal Resources in a Page File](#)” on page 94.

➤ To import a view with an Import Template View control

1. If it does not already exist, create the template view you want to import.
2. In the Project Explorer view, expand the **WebContent** node of the project you want to import the view to, and then double-click the View Declaration Language (VDL) file for the target view to open it in the editor.
3. Drag the view you want to import from the Project Explorer view and drop it into the open VDL file in the editor.
4. In the Drop Candidates dialog box, click **Create “Import Template View” Control** and then click **OK**.
5. Save the project.

Working with Page Templates

You can use the Import Template View control to import a template view into your project. You can also use CAF controls to inject content into the imported view. For more information about how to import a view with the Import Template View control, see “[Importing a View into a VDL File](#)” on page 27.

Note:

For a portlet application, you might want to consider using JSF Facelets functionality to incorporate an XHTML-based view as a template. For more information, see “[About Facelets in CAF](#)” on page 90 and “[Embedding Portal Resources in a Page File](#)” on page 94

You can accomplish this content injection through the use of a marker control. A marker control is any control (often, a Panel control) that is used to mark a section that is to be replaced by other content. You specify the content to inject by including a Content Parameter control as a child of the Import Template View control.

For example, the following conceptual diagram shows how a template view (View B) contains markers for the page title, the advanced-search property group, and the search-results table:

Search Page Template (View B)

The diagram illustrates the structure of the Search Page Template (View B). It consists of several sections: a top search bar with a 'Search for' label and a text input containing 'marker: page-title'; a 'Basic Search' section with a 'Keywords' input; an 'Advanced Search' section with a 'marker: advanced-search' input; a 'Search' button; and a 'Search Results' section with a 'marker: search-results' input.

The top-level instance view (View A) imports the template view, and specifies the content for the page title, advanced-search, and search-results sections:

Search Page Results (View A)

The diagram illustrates the structure of the Search Page Results (View A). It shows an 'import: view-b' section with three parameters: 'param: page-title' (Customers), 'param: advanced-search' (Country: [input], Active within last: [Year]), and 'param: search-results' (a table with columns ID, Name, Country, Last Order Date and rows of customer data).

The name of the Content Parameter control specified as a child of the Import Template View control must be the ID or Control ID value of the control in the imported view that the Content Parameter's content will replace. For example:

```
&=Control Parameter (name=mySpecialControl)
```

If no Content Parameter control is present in the Import Template View control for a marker in the imported view, the marker's content is rendered as is.

You can use the Java class `UIParameter` an Import Template View control to set property values on the imported view's page bean. The name of the `UIParameter` specifies the name of the page bean property, and the value specifies the initial value for the property.

In the above example, if View B's page bean had a property called `pageTitle`, a `UIParameter` with the name `pageTitle` could be used by View A to set that property (for example, it might set the property's value to "Customers"). View B could then use the page bean property to display the `pageTitle`, or use it for some other purpose, such as the input to a Web service.

Using Content Parameters with Page Templates

You can use one or more Content Parameter controls as children of an Import Template View control to inject content from the importing view to a page template when the page is displayed. For more information, see [“Working with Page Templates” on page 27](#).

➤ To use a Content Parameter control with a page template

1. In the Solutions view, double-click the page template view you want to work with to open it in the editor.
2. On the design canvas, select or add the control you want to use as a marker.
3. On the **General** tab of the Properties view, type a unique ID in the **ID** field.
4. In the Solutions view, double-click the target view to open it in the editor.
5. Drag an Import Template View control into the target view design canvas. For additional information, see [“Importing a View into a VDL File” on page 27](#).
6. In the Palette view, expand **CAF Html > Control > Logic**, and then drag a Content Parameter control to the Import Template View.
7. Select the Content Parameter control, and in the **Name** field of the Properties view, type the same value you used for the ID in step 3.

Navigation in Web Applications

In web applications, the developer is responsible for providing all page navigation within the application. The following table lists the controls that Composite Application Framework provides:

CAF control	Function
Breadcrumbs	Bread crumbs navigation as a list of links, starting with the home page. For more information, see “Hiding Breadcrumbs on the Home Page” on page 32 .
Popup Menus	Top-level navigation as a list of links. Click or hover over one of the top-level links displays the children of that page as a popup menu of second-level links. These menus do not include the home node.

CAF control	Function
Static Menus	Top-level navigation and second-and third-level pages as a list of links. These menus do not include the home node.
Toggle Menus	Top-level navigation as a list of links. Click or hover over one of the top-level links displays the children of that page as a list of second-level links. These menus do not include the home node.

For more information about these controls, see *webMethods CAF Tag Library Reference*. For information about using *webMethods CAF Tag Library Reference*, see [“Finding Information about CAF Controls” on page 67](#).

The hierarchy for each of these controls is provided by an application navigation bean, App Nav. Once established, all navigation controls in the web application use the same App Nav bean. In Software AG Designer, use the Application Navigation Configuration Editor to associate the menus with the pages in the application. For more information about configuring application navigation, see [“Configuring Application Navigation” on page 30](#).

Configuring Application Navigation

Use the Application Navigation Configuration Editor to associate the page navigation controls with the pages in a web application. For example, you may have a tree hierarchy with a home page and child pages, which in turn have their own children.

You can use the Application Navigation Configuration Editor to design the hierarchy. The App Nav bean exposes the hierarchy. The Composite Application Framework navigation controls use the App Nav bean to enable the navigation.

To apply this procedure, you must have two or more pages in your application.

➤ To create an application navigation configuration

1. In the **User Interfaces** folder in the Solutions view, open the web application you want to work with and double-click the **Web Navigation** node.
2. In the Web Application Navigation editor, click the **Configuration** tab.

By default, the Home node is provided. Click the Home node to view its Navigation Page Details.

3. To add a navigation node, do either of the following:
 - Select the parent node (Home in the initial case) and click **Add**. The editor adds a new node under the parent node, and gives it an arbitrary title. Expand the Home node to see the new node.
 - Drag a web application view from the Solutions view and drop it under the node in the editor that will serve as its parent.

4. Select the new navigation node and make changes in the Navigation Page Details as necessary:
 - In the **Title** field, accept the existing title or type a new title for the node. If a view is not yet specified, do either of the following:
 - In the **ID** field, click **Browse**, select the view for this node, and click **OK**.
 - In the **URL** field, click **Browse**, select the view for this node, and click **OK**. Values take the form */project/view*. You can also type an external URL.
 - (Optional) In the **Resource Key** field, if you have set a resource key for the view, type the resource key. The value of the resource key is used in place of the text in the Title field.
5. There are some other actions you can perform on the Application Navigation Configuration Editor:
 - To delete a node, select that node and click **Delete**.
 - To move a node within a parent node, select the node and click **Up** or **Down**.
 - To move a node from one parent node to another, drag the node. When you drag a node, all of its child nodes move with it.

Adding a Home Link to a Web Application

You can provide navigation back to the home page from anywhere in a web application by adding a home page link to a header file that is imported into all pages of the web application. Use the App Nav bean to configure the link to the home page.

You must configure the navigation of the application before you can use the following procedure. For more information, see [“Configuring Application Navigation” on page 30](#).


➤ To add a link to the home page on all pages in a web application

1. In the UI Development perspective, click the Solutions view, expand the **User Interfaces** folder, expand the project, and then expand the **WebContent** folder that contains the header view.
2. Double-click the header view to open it in the editor.

Note:

If you have only recently specified the web application’s navigation settings, you might have to click the Refresh toolbar button in the Bindings view to see the App Nav Bean to be able to complete the next steps.

3. In the Palette view, expand **CAF Html > Control > Output** and drag the **Link** control to the desired location in the header view design canvas.

4. Select the **Link** control in the design canvas, click the Properties view, and then click the **Value** tab.
5. In the **Value** tab, do the following:
 - a. Next to the **Value** field, click  to create a binding expression to the home page. In the Expression Binding dialog box, select **Managed Beans > header > App Nav Bean > Root > Url** and click **OK**.
 - b. In the **Label** field, type the link display text.

The control is now bound to the root view as defined in the App Nav bean.

6. Save the web application project.


Hiding Breadcrumbs on the Home Page

You can use the Breadcrumbs control to display levels of navigation starting from the home page. Before you can use breadcrumbs, you must configure the application navigation. For more information, see [“Configuring Application Navigation” on page 30](#).

While breadcrumbs are useful for navigation when the user is somewhere inside the hierarchy of web pages, they provide no useful purpose on a top level page. The following procedure describes how to place the Breadcrumbs control on a page and then hide the control when the home page is displayed.

For more information about these controls, see *webMethods CAF Tag Library Reference*. For information about using *webMethods CAF Tag Library Reference*, see [“Finding Information about CAF Controls” on page 67](#).

➤ To add a breadcrumbs control and hide it on the home page

1. In the UI Development perspective, click the Solutions view, expand the **User Interfaces** folder, expand the project, and then expand the **WebContent** folder that contains the header view.
2. Double-click the header view to open it in the editor.
3. In the Palette view, expand **CAF Html > Control > Webapp**, and then drag the Breadcrumbs control to the desired location in the header view design canvas.
4. Select the Breadcrumbs control in the design canvas.
5. In the Properties view, click the **General** tab, then click  next to the **Rendered** field.
6. In the Expression Binding dialog box, select **Managed Beans > header > App Nav Bean > Current > ID**, then click **OK**.

7. In the **Rendered** field, modify the binding expression to render the Breadcrumbs control when the ID of the current view is not the same ID as the home page by appending `!="/home.view"` to the binding expression. For example:

```
#{header_view.appNavBean.current.id !="/home.view"}
```

For information about how to locate and copy the ID of the home page, see [“Finding the Home Page ID” on page 35](#).

8. Click **OK**.
9. Save the web application project.

Setting Application Initialization Parameters

Application initialization parameters are variables that you can add as default values in a web application. Application initialization parameters are read-only values that are used at runtime, but the server administrator can alter them as needed.

You can use application initialization parameters to customize the behavior of a web application, either to display the value or use it in a binding expression. Application initialization parameters are expressed only as strings, and are valid only within a web application. For more information about variables that can span web applications, see [“Setting Environment Variables” on page 34](#).

➤ To create an application initialization parameter for a web application

1. In the UI Development perspective, click the Solutions view, expand the **User Interfaces** folder, expand the project, and then expand the **WebContent** folder that contains the web application views.
2. Double-click the view you want to work with to open it in the editor.
3. In the Bindings view, expand **Implicit Variables > Application Initialization Parameters** to view the existing initialization parameters.
4. Right-click **Application Initialization Parameters** and click **Add > Data**.
5. In the Add New Property dialog box, type a name for the initialization parameter in the **Property Name** field and click **Finish**.

The new application initialization parameter appears in the Bindings view under the Application Initialization Parameters node.

6. Click the new application initialization parameter in the **Bindings** view.
7. In the **Properties** view, click the **Data Binding** tab and in the **Value** field, type a value for the parameter.

8. Save the web application project.

You can now drag this parameter from the Bindings view to a control on the design canvas to bind the control to the parameter.

Setting Environment Variables

Environment variables are variables you can add as default values in a web application. The environment variable values are used at run time, but the server administrator can alter them as needed. You can use environment variables to customize the behavior of a web application, either to display a value or use it in a binding expression.

Unlike application initialization parameters, environment variables enable you to use primitive and object data types, and to share these variables with other web applications. For more information about application initialization parameters, see [“Setting Application Initialization Parameters” on page 33](#).

Note:

For primitive data types, you can specify a default value, and change it at run time either programmatically or manually, as an administrator. You must programmatically put object data types in to the JNDI tree.

Environment variables operate in the Java Naming and Directory Interface (JNDI) environment. While use of JNDI is widespread in application servers, you should make sure a specific application server supports JNDI before attempting to deploy a web application that uses environment variables. The current web application, or another application, can programmatically update environment variables providing a way to share services between web applications in the same container.

Common configuration parameters to define as environment variables are anything that might differ for an application deployed in different environments, for example, variables that will have separate values for development, staging and production environments. When you add a web service connector, doing so automatically adds common web service configuration values such as endpoint, user name, and password to the application environment variables.

➤ To create an environment variable for a web application

1. In the UI Development perspective, click the Solutions view, expand the **User Interfaces** folder, expand the project, and then expand the **WebContent** folder that contains the web application views.
2. Double-click the view you want to work with to open it in the editor.
3. In the Bindings view, expand **Implicit Variables > Environment Variables** to view the existing environment variables.
4. Right-click **Environment Variables** and click **Add > Data**.

5. In the Add New Property dialog box, type a name for the environment variable in the **Property Name** field.
6. In the **Property Type** field, type a primitive or object data type, or click the **Browse** button to select an existing data type.
7. Click an option to specify if the data type is to be scalar, a modifiable list, or a read-only array and then click **Finish**.

The new environment variable appears in the Bindings view under the **Environment Variables** node.

8. In the Bindings view, select the newly created environment variable property.
9. In the Properties view, click the **Data Binding** tab.
10. On the **Data Binding** tab, in the **Value** field, type a variable for the property.
11. Save the web application project.

You can drag the new environment variable property from the Bindings view to a control on the design canvas to bind the control to the property.

Finding the Home Page ID

When you use a Breadcrumbs control on a page and you want to hide the control on a home page, you must specify the ID of the home page.

For more information about hiding the Breadcrumbs control, see [“Hiding Breadcrumbs on the Home Page” on page 32](#).

➤ To determine the ID of a page

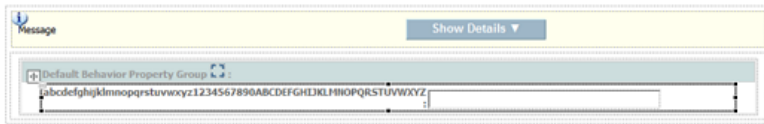
1. In the **User Interfaces** folder in the Solutions view, open the web application with which you want to work and double-click the **Web Navigation** node.
2. In the Web Application Navigation editor, click the **Configuration** tab at the bottom and select the home page.
3. In **Navigation Page Details**, the value in the **ID** field is the home page for the web application. You can use the ID in a binding expression to hide the breadcrumbs.

Managing Text Overflow in Property Lines

When you are designing a Composite Application Framework user interface that contains the Property Group and Property Line controls, you can manage the label text overflow for those controls.

For more information about the Property Group and Property Line controls, see the *webMethods CAF Tag Library Reference*.

By default, if the text is too long for the width available in the specified Property Line control label, the text is wrapped into one or more additional lines. This wrapped label text can occur with very long labels, or when translating English words and phrases into other languages, such as German or Russian. The images below show the default behavior of very long Property Line control label text (that is, without any of the compensating property settings). At design time, the label text appears as:



When the portlet is published to My webMethods Server, the Property Line control label text is wrapped to a second line aligned to the right.

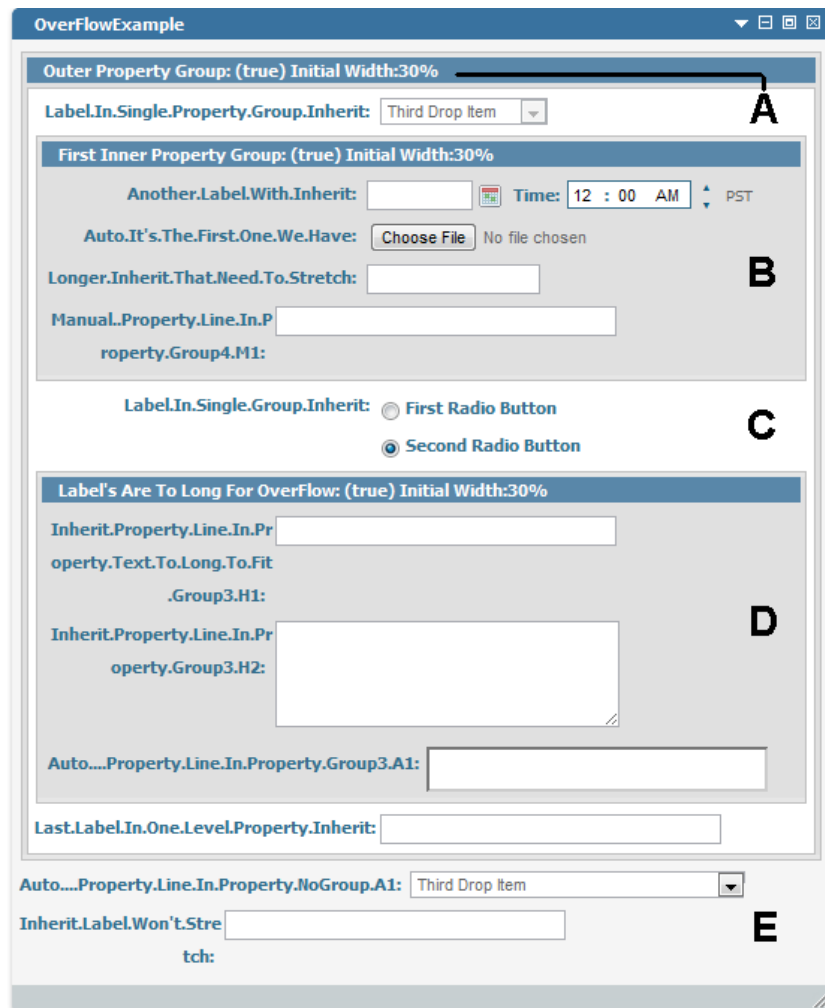
If the Property Line control's label text contains spaces instead of being one long word, the line breaks are automatically adjusted in HTML and managing the overflow text is less of an issue.

One solution to managing label text overflow is to externalize the label of the property line into a translatable resource bundle. However, this is a time consuming process.

A more efficient approach is to use the `Group Overflow Control` property in the Property Group control and the `Overflow Control` property in the Property Line control to manage the overflow of label text.

- When set to *true*, the `Group Overflow Control` property in the Property Group control overrides the label width of any child Property Line controls and adjusts them to keep all labels aligned. There is no default setting for this property.
- By default, the `Overflow Control` property in the Property Line control is set to *inherit*, which means that the setting of the `Group Overflow Control` property in the Property Group control is automatically applied to the Property Line control. Other possible values for the `Overflow Control` property are:
 - *auto*. Adjusts the label length automatically to match the length of other property line labels when the `Group Overflow` property in the parent Property Group control is set to *true*. When there are no other property line labels to match, *auto* adjusts the lone property line label
 - *manual*. When the `Group Overflow` property in the parent Property Group control is set to *true*, the overflow text in the Property Line control label is not adjusted.

The following images show an example of various Property Group and Property Line control options using the `Group Overflow Control` and the `Overflow Control` properties.



The portlet example above shows Property Group panels with Property Line panel children, and Property Line panels without a parent Property Group panel. Each example set is labeled with an alphabetical letter (A, B, C, D) and the Property Group and Property Line label names illustrate the configured overflow property setting.

The examples show the following:

- **A.** Demonstrates an Outer Property Group panel with two child Property Groups and three Property Line panels that are placed outside of the nested Property Group children. In this example:
- **B.** The First Inner Property Group control has its Group Overflow Control property set to *true*.
 - Its first child Property Line control (with the drop-down list) has its Overflow Control property set to *inherit*.
 - The next child Property Line control (with the radio buttons) has its Overflow Control property set to *auto*. See the note below about radio button use.
 - The third child Property Line control's Overflow Control property is set to *inherit*.
 - The last child Property Line control's Overflow Control is set to *manual*.

The first three Property Lines are adjusted and aligned right, while the last line is not adjusted (because of its *manual* setting), causing the label to overflow to the next line.

Note:

If you are using a Radio Button Group control with its `Layout` property set to *linedirection* in a Property Line control, the Radio Button Group control's layout consumes 100% of the available space, preventing the Property Line control's `Overflow Control` property from managing the label text width. For more information about the Radio Button Group and Radio Button controls, see the *webMethods CAF Tag Library Reference*.

- C. Demonstrates a single Property Line panel with a Radio Button control. The Property Line panel has its `Overflow Control` property set to *inherit*, inheriting its behavior from the parent, in this case the outer Property Group. The single Property Line panel is not influenced by the behavior of the inner Property Group control's `Group Overflow Control` setting.
- D. Demonstrates a Property Group panel with two Property Line controls that have their `Overflow Control` property set to *inherit*. However, the labels are too long for the `Overflow Control` property setting to adjust the alignment evenly. A third Property Line control has its `Overflow Control` property set to *auto*, enabling the label to adjust to the available space by relocating the text input box.
- E. Demonstrates two Property Lines that are located outside of any Property Group control. The first Property Line control has its `Overflow Control` property set to *auto*, so its long label adjusts to the available width. The second Property Line control has its `Overflow Control` property set to *inherit* (the default), but since it is not a child of a Property Group control, the label does not adjust to the available width.

Placing Modal Dialog and Popup Panel Controls

You can specify the placement of the CAF Modal Dialog and Popup Panel controls with the control properties described below. If you do not use the properties described here, you must use JavaScript to set their placement and offset relative to another user interface element.

For more information about the Modal Dialog and Popup Panel controls, see *webMethods CAF Tag Library Reference*.

➤ To specify the placement of a Modal Dialog or Popup Panel control

1. Locate the web or portlet view you want to work with and open in the editor.
2. Select the Modal Dialog or Popup Panel control you want to work with, and specify the following properties in the Properties view:

For Popup Panel controls:

- **For** (on the **General** tab). Identifies the control ID of the button or other user interface element that is used as the base from which to offset the Modal Dialog.

You do not have to set the position property in the control specified in the **Target For** property.

- **Position** (on the **Display** tab). Indicates where the Modal Dialog is positioned in relation to the control listed in the **Target For** property.
- **Offset X** (on the **Display** tab). Sets the Modal Dialog to the left of the default position. Set the **Offset X** value to the number of pixels to the left on an X/Y axis.
- **Offset Y** (on the **Display** tab). Places the Modal Dialog from the top of the default position. Set the **Offset Y** value to the number of pixels from the top on an X/Y axis.

For Modal Dialog controls, specify the following on the **Extra Offset** tab:

- **Target For**. Lists the control ID of the button or other user interface element that is used as the base from which to offset the Modal Dialog.

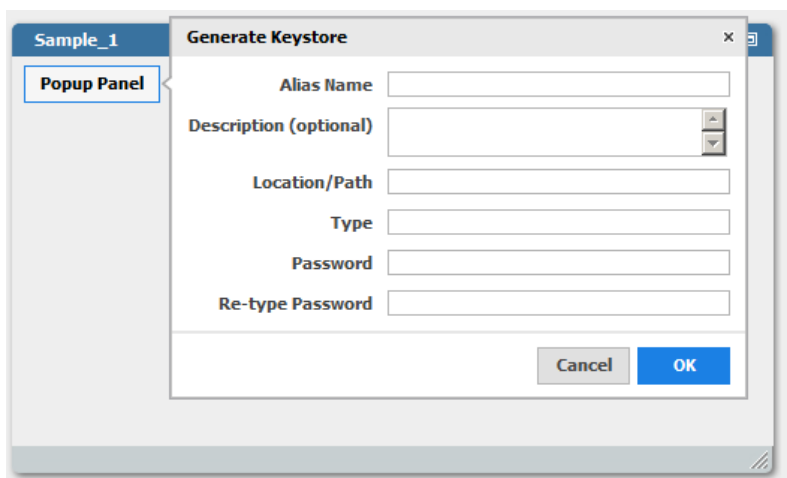
You do not have to set the position property in the control specified in the **Target For** property.

The placement for the Popup Panel control works the same way as the properties listed above for the Modal Dialog, except that you specify the **For** property on the **General** tab in the **Properties** view.

- **Position**. Indicates where the Modal Dialog is positioned in relation to the control listed in the **Target For** property.
- **Offset X**. Sets the Modal Dialog to the left of the default position. Set the **Offset X** value to the number of pixels to the left on an X/Y axis.
- **Offset Y**. Places the Modal Dialog from the top of the default position. Set the **Offset Y** value to the number of pixels from the top on an X/Y axis.

3. Save the web or portlet view.

For example, using the **Position** property in the Popup Panel, you can set the control to display in the screen as offset from the another user interface element, such as a Button control. The following image shows Generate Keystore Popup Panel offset to the right of the Popup Panel button:



If you specify an **Offset X** property of 10 pixels, Composite Application Framework places the Popup Panel 10 pixels to the right of the control listed as the target in the **Position** property. If you set the **Position** property variable to *center*, the Popup panel is placed 10 pixels offset from the center of the screen.

Importing CAF Projects

You can import an existing Composite Application Framework (CAF) project in to a Software AG Designer workspace. Designer automatically adds the imported project to your designated workspace.

Note:

If the project you import is implemented with .view pages and you want to migrate it, or any of the views within it, to JSF Facelets XHTML format, you must do so after you import the project. For more information, see [“Working with Facelets” on page 89](#).

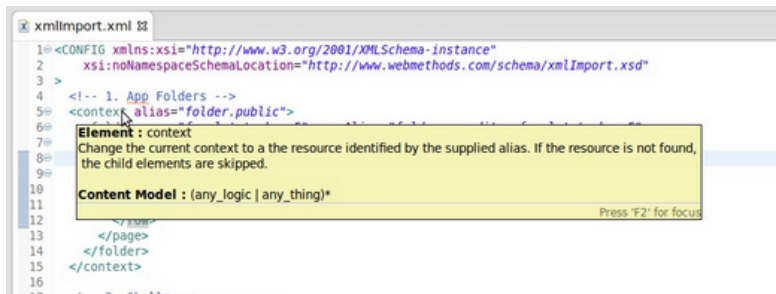
➤ To import a CAF project into your workspace

1. In Designer, click **File > Import > Software AG > Existing CAF Projects into Workspace**, and then click **Next**.
2. In the Import dialog, choose one of the following:
 - In **Select root directory**, click **Browse** to choose the directory location of your Composite Application Framework project.
 - In **Select archive file**, click **Browse** to import and extract a zip file into your workspace, and then click **Open**.
3. Click **Finish**.

The Project Explorer and Navigation views now display the imported CAF project, and the Solutions view displays the CAF project in the User Interfaces folder.

Validation Support for xmlImport Files

The Designer XML editor provides built-in validation and code completion assistance for xmlImport files, if the file contains a schema declaration. When editing these files, pop-up displays appear with information about elements, attributes, and content models when the cursor hovers over the related XML text. For example:



All templates that create xmlImport files declare the XSD schema on the root node as follows:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.webmethods.com/schema/xmlImport.xsd">
</config>
```

This occurs by default for all new projects. To add this schema support to existing projects, you must manually update the templates for their xmlImport files by adding the schema attributes described above to the <config> element, and make any corrections for non-conforming or invalid structures in the XML file.

Specifying a Target Runtime Server

You can use an application server other than My webMethods Server when creating Composite Application Framework dynamic web applications. The supported Composite Application Framework features depend on the version of the application server and the version of Java technology supported by the specific application server.

➤ To specify a target runtime server for a web application

1. In the UI Development Perspective, click **File > New > Web Application Project**.
2. In the New Web Application Project dialog, in **Project Name**, type a name for the web application project.

Note:

When naming the project do not use specific characters. For more information, see [“About CAF File Names and Path Lengths” on page 20](#).

3. In **Target Runtime**, click **New Runtime** to open the New Server Runtime Environment dialog box.
4. Expand the folder of a supported application server, click the server version to use for the application, and click **Next**.
5. In **New Server Runtime Environment**, select the installation directory for the server and click **Finish** to add the new server environment to the workspace.

When the new server appears in the Servers view, you can publish your application to the server. For more information, see [“Adding a Portlet Application Project to My webMethods Server” on page 68](#) and [“Publishing a Portlet Application to My webMethods Server” on page 69](#).

Debugging CAF Applications Remotely

While writing and testing your code, you can remotely debug Composite Application Framework applications on a remote instance of My webMethods Server. To use the Software AG Designer built-in Java debugging features, you must run My webMethods Server in debug mode.

You can start the server in debug mode from the command line using the option `-d`. For more information about working with the server commands and starting the server in debug mode, see *Administering My webMethods Server*.

To set general debug preferences in Designer, select **Window > Preferences > Run/Debug**. For more information, see the Eclipse Java development user guide available in the Software AG Designer online help.

➤ To debug CAF applications remotely

1. In the UI Development perspective, click the Servers view.
2. In the Servers view, select the remote My webMethods Server you want to work with, and then click the debug mode tool bar button .

The default My webMethods Server Java debug port number is 5000. You must also verify that the remote My webMethods Server instance is configured to run in debug mode.

When the remote My webMethods Server is started in debug mode the status display as **[Debugging, Synchronized]**.

4 Developing webMethods OpenUI Applications

■ About webMethods OpenUI	44
■ About webMethods OpenUI Shell	44
■ About JSF 2.x Facelets Support	45
■ Additional Resources for Developing OpenUI Applications	45
■ Security in OpenUI Applications	46

About webMethods OpenUI

webMethods OpenUI enhances the functions of Composite Application Framework (CAF). With OpenUI, you can build CAF applications, using prevailing design principles, such as responsive web design and single page applications. OpenUI enables you to:

- Customize your application by using third-party JavaScript and CSS libraries and frameworks.
- Use the existing CAF and My webMethods Server framework for security, navigation, task management, and web services.
- Develop all user interface components in Software AG Designer, including My webMethods Server portlets, pages, shells, shell rules, and skin rules.

OpenUI provides the following functions:

- **webMethods OpenUI Shell** - an implementation of My webMethods Server shells, based on JavaServer Faces (JSF) Facelets. You use OpenUI Shell for the development of customized shell templates. For more information about OpenUI Shell, see [“About webMethods OpenUI Shell” on page 44](#).
- **JSF 2.x support** - an implementation of portlet views, based on JSF Facelets. You can customize portlets, using third-party libraries. For more information about the JSF Facelets functionality in CAF and OpenUI applications, see [“About JSF 2.x Facelets Support” on page 45](#).

Apart from the functionality that OpenUI introduces, OpenUI applications that are not heavily customized would behave as, and use all the functions and components of CAF portlet applications, such as:

CAF tags

Web services

Converters and validators

Java Content Repository (JCR) providers

CAF events and notifications

Attachment providers

You deploy OpenUI applications to My webMethods Server in the same way that you deploy portlet applications. For more information about how to deploy an OpenUI application, see [“Publishing an Application to My webMethods Server” on page 162](#).

About webMethods OpenUI Shell

webMethods OpenUI Shell is an implementation of My webMethods Server shells that uses a Facelets page template for creating and rendering an xhtml page. OpenUI Shell enables you to customize the look and feel of the page by using standard and custom HTML, CSS, and JavaScript frameworks along with CAF portlets and JSF tags.

The page template may or may not include the framing around the page body: header, footer, leftnav, and rightnav. You can use portlets to render the body of the page, either directly in the page templates provided in Software AG Designer or by delegating to the existing My webMethods Server resource renderers. Conversely, you can have an OpenUI application without any portlets.

For more information about developing OpenUI Shell projects and pages, see [“Getting Started with webMethods OpenUI Shell Page Development” on page 73](#). For more information about CAF JSF tags, see *webMethods CAF Tag Library Reference*.

About JSF 2.x Facelets Support

With JSF 2.x, portlet views in CAF and OpenUI applications are implemented as .xhtml files with Facelets support. Facelets support enables you to use:

- Third-party libraries to customize the portlet view
- CAF JSF tags to add functionality to the portlet view
- Java annotations to declare managed beans
- Implicit navigation between portlet views

For more information about the JSF Facelets functionality and working with Facelets in CAF and OpenUI applications, see [“Working with Facelets” on page 89](#).

By default, Software AG Designer creates portlet applications and new view pages using the .xhtml file format. You can create legacy view pages in the .view format by specifying a .view file name extension when creating a view page.

For more information about the Facelets page declaration language, see the Oracle Java documentation website.

Additional Resources for Developing OpenUI Applications

You can find additional information about how to use OpenUI, including code samples and usage examples, on [Github](#) and the Software AG TECHcommunity website.

The [OpenCAF Component Showcase](#) shows how you use most components and functions of OpenUI. You can find code samples for the following components:

- All user interface tags, including JSF standard tags, caf, caf_f, caf_h, cafp_f, cafp_h, jcr_h, mws_f, mws_h, and mws_ui tags
- Client-side-model JavaScript for validators
- Custom validators and converters
- OpenUI Shell page templates
- My webMethods Server skins
- Integration of third-party JavaScript frameworks such as AngularJS and JQuery

- User interface patterns, including drag-and-drop functionality using HTML5, Accordion panels, and master/details patterns
- My webMethods Server resource renderers that you can use for the generation of specific content

In addition, you may find the following articles and code samples useful:

- [Dynamic Import of Views for CAF and OpenCAF](#)
- [Using OpenCAF with AngularJS and Bootstrap](#)

Security in OpenUI Applications

You define security roles for the users of an OpenUI application in the same way that you define security roles for a portlet application. For OpenUI projects that do not contain any portlets, you define security roles directly in the WebContent\WEB-INF\web.xml file of the project in Software AG Designer as follows:

```
<security-role>
  <description/>
  <role-name>execute_some_action</role-name>
</security-role>
```

For more information about adding security roles to a portlet application and mitigating security vulnerabilities in portlets, see [“Security in CAF Applications” on page 47](#).

5 Security in CAF Applications

■ Security Roles in Web and Portlet Applications	48
■ Mitigating Portlet Security Vulnerabilities	50

Security Roles in Web and Portlet Applications

A security role is a standard J2EE concept for determining who can have access to a site, a page, or a control on a page. Security roles identify My webMethods Server groups, roles, and users, and define permissions to view or modify resources in an application.

When designing the user interface for a web or portlet application, you can define security roles for the users of the application according to your organization's security policies. In Composite Application Framework, you can define application and portlet security roles to implement role-based security.

Security roles also provide a means of binding a functional privilege contained in a task to a control. In this case, the security role has no direct relationship to any My webMethods Server user role. In this way, you can specify if the control is rendered, disabled, or read-only, depending on the user's security role.

For more information, see:

- [“Binding a Security Role to a Control” on page 155](#)
- [“Binding Security Roles in an Access Control Panel” on page 155](#)

Add a Security Role to a Web Application

You can create security roles for a web application projects in Composite Application Framework.

- For more information about security roles, see [“Security Roles in Web and Portlet Applications” on page 48](#).
- For information about adding a security role to a portlet, see [“Add a Security Role to a Portlet Application” on page 49](#).

➤ To add a security role to a web application

1. Open a web application project in the Navigator view or the Package Explorer view, expand the **WebContent** node, and drag the View Declaration Language (VDL) file for a view to the editor.
2. In the Bindings view, expand the **Implicit Variables** node.
3. Click the **Application Security Roles** node, then right-click and select **Add > Application Security Role**.
4. In the Web Application Security Role dialog box, type a name in the **Role Name** field, and in **Role Description**, type a brief description of the role.
5. Click **Finish**.

Note:

You can also add and modify security roles directly in the web.xml file, using either the XML editor or the text editor.

Add a Security Role to a Portlet Application

You can create security roles for a portlet application projects in Composite Application Framework.

- For more information about security roles, see [“Security Roles in Web and Portlet Applications” on page 48](#).
- For information about adding a security role to a web application, see [“Add a Security Role to a Web Application” on page 48](#).

When the portlet is published to an instance of My webMethods Server, the security role defined in this procedure is listed in the My webMethods Server roles as a static role. The system administrator user, sysadmin, is the initial member of the role on My webMethods Server. You can assign users, groups, or other roles as members of the role. For more information, see *Administering My webMethods Server*.

Tip:

You can view the security roles defined for the project in the Bindings view under **Implicit Variables** in the **Application Security Role** node, or in the Portlet Application Configuration Editor, as described below.

➤ To add a security role for a portlet application

1. Open a portlet application project in the Navigator view or the Package Explorer view.
2. Expand the **WebContent** node, expand the **WEB-INF** folder, and then double-click the portlet.xml file
3. Click the **Configuration** tab.
4. Expand the **Portlets** node, expand the portlet, click **Security Roles**, and then click **Add**.
5. In the Create Security Role Reference dialog box, type a name in the **Role Name** field, and in **Role Description**, type a brief description of the role.
6. (Optional) If you have created functional privileges in the application in a privileges.xml file and want to add a privilege to the security role, click **Privileges**, click a functional privilege in the list, and then click **OK**.
7. (Optional) If you want to create a role link, click **Add**, type a name for the role link, and then click **OK**.
8. Click **Finish**.

Note:

You can also add and modify security roles directly in the web.xml file, using either the XML editor or the text editor.

Mitigating Portlet Security Vulnerabilities

Portlets are vulnerable to code injection attacks, which occur when malicious code is inserted into known or trustworthy web sites. An attacker can construct a URL to invoke actions even if there are no portlet action links in the page.

The malicious attacker uses the URL to invoke any of the public, zero-argument methods on any of the portlet's managed beans with a binding expression in the URL's `targetAction` parameter. You can mitigate this portlet security vulnerability by enabling annotated portlet actions.

In Composite Application Framework (CAF), you can use annotated portlet actions in the Extended Portlet Url control. When the user clicks the Extended Portlet Url link, the action request performs the action named in the `targetAction` property of the Extended Portlet Url control.

When you add a control that uses the Extended Portlet Url control to a view, set the `Type` property to `action` to make the link an action request that invokes the `targetAction` when loaded. The `Anti-XSRF Token` property instructs the portlet container to check for a valid `axsrft` token before allowing the target action to run. You must add the `@PortletAction` annotation and define the `targetAction` in the source code providing the methods to fulfill the action request.

This approach enables you to limit actions that are callable using the portlet URL to a fixed set of methods that you specify. Configure the portlet to limit portlet URL actions to those methods on the portlet's preferences bean or active page bean that have been annotated with the `@PortletAction` annotation.

The following example demonstrates a portion of the `portlet.xml` file with the annotated portlet actions `init-param` tags. CAF generates the `init-param` for the `ANNOTATED_PORTLET_ACTIONS` when the portlet is created.

```
<portlet>
<portlet-name>My_company_portlet</portlet-name>
  <portlet-class>com.webmethods.caf.faces.portlet.FacesPortlet</portlet-class>
    <init-param>
      <name>com.webmethods.caf.faces.portlet.INIT_VIEW</ name>
      <valu e>/My_company_portlet/default.view</value>
    </init-param>
    <init-param>
      <name>ANNOTATED_PORTLET_ACTIONS</name >
      <value>true</value>
    </init-param>
    :
    :
  </portlet>
```

New portlets created in CAF automatically have the `init-param` added. For portlet applications created in older versions of CAF, be sure to check for the use of annotated portlet actions. You may have to manually add the `init-param` to the `portlet.xml` file.

Add the Extended Portlet Url control to a Portlet Simple Link or Portlet Url Script. For more information, see the CAF Tag Library reference documentation for the Extended Portlet URL, Portlet Simple Link, and the Portlet URL Script controls. Do not use binding expressions when defining the portlet `targetAction` property, instead use a simple method name, such as `SayHelloWorld`.

When annotated portlet actions are enabled, the action handler source code from a portlet action link must contain the `@PortletAction` annotation. The following code snippet shows the `@PortletAction` annotation.

```
@PortletAction
    public String doSayHello() {
        error(FacesMessage.SEVERITY_INFO, "Hello, the 'doSayHello' portlet action
was executed", null);
        return OUTCOME_OK;
    }
```

You can enable or disable the `axsrft` token for specific action methods. The following example shows the `axsrft` token and how it applies to the `@PortletAction`.

```
@PortletAction(axsrft = true)
    public String doSayHelloWithAxsrfRequired() {
        error(FacesMessage.SEVERITY_INFO, "Hello, the
'doSayHelloWithAxsrfRequired' portlet action was executed", null);
        return OUTCOME_OK;
    }
```

When annotated portlet actions are disabled, you can invoke the action method from the portlet action link with a full binding expression. For more information about binding expressions, see [“About the Bindings View” on page 152](#).

6 Getting Started with Web Application Development

■ About Web Applications	54
■ Creating a Web Application Project	54
■ Adding a View to a Web Application	55
■ Adding a Control to a Web Application View	57
■ Preparing a Web Application for Localization	57

About Web Applications

A web application is a resource represented as a hierarchy of files and functions accessed using a web browser. In Composite Application Framework (CAF), you can create a web application project to deploy to My webMethods Server.

In a CAF web application, you create views (pages) intended to be displayed by an application server as web pages. You must first create a web application project, and then add one or more views for deployment to My webMethods Server.

You create a CAF web application project in the UI Development perspective in Software AG Designer. The UI Development perspective contains a set of views and editors that support development of a web or portlet application.

When you create a web application project that uses My webMethods Server as the runtime application server, CAF provides access to all of the Software AG Designer supported controls in a Dynamic Web Module, including JavaServer Faces, and the default configuration for My webMethods Server.

Note:

Events, which are created and maintained by means of the Events Editor, are not supported for CAF web application projects. Events are only supported in CAF portlet application projects.

Dynamic Web Application Projects

In Composite Application Framework, you can create dynamic web applications using JavaServer Faces and Ajax.

- A dynamic web application project enables to the user to interact with the content of the web application.
- A static web application contains HTML pages with content that enables the user to read the content or download a file loaded as a link on the page.

With CAF, JavaServer Faces, and Ajax you can create dynamic web applications that provide custom content and interactivity to your users.

My webMethods Server is the default application runtime server for CAF web application projects.

Creating a Web Application Project

When you create a web application project in Software AG Designer the default application server is My webMethods Server. Although the **Target Runtime** list in the New Web Application Project wizard contains other server types, only My webMethods Server is supported as a target runtime.

➤ To create a web application project

1. In the UI Development perspective: **File > New > Web Application Project**.

2. In the **Project Name** field, type a name for the project. For information about the supported characters for the name, see [“About CAF File Names and Path Lengths” on page 20](#).
3. In the **Target Runtime** list, accept **My webMethods Server** as the default server.
4. In **Dynamic web module version**, accept the default value to define a Servlet API version. If you need to specify an earlier version, click the drop-down arrow.
5. In **Configuration**, accept the default **CAF Web Application** or click the drop-down arrow to select from a list of other available configurations.
6. Click **Modify** if you want to add other facets to the project. Select the facet you want to add and then click **OK**.
7. (Optional) In **Working sets**, click **Add Project to working set** to and then click **Select** to select a working set resource for the project.
8. In **Initial Project Contents** area, in the **Project Templates** list, accept the default value **Empty Project**, or click **Starter Web Application**.
 - **Empty Project** (default). The JSF project is created with a minimum of folders and resources. You will create your project structure and view files and other resources later with the New Portlet and New View wizards.
 - **Starter Web Application**. This option creates a JSF project that contains a basic set of default views. For more information, see [“About Web Application Project Templates” on page 24](#).
9. Click **Finish** to create the web application project with default settings, or click **Next** to continue configuring the project to your needs.

You can view and work with the new web application in the Solutions view, the Navigator view, or the Package Explorer view.

Adding a View to a Web Application

A VDL file contains controls and bindings to enable the user to interact with the application and the data underlying it. You can drag a control from the Palette view onto the VDL file displayed in the editor. For more information, see [“Adding a Control to a Web Application View” on page 57](#).

For information about file naming conventions, see [“About CAF File Names and Path Lengths” on page 20](#).

» To add a view to a web application

1. In the UI Development perspective: **File > New > Web Application View**.

2. In the JSF Web Application View File wizard, specify the web application project to which you want to add the view by doing one of the following:

- Accept the displayed web application project.
- Select a web application project from the list.
- Type the name of an existing web application project

3. In the **File name** field, type a name for the VDL file.

Note:

If you type the file name only, without a file name extension, the view file will be created in .xhtml format by default. If you type the file name with a .view file name extension, the view file will be created as a legacy page view.

(Optional) Click **Advanced** to configure a link between the VDL file and a file in your file system.

Note:

From this point forward, you can click **Finish** to complete the creation of the VDL file with the default settings, or click **Next** to change the default configuration.

4. Click **Next**.
5. On the **Web View Options** page, select the template to use from the **Template** list.

(Optional) Clear the **Support Application Navigation** check box if you have no need for navigation functionality. When selected, this option enables the view to be able to participate in the web application navigation model.

6. Click **Next**.
7. In **Managed Bean**, accept the default name, or type a new bean name, and select a scope from the **Managed Bean Scope** list for the managed bean.
8. Click **Next**.
9. On the Java Type page, do any or all of the following:
 - Click **Browse** to specify a different package.
 - Click **Add** to select an existing interface for the view.
 - Click **Generate Comments** to automatically add comments while designing the view. Click the **here** link to access preferences for Java code templates.
10. Click **Finish**.

The new VDL file opens in the editor and is listed in the **WebContent** folder of the project.

Adding a Control to a Web Application View

In your Composite Application Framework web application, use the editor's design canvas to add controls to the view. You add a control by dragging the control from the Palette view onto the design canvas and placing it within the view frame.

Note:

The structure of the control folders in the Palette view is different for .view files and .xhtml files. However, the control functionality is the same.

➤ To add a control to a view

1. In the UI Development perspective, in the Navigation view, open the web application project and expand the **WebContent** folder.
2. Double-click the View Declaration Language (VDL) file to which you want to add a control, or drag the file onto the design canvas.
3. In the Palette view, do either of the following to find the control you want to work with:
 - Begin typing the control name in the filter field at the top of the Palette view to see matching controls.
 - Expand the folders in the Palette view to display the available controls.
4. Drag the control to the desired location in the design canvas.
 - If the control that you selected is not permitted in the area where you want to place it, the cursor displays a “no action” icon (a circle with a diagonal line through it).
 - If the cursor is over a permitted area for the control, the cursor displays a small gray square with a + sign, as well as a pop-up tooltip stating where the control will be positioned.
 - Some controls have facets into which you can drop other controls, and some controls require child controls to function properly.
5. In the Properties view, complete the **General**, **Display**, and other related properties.
6. Click **Preview** to see the control added to the view.
7. Save the view.

Preparing a Web Application for Localization

You can prepare a Composite Application Framework web application for localization by exporting the user interface strings into a message resource bundle.

The localization tool automatically replaces the strings with the appropriate binding expressions. For each control, the localization tool uses the following format for each binding expression, where the project identifies the name of the project, `viewFilename` is the name of the VDL file for the view, and `stringID` is derived from the first four words of the original string.

```
#{project.applicationResources["g11n.l10ndefault.viewFilename.stringID"]}
```

The externalized strings are included as resources to the message bundle associated with the page bean.

If you need to recover the original VDL file, you can use the Eclipse Local History feature. To access **Local History**, right-click the VDL file and click **Replace with > Local History**.

➤ To prepare a web application strings for localization

1. In the Navigation view, open a web application project and expand the **WebContent** folder.
2. Right-click the default view and click **CAF Tools > Externalize Strings from a view**.
3. In the **Externalize Strings** wizard, accept the default prefix, or type a new prefix in the **Enter a common prefix for generated keys (optional)** text box.

The prefix automatically appears in the **Key** column.

(Optional) Clear the **Filter all existing ignored and externalized entries** check box to view ignored or externalized strings.

4. In the **Strings to Externalize** list box, select an entry in the list to change it. By default, each string is set to Externalize.

The following table lists the available options for string localization:

Column Display	Button Name	Description
<input checked="" type="checkbox"/>	Externalize	Marks the string for export into a separate file.
<input checked="" type="checkbox"/>	Ignore	Marks the string as one to be ignored for localization; the string should not be localized.
<input type="checkbox"/>	Internalize	Removes marks to ignore the string for localization.
Key value	Edit	Allows you to edit the string and key.
	Revert	Restores the original string.
Key	Rename keys	Inactive.
Resource Bundle		Click the column entry to select the resource bundle that you want to export.

5. Click **Next**.
6. In the **Found Problems** list, review the list for any error entries, and click **Back** to resolve them.
7. Click **Next**, and then click **Finish**.

7 Getting Started with Portlet Application Development

■ About Portlet Application Projects	62
■ Creating a Portlet Application Project	62
■ Creating a Portlet	63
■ About Portlet Types	63
■ About Portlet Modes	64
■ Adding a View to a Portlet	65
■ Adding a Control to a Portlet View	66
■ Finding Information about CAF Controls	67
■ Adding a Portlet Application Project to My webMethods Server	68
■ Importing My webMethods Server Assets into a Portlet Application	68
■ Publishing a Portlet Application to My webMethods Server	69
■ Adding a Portlet to a My webMethods Server Page	69
■ Uninstalling Portlet Applications from My webMethods Server	70
■ Troubleshooting Portlet Applications	70
■ Repairing a Portlet Application Project	71
■ Opening the Portlet Application Configuration Editor	71

About Portlet Application Projects

A portlet application project is a container for managing one or more portlets. After creating the portlet application project, you can create and configure one or more portlets within it.

After creating a portlet, you can add one or more portlet views, and you can include a dynamic web application in a portlet view. You can view the user interface of each portlet view with the **Preview** tab of the editor, which provides a static HTML version of the view.

Creating a Portlet Application Project

Before you can create a Composite Application Framework portlet, you must first create a portlet application project.

For more information about creating a portlet, see [“Creating a Portlet” on page 63](#).

➤ To create a portlet application

1. In the UI Development perspective, click **File > New > Portlet Application Project**.
2. In the **New Portlet Application Project** wizard, in **Project Name**, type a name for the portlet application project.
3. In **Target Runtime** list, leave **My webMethods Server** as the default.
4. In **Dynamic web module version**, accept the default value.
5. In **Configuration**, use the default **CAF Portlet Application**.
6. (Optional) In **Working sets**, click **Add Project to working set** to use a subset of resources, and then click **Select** to select the working set resource for the project.
7. In **Initial Project Contents**, select one of the following in the **Project Templates** list:
 - **Empty Project** (default). The JSF project is created with a minimum of folders and resources. You will create your project structure and view files and other resources later with the New Portlet and New View wizards.
 - **OpenUI Shell**. This always creates a JSF project with Facelets support. The project is created without any portlets, but it does contain a page.xhtml page. For more information about creating an webMethods OpenUI Shell page template, see [“Creating an OpenUI Shell Page Project” on page 74](#).
 - **Search Application**. This creates a JSF project that contains two default portlets: a Search Bar portlet that enables the user to enter search criteria and save searches, and a Search Results portlet that displays the items returned by the search. Use the **Display Style** list to specify if you want table results, tree results, or a display that can be toggled from table to tree.

- Click **Next** to specify any custom settings you want to apply to the project, or click **Finish** to create the project with the remaining default settings.

Creating a Portlet

You must create a portlet in an existing portlet application project. A portlet is created with a single default view page. You can add more pages, or *views*, to the portlet. For more information, see [“Adding a View to a Portlet” on page 65](#).

➤ To create a portlet

- In the UI Development perspective, in the Navigation view of an opened portlet application project, select the portlet application, and right-click **New > Portlet**.
- In the New Portlet wizard, select the **Portlet Type** from the list. To develop your own portlet, select **Generic**. For more information, see [“About Portlet Types” on page 63](#).
- Click **Next**.
- On the Create Portlet page, type a name for the portlet in the **Portlet Name** field.
- Skip the **Expiration Cache** field. Composite Application Framework does not use this value.
- In **Supported Locales**, click **Add** to include other language codes, or skip if the default *en* is the only language this portlet supports.
- In **Portlet Modes**, select the modes the portlet is to support (**view** is preselected and cannot be changed). For more information, see [“About Portlet Modes” on page 64](#).
- (Optional) In **Portlet Info**, in **Short Title**, type an alternate title for the portlet.
- (Optional) In **Keywords**, type keywords, separated by commas, to enable users to search for the portlet.
- Click **Next** to specify any custom settings you want to apply to the portlet, or click **Finish** to create the portlet with the remaining default settings.

You view and work with the new portlet in the Solutions view, the Navigator view, or the Package Explorer view, depending on the project type.

About Portlet Types

The following table lists the portlet types, available for selection when creating a Composite Application Framework portlet application project:

Portlet Type	Description
Generic	A generic portlet. Suitable for creating a basic portlet that you can modify to your desired implementation.
Search Bar	A predefined portlet for use in developing a search bar portlet. Contains UI elements that enable the user to enter search criteria, set options, and save searches. You must provide wiring to connect this portlet to a Search Results portlet.
Search Results	A predefined portlet that provides search results. Contains UI elements that display the items returned by the search. You must provide wiring to connect this portlet to a Search Bar portlet. When you select this portlet type, use the Display Style list to select table results, tree results, or a display that can be toggled from table to tree.
Search Bar + Results	A predefined portlet that provides both the search bar and the search results, requiring no portlet-to-portlet wiring. When you select this portlet type, use the Display Style list to select table results, tree results, or a display that can be toggled from table to tree.
Task [...]	Predefined webMethods task portlets. These are generally selected when you are working with a task application project. For more information about task development, see <i>webMethods BPM Task Development Help</i> .

About Portlet Modes

Portlet modes are defined in the Java JSR-168 portlet standard to indicate the function a portlet is to perform. All portlets must operate in view mode. You can specify additional modes for a portlet as required. The following table lists the modes, available in Composite Application Framework:

Portlet Mode	Description
View	The user can view, navigate, or interact with the portlet. The view mode is required and is selected by default for all portlets. Composite Application Framework creates a View Declaration Language (VDL) file in the portlet with the name default.xhtml.
Edit	Creates an edit page that enables user customization or preference setting for the portlet. When you select this mode, Composite Application Framework creates a VDL file in the portlet with the name edit.xhtml.
Help	Creates a help page for the portlet. When you select this mode, Composite Application Framework creates a VDL file in the portlet with the name help.xhtml.

Adding a View to a Portlet

When you create a portlet, it is created with a default view file. An edit or help view may be present if you specified other portlet modes. These views, or *pages*, are created as JSF View Declaration Language (VDL) files. You can drag a control from the Palette view onto the VDL file displayed in the editor. For more information, see [“Adding a Control to a Portlet View” on page 66](#).

You can add additional views to the portlet to extend the portlet functionality. You can use one of the available predefined view templates to create the view, or you can add an empty view using this procedure.

➤ To add a view to a portlet

1. In the UI Development perspective, in the Navigation view, open the portlet application project and expand the **WebContent** folder.
2. Select the portlet that will receive the new VDL file, and right-click **New > Portlet View**.
3. In the JSF Portlet View wizard, type a name for the VDL file in the **File name** field.

Note:

If you type the file name only, without a file name extension, the view file will be created in .xhtml format by default. If you type the file name with a .view file name extension, the view file will be created as a legacy page view.

4. (Optional) Click **Advanced** if you want to link to an existing file in your file system.
5. Click **Next**
6. In the **For Portlet** list, select the portlet where you want to create the view.
7. In **For Portlet Mode**, click the portlet mode you want to apply to this view. For more information, see [“About Portlet Modes” on page 64](#).
8. In the **Template** list, select a template to use for the VDL file. The following table lists the available templates:

Template	Description
Form	(Default) Creates a basic view with a Formatted Message control and a JSF command form. You can modify the view to your desired implementation.
Empty	Creates a basic view with no components other than a View Root control. You can modify the view to your desired implementation.

Template	Description
Search Bar	Creates a predefined view for use in developing a search bar portlet. Contains user interface elements that enable the user to enter search criteria, set options, and save searches.
Task Notification	Creates a task notification view. This is generally selected when you are working with a task application project. For more information, see the <i>webMethods BPM Task Development Help</i> .

- Click **Next**.
- In the **Managed Bean Scope** list, choose the scope for portlet view. For more information, see [“About Managed Beans” on page 19](#).
- Click **Next** to specify any custom settings you want to apply to the view, or click **Finish** to create the view with the remaining default settings.

The new view appears in the design canvas (a JSF graphical editor), and in the Solutions view, the Navigator view, or the Package Explorer view, depending on the project type.

Adding a Control to a Portlet View

When you drop a control onto the design canvas, the canvas provides visible references to indicate valid drop points. Some controls have facets into which you can drop other controls and some controls require parent or child controls to function properly.

Note:

The structure of the control folders in the Palette view is different for .view files and .xhtml files. However, the control functionality is the same.

➤ To add a control to a portlet view

- In the UI Development perspective, locate the portlet you want to work with in the Navigation view, and expand the **WebContent** folder.
- Double-click the view to which you want to add a control to open it in the editor, or drag the view to the editor to open it.
- In the Palette view, expand the nodes to display the available controls, or type in the filter box at the top to locate the control you want to work with.
- From the Palette view, drag a control to the design canvas, release the control in the Form command of the view.
 - As you move the control over the design canvas, observe that a vertical red bar appears in the editor when you pass over an allowed location.

- If the control that you selected is not permitted in the area where you want to place it, the cursor displays a “no action” icon (a circle with a diagonal line through it).
 - If the cursor is over a permitted area for the control, the cursor displays a small gray square with a + sign, as well as a pop-up tooltip stating where the control will be positioned.
 - Some controls have facets into which you can drop other controls, and some controls require child controls to function properly.
5. Select the newly added control in the view editor, and in the Properties view, enter values for the control's properties as required.
 6. Click the **Preview** tab to see your results.
 7. Save the view.

Finding Information about CAF Controls

You can find descriptions of the CAF controls that Composite Application Framework provides in *webMethods CAF Tag Library Reference*. The control descriptions are available as HTML-based View Declaration Language Documentation and are viewed with your web browser in a format similar to Javadoc.

When using *webMethods CAF Tag Library Reference*, you can change how the controls are displayed in the navigation area. In the upper left of the page, click either **By Tag Name** or **By Palette Location**.

- Click **By Tag Name** to display a flat list of all controls by their tag names, organized by tag library.

You might find this organization helpful when you are developing an XHTML-based view for a JSF Facelets application.

- Click **By Palette Location** to organize the controls in the same way the controls are organized in the Palette view.

You might find this organization helpful if you are developing your JSF application with .view files.

In both cases, you can determine how to find a specific control in the Palette view by clicking the control tag name to view its description. In the **Converter Information** section, the location is listed in the **Palette Location** field.

In addition to *webMethods CAF Tag Library Reference*, the following sections provide information about CAF controls:

- [“User Interface Controls Concepts” on page 243](#)
- [“Understanding the Client-side Model” on page 263](#)
- [“Using Converters and Validators” on page 275](#)

Adding a Portlet Application Project to My webMethods Server

During portlet application development, you can test the individual portlet application views by using the **Preview** tab in the view editor.

When your portlet application is ready to test on a runtime server, you can add the portlet project to My webMethods Server, and then publish your Composite Application Framework (CAF) portlet application project to My webMethods Server.

The CAF portlet application project uses My webMethods Server as the default target runtime environment. CAF portlet application projects can be published only to My webMethods Server.

If you apply any changes to the portlet application project after adding the project to My webMethods Server, you must republish the project to the server to view the most recent updates. For more information about publishing to a server, see [“Publishing a Portlet Application to My webMethods Server” on page 69](#).

➤ To add a portlet application to My webMethods Server

1. In the Servers view, right-click an instance of My webMethods Server and then click **Add and Remove**.

Note:

By default, the check box **If server is started, publish changes immediately** is selected in the Add and Remove dialog box. If you do not want to publish the project immediately, be sure to clear this check box before you click **Finish**.

2. To add the portlet application to the server, move an available project to the **Configured** box and click **Finish**.

If My webMethods Server is running and the check box described above is selected, Composite Application Framework immediately publishes the portlet application.

Importing My webMethods Server Assets into a Portlet Application

You can import portlet assets from an instance of My webMethods Server to a Composite Application Framework (CAF) portlet application project as an XML import file.

➤ To import from My webMethods Server to a portlet application project

1. In the UI Development perspective, in the MWS Admin view, expand the tree node that contains the My webMethods Server runtime asset type that you want to import, and then right-click **Extract Asset into Project**.
2. In the **Extract Asset into Project** wizard, select the project you want to import the asset into. Click **New** to extract the asset into a new, empty My webMethods Server content project.

3. Click **Finish**.



A folder containing an `xmlImport.xml` file and other files representing the asset is added to the `WEB-INF\assets` folder in the project. An `assets.txt` file is created in the `WEB-INF` folder as well.

Publishing a Portlet Application to My webMethods Server

When you publish a portlet application to an instance of My webMethods Server, all portlets in the portlet application are available for use on the server. When you make changes to a portlet in Composite Application Framework, you must republish the portlet application to the server.

For information about adding portlets to server pages, see *Administering My webMethods Server*. If you already have the portlet displayed on a server page, refresh the page to see changes to the portlet.

➤ To publish a portlet application to My webMethods Server


1. In the UI Development perspective, click the Servers view.
2. If the server instance is not running, select the My webMethods Server instance and click the .
3. After My webMethods Server local host is started, click .

Adding a Portlet to a My webMethods Server Page

After you publish a portlet to My webMethods Server, you can add the portlet to a folder or page on My webMethods Server using the MWS Admin view by dragging the portlet directly to the location where you want it to reside on the server.

After you have published a portlet application to My webMethods Server, you can log in to My webMethods Server as a system administrator user and add any published portlets to the page.

➤ To add a portlet from a portlet application to a page on My webMethods Server

1. Log into My webMethods Server as system administrator and either search for the project, or if you know its location, browse to the page.
2. In the page title bar, click , and then click **Edit Page**.
3. The **Tools** tab on the left lists the portlets you can add to a page, including your published portlets.
4. Select the **View As Expert** check box to view the column, row, and portlet borders to help with placement.

5. In the **Tools** tab, select the portlet you want to add and drag it on to the page.

If the **View As Expert** check box is selected, a red box appears beneath the cursor location whenever the cursor is over a page column, indicating where the portlet will be positioned if you release the mouse button.

6. Click **Save**.

For more information about working with portlets in My webMethods Server pages, see *Administering My webMethods Server*.

Uninstalling Portlet Applications from My webMethods Server

As you are developing portlet applications you may find that you want to uninstall portlets or portlet applications that you have published to My webMethods Server.


➤ To uninstall a portlet or portlet application using the Install Administration page

1. Log into My webMethods Server as system administrator. You can do this from with the MWS Admin view by double-clicking the server instance name. A browser session opens the My webMethods Server login page in the editor view.
2. Browse to **Administration > Configuration > Install Administration**.
3. In the tree, select the portlet applications you want to uninstall. Your portlet applications appear as .war files under the **Independent Components** node.
4. Click **Uninstall Selected**. In the confirm dialog box, click **Uninstall**.

Important:

You must also remove the portlet application project from the server instance in the Servers view. If you do not, the project will be republished to the server with the next publish action. For additional information, see [“Adding a Portlet Application Project to My webMethods Server” on page 68](#).

Troubleshooting Portlet Applications

In the Solutions or Project Explorer view, the  overlay displayed next to a node indicates an error needs your attention. For example, the project might contain a template that requires you to specify values for certain views.

Tip:

You can find additional information about errors by clicking the Problems view in the UI Development perspective. You can also find information in the Error Log view, accessible by clicking **Window > Show View > Other > General > Error Log**. For more information about working with the Problems view and Error Log view, see the *Eclipse Workbench User Guide* in the Designer online help.

To address errors in your project, see [“Repairing a Portlet Application Project” on page 71](#) for more information.

Errors caused by the Import Wizard

If you did not use the Software AG Designer import wizard to bring the application project into the workspace, you might need to repair the project environment to comply with the Designer workspace requirements. For example, the following import methods can result in errors in a project:

- Importing projects using **File > Import > General > Existing Projects into Workspace**.
- Importing projects using **File > Import > CVS > Existing Projects from CVS**

Always use the Software AG Designer import wizard to import the project:

- **File > Import > Software AG > Existing CAF Projects into Workspace**

The Software AG Designer import wizard automatically repairs the project while the other import wizards do not.

Errors caused by the name of My webMethods Server run time

If the My webMethods Server run time is manually removed and added again with a different name, this might require you to update to the existing projects in the workspace to use the new My webMethods Server.

Repairing a Portlet Application Project

In general, if there are build errors for the project or validation errors in the VDL files (that is, the .view and/or .xhtml files) in the project, the first action to take is to run the **Repair CAF Project** action.

➤ To repair a portlet application project

1. Do either of the following:

- In the Navigator or Project Explorer view, right-click the portlet application and click **CAF Tools > Repair CAF Project**.
- In the **Solutions** view, right-click the portlet application and click **Repair Project**.

Opening the Portlet Application Configuration Editor

The Portlet Application Configuration editor manages the portlet descriptors for all the portlets contained in a portlet application project. You can create portlets and various objects used by portlets and the portlet application.

When you create a portlet application, the Portlet Application Configuration editor is automatically displayed. If you close the editor, you can open it again by doing the following:

➤ **To open the Portlet Application Configuration editor**

1. Do either of the following:

- In the Solutions view, double-click any portlet within the portlet application.
- In the Project Explorer view or Navigator view:
 1. Locate the portlet application in the tree view and expand it to expose the portlet.xml file:
portlet_application/WebContent/WEB-INF/portlet.xml
 2. Double-click the portlet.xml file.

8 Getting Started with webMethods OpenUI Shell

Page Development

■ About webMethods OpenUI Shell Page Templates	74
■ Creating an OpenUI Shell Page Project	74
■ Modifying an OpenUI Shell Custom Application Page	75
■ Modifying an OpenUI Shell Login Page	77
■ Modifying an OpenUI Shell Maintenance Page	78
■ Adding an OpenUI Shell Page to an Existing Project	80
■ About Creating Custom Templates with webMethods OpenUI	80
■ Creating a Custom OpenUI Page Template	80
■ Creating a Custom OpenUI Project Template	82
■ Editing a Custom Page or Project Template	83
■ Exporting a Custom Page or Project Template	83
■ Importing a Custom Page or Project Template in Designer	84
■ Deleting a Custom Page or Project Template	84
■ Default webMethods OpenUI Page Templates	84
■ Variable Conversion When Applying a Page Template	85
■ Handling of Existing Files When Applying a Page Template	87

About webMethods OpenUI Shell Page Templates

webMethods OpenUI Shell is an implementation of My webMethods Server shells, based on JavaServer Faces (JSF) Facelets, that uses a Facelets page template for creating and rendering an xhtml page. You create an OpenUI Shell page project that contains an xhtml page in Software AG Designer. You can use either pre-defined project and page templates to create your OpenUI Shell application, or custom project and page templates that you define on the Preferences > Software AG > UI Development > Templates page in Designer.

After you create your OpenUI Shell page project, you can use portlets along with CAF JSF tags to render the body of the page. You can also add more xhtml pages to your project.

For more information about the OpenUI Shell functionality, see [“About webMethods OpenUI Shell” on page 44](#). For more information about OpenUI, see [“Developing webMethods OpenUI Applications” on page 43](#). For more information about CAF JSF tags, see *webMethods CAF Tag Library Reference*.

Creating an OpenUI Shell Page Project

Use the following procedure to create an OpenUI Shell page project. The OpenUI Shell page project contains an xhtml page template that can be further modified. You deploy the project to My webMethods Server.

For more information about how to deploy a project to My webMethods Server, see [“Publishing an Application to My webMethods Server” on page 162](#).

➤ To create an OpenUI Shell page project

1. In the UI Development perspective, go to **File > New > Portlet Application Project**.
2. In the **Project name** field, type a name for the project.
3. In the **Target runtime** field, select **My webMethods Server**.
4. In the **Dynamic web module version** field, accept the default value.
5. In the **Configuration** field, accept the default value **CAF Portlet Application**.
6. In the **Project Template** field, select one of the following options:
 - **OpenUI Shell** - uses the pre-defined OpenUI Shell templates to create the project.
 - **Custom project template** - uses a custom project template defined and enabled on the Preferences > Software AG > UI Development > Templates page to create the project.
7. In the **Initial Content** field, select the type of page that you want to create:
 - **<custom>**

Note:

Selecting the **<custom>** option automatically opens the Page Options wizard where you can specify options for the initial content of the page.

- **Custom Application Page With Default Framing**
- **Custom Application Page With Empty Framing**
- **Login Page**
- **Maintenance Page**

8. Do one of the following:

- To create the selected page with default settings, click **Finish**.
- To further modify the selected page, click **Modify**.
 - To modify a custom application page with default or empty framing, see [“Modifying an OpenUI Shell Custom Application Page” on page 75](#).
 - To modify a login page, see [“Modifying an OpenUI Shell Login Page” on page 77](#).
 - To modify a maintenance page, see [“Modifying an OpenUI Shell Maintenance Page” on page 78](#).

Designer generates an OpenUI Shell page project that includes the following files in the WebContent folder:

- *fileName.xhtml* - A JSF Facelets template that is responsible for rendering the shell.
- WEB-INF/shells/*fileName_xhtml.xml* - An xmlImport file that contains instructions for registering the OpenUI Shell page template with My webMethods Server during deployment of the OpenUI project. The file contains instructions for creating or updating My webMethods Server resources, and for creating a shell rule.
- WEB-INF/xmlImport.xml - An xmlImport file that loads the WEB-INF/shells/*fileName_xhtml.xml* file during deployment of the OpenUI project.

Modifying an OpenUI Shell Custom Application Page

Use the following procedure to modify an OpenUI Shell custom application page.

➤ To modify a custom application page with default framing or empty framing

1. Create an OpenUI Shell page project. For more information about how to create a project, see [“Creating an OpenUI Shell Page Project” on page 74](#).
2. In the Page Options wizard, specify values for the following fields:

Field	Description and Value
Page purpose	The purpose of the page. Select Custom application page .
Filename	The file name of the xhtml page that Designer generates. Specify a simple file name, for example <code>WebContent/page1.xhtml</code> , or a unique path in the project, for example <code>WebContent/folder1/page.xhtml</code> . The default value is <code>WebContent/page.xhtml</code> .
Initial content	<p>The initial content of the page. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Default custom shell page templates, if the default templates are enabled on the Preferences > Software AG > UI Development > Templates page. Values are: <ul style="list-style-type: none"> ■ shell_default. Creates a page that uses the default My webMethods Server shell sections: header, footer, leftnav, and rightnav. ■ shell_empty. Creates an empty page. ■ Any user-created custom page templates with a shell context that are enabled on the Templates page. ■ Pre-defined shell page options. Values are: <ul style="list-style-type: none"> ■ Page with default framing. Creates a page that uses the default My webMethods Server shell sections: header, footer, leftnav, and rightnav. ■ Page with empty framing. Creates an empty page.
Parent shell alias	<p>The alias of the parent shell to use for creating your OpenUI shell. Select one of the following shells, or type the alias of another existing shell:</p> <ul style="list-style-type: none"> ■ shell.blank ■ shell.noodle - This is the default value. ■ shell.noodle.guest <p>The OpenUI shell initially inherits all properties of the parent shell. You can replace any inherited property with a new, custom property.</p>
Associated folder in MWS	<p>The root My webMethods Server folder that your OpenUI shell should be mapped to. Select one of the following options and specify values for the fields as required:</p> <ul style="list-style-type: none"> ■ Create a new folder in MWS - Creates a new folder to contain the pages that your shell will render. <ul style="list-style-type: none"> ■ Parent folder alias - the alias of the parent folder in the My webMethods Server content repository in which the new folder

Field	Description and Value
	<p>will be created. The default parent folder alias is <code>folder.custom.apps</code>. You can specify any custom alias for the parent folder.</p> <ul style="list-style-type: none"> ■ New folder name - The name of the new folder. By default, the folder name is the same as the OpenUI Shell page project name. <div> <p>Note: If you specify a folder name that is different from the OpenUI Shell project name, make sure that the name is unique in the system.</p> </div> <ul style="list-style-type: none"> ■ Create a new workspace - Specify whether to create a new workspace for the folder. ■ Workspace alias - the alias of the new workspace. By default the alias is the same as the OpenUI Shell project name. <div> <p>Note: If you specify a workspace name that is different from the OpenUI Shell project name, make sure that the name is unique in the system.</p> </div> <ul style="list-style-type: none"> ■ Use an existing folder - Uses an existing folder to contain the pages that the OpenUI shell will render. ■ Existing folder alias - The alias of the existing folder with which to associate the shell.
	<p>Note: If you do not specify a value for an option, Designer creates the page with the default value for the option.</p>

Modifying an OpenUI Shell Login Page

Use the following procedure to modify an OpenUI Shell login page.

➤ To modify a login page

1. Create an OpenUI Shell page project. For more information about how to create a project, see [“Creating an OpenUI Shell Page Project” on page 74](#).
2. In the Page Options wizard, specify values for the following fields:

Field	Description and Value
Page purpose	The purpose of the page. Select Login page .
Filename	The file name of the xhtml page that Designer generates. Specify a simple file name, for example <code>WebContent/login1.xhtml</code> , or a unique path in the project, for example <code>WebContent/folder1/login.xhtml</code> . The default value is <code>WebContent/login.xhtml</code> .
Initial content	<p>The initial content of the page. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Default custom login page templates, if the default templates are enabled on the Preferences > Software AG > UI Development > Templates page. Values are: <ul style="list-style-type: none"> ■ login_inline ■ login_default ■ Any user-created custom page templates with a login context that are enabled on the Templates page. ■ Pre-defined shell page options. Values are: <ul style="list-style-type: none"> ■ Simple login page <p>Use current resource - Specifies whether to render the current resource using the default renderer. The option is selected by default. If you clear the option, Designer creates a simple page including a form, username input, password input, and submit button.</p>
Parent shell alias	<p>The alias of the parent shell to use for creating your OpenUI shell. Select one of the following shells, or type the alias of another existing shell:</p> <ul style="list-style-type: none"> ■ shell.blank ■ shell.noodle ■ shell.noodle.guest - This is the default value. <p>The OpenUI shell initially inherits all properties of the parent shell. You can replace any inherited property with a new, custom property.</p>

Note:

If you do not specify a value for an option, Designer creates the page with the default value for the option.

Modifying an OpenUI Shell Maintenance Page

Use the following procedure to modify an OpenUI Shell maintenance page.

➤ **To modify a maintenance page**

1. Create an OpenUI Shell page project. For more information about how to create a project, see [“Creating an OpenUI Shell Page Project” on page 74](#).
2. In the Page Options wizard, specify values for the following fields:

Field	Description and Value
Page purpose	The purpose of the page. Select Maintenance page .
Filename	The file name of the xhtml page that Designer generates. Specify a simple file name, for example <code>WebContent/maintenance1.xhtml</code> , or a unique path in the project, for example <code>WebContent/folder1/maintenance.xhtml</code> . The default value is <code>WebContent/maintenance.xhtml</code> .
Initial content	<p>The initial content of the page. Select one of the following options:</p> <ul style="list-style-type: none"> ■ Default custom maintenance page templates, if the default templates are enabled on the Preferences > Software AG > UI Development > Templates page. Values are: <ul style="list-style-type: none"> ■ maintenance_default ■ maintenance_inline ■ Any user-created custom page templates with a maintenance context that are enabled on the Templates page. ■ Pre-defined shell page options. Values are: <ul style="list-style-type: none"> ■ Simple maintenance page
Parent shell alias	<p>The alias of the parent shell to use for creating your OpenUI shell. Select one of the following shells, or type the alias of another existing shell:</p> <ul style="list-style-type: none"> ■ shell.blank ■ shell.noodle ■ shell.noodle.guest - This is the default value. <p>The OpenUI shell initially inherits all properties of the parent shell. You can replace any inherited property with a new, custom property.</p>

Note:

If you do not specify a value for an option, Designer creates the page with the default value for the option.

Adding an OpenUI Shell Page to an Existing Project

Use the following procedure to add a page to an existing OpenUI Shell page project.

➤ To add an OpenUI Shell page to an existing project

1. Navigate to the project in which you want to create the OpenUI Shell page and click **File > New > OpenUI Shell Page**.
2. In the Page Options wizard, select the project in which you want to create the page and specify values for the initial content options as required, depending on the type of page that you select.

For information about the fields and values to specify, see:

- [“Modifying an OpenUI Shell Custom Application Page” on page 75](#)
- [“Modifying an OpenUI Shell Login Page” on page 77](#)
- [“Modifying an OpenUI Shell Maintenance Page” on page 78](#)

If you do not specify a value for an option, Designer creates the page with the default value for the option.

3. Click **Finish**.

About Creating Custom Templates with webMethods OpenUI

webMethods OpenUI enables you to create custom page and project templates, and to share them with the technical community. After you create a custom page or project template, you can export it in a zip file, so that another developer can import the file and re-use or modify the template.

When creating an OpenUI custom page template, you can use any third-party libraries and frameworks, such as HTML, JavaScript, AngularJS, and jQuery, to modify the look and feel, and content of the page.

You can create, edit, export, import, and delete custom page and project templates on the Preferences > Software AG > UI Development > Templates page in Designer. After you create a custom page template, you can use it in a custom project template or directly in an OpenUI Shell page project.

In addition, the Templates page contains a set of default login, maintenance, and shell templates that you can edit and share, or use in your projects.

Creating a Custom OpenUI Page Template

Use the following procedure to create a custom page template.

➤ To create a custom page template

1. In Designer, go to **Window > Preferences**.
2. Expand the **Software AG** node and then expand the **UI Development** node.
3. Click **Templates**.
4. On the Templates page, click **New Page**.
5. Specify values for the following fields:

Field	Description
Name	Required. The name of the page template.
Context	Required. The purpose of the page template. Values are: <ul style="list-style-type: none"> ■ Shell ■ Login ■ Maintenance
Description	Optional. A description of the page template.
Resources	Optional. The resources that the page template requires. <ul style="list-style-type: none"> ■ Click Link to connect to the already existing static resources in the Resources folder. ■ Click Import to import a folder from your local system into the Resources folder. Select one or more resources and click OK .
Pattern	Required. The definition of the page template. Enter your custom code to modify the content of the page. <div> <p>Note: Although you can save the template without specifying anything in the field, leaving the field empty creates an empty file when you use the template.</p> </div>

6. Click **OK**.

After you create the custom page template, when you create a new OpenUI Shell page project, you can select the page template in the **Initial Content** field of the Page Options wizard.

Note:

If you do not want the template to show as an **Initial Content** option in the Page Options wizard, clear the check box next to the template on the Templates page.

For more information about creating an OpenUI Shell project, see [“Creating an OpenUI Shell Page Project” on page 74](#).

Creating a Custom OpenUI Project Template

Use the following procedure to create a custom project template.

➤ To create a custom project template

1. In Designer, go to **Window > Preferences**.
2. Expand the **Software AG** node and then expand the **UI Development** node.
3. Click **Templates**.
4. On the Templates page, click **New Project**.
5. Specify values for the following fields:

Field	Description
Name	Required. The name of the project template.
Description	Optional. A description of the project template.
Resources	<p>Optional. The resources that the project template requires. Do one of the following:</p> <ul style="list-style-type: none">■ Click Link to connect to the already existing static resources in the Resources folder.■ Click Import to import a folder from your local system into the Resources folder. <p>Select one or more resources and click OK.</p>
Page	<p>Optional. The pages that the project template contains.</p> <p>Click Add to add an already existing page template. The page template is available in the Initial Content field of the Page Options wizard. For more information about the fields and values to specify in the Page Options wizard, depending on the purpose of the page, see:</p> <ul style="list-style-type: none">■ “Modifying an OpenUI Shell Custom Application Page” on page 75■ “Modifying an OpenUI Shell Login Page” on page 77■ “Modifying an OpenUI Shell Maintenance Page” on page 78

6. Click **OK**.

After you create the custom project template, when you create an OpenUI Shell project, you can select the template in the **Project Template** field of the New Portlet Application Project wizard.

Note:

If you do not want the template to show as a **Project Template** option in the New Portlet Application Project wizard, clear the check box next to the template on the Templates page.

For more information about creating an OpenUI Shell project, see [“Creating an OpenUI Shell Page Project” on page 74](#).

Editing a Custom Page or Project Template

Use the following procedure to edit a custom page or project template.

➤ **To edit a page or project template**

1. In Designer, go to **Window > Preferences**.
2. Expand the **Software AG** node and then expand the **UI Development** node.
3. Click **Templates**.
4. Select the page or project template that you want to modify and click **Edit**.
5. Specify new options for the page or project template as needed.
6. Click **OK**.

Exporting a Custom Page or Project Template

Use the following procedure to export a single page or project template, or multiple page and project templates into a zip file. You can export any combination of page and project templates.

Click **Export All** to export all page and project templates into a single zip file, regardless of whether they are selected or not.

➤ **To export a page or project template**

1. In Designer, go to **Window > Preferences**.
2. Expand the **Software AG** node and then expand the **UI Development** node.
3. Click **Templates**.
4. Select the page or project template(s) that you want to export and click **Export**.

5. Specify the output file.
6. Click **OK**.

Designer exports the selected templates into a zip file and saves the file in your file system. The zip file contains the following files and folders:

- templates.xml - an xml file that contains definitions of the templates.
- Templates - a folder that contains the exported templates.
- Resources - a folder that contains the resources linked to the exported templates.

Importing a Custom Page or Project Template in Designer

Use the following procedure to import a page or project template in Designer from a zip file.

➤ To import a page or project template

1. In Designer, go to **Window > Preferences**.
2. Expand the **Software AG** node and then expand the **UI Development** node.
3. Click **Templates > Import** and browse to the templates zip file to import.

Deleting a Custom Page or Project Template

Use the following procedure to delete one or more custom page or project templates.

➤ To delete page or project templates

1. In Designer, go to **Window > Preferences**.
2. Expand the **Software AG** node and then expand the **UI Development** node.
3. Click **Templates**.
4. Select the page or project template(s) that you want to delete and click **Remove**.

Default webMethods OpenUI Page Templates

The Preferences > Software AG > UI Development > Templates page in Designer contains a set of default login, maintenance, and shell page templates that you can use in your OpenUI Shell project. You can edit, copy, share, or delete the content of a default page template and use it in your custom page template.

By default, the default page templates on the Templates page are available as initial content options when you modify an OpenUI Shell page project.

Note:

If you do not want a template to show as an initial content option, clear the check box next to the template on the Templates page.

The following table lists the default page templates, available on the **Templates** page.

Page Template	Description
login_default	A login page with shell section framing and a login portlet.
login_inline	A login page with shell section framing and an inline login form.
maintenance_default	A maintenance page with shell section framing and a login portlet.
maintenance_inline	A maintenance page with shell section framing and an inline login form.
shell_default	A shell page with shell section framing.
shell_empty	A shell page without shell section framing.

For more information about modifying an OpenUI Shell page, see:

- [“Modifying an OpenUI Shell Custom Application Page” on page 75](#)
- [“Modifying an OpenUI Shell Login Page” on page 77](#)
- [“Modifying an OpenUI Shell Maintenance Page” on page 78](#)

Variable Conversion When Applying a Page Template

When you use a custom page template in an OpenUI Shell page project, Designer replaces certain variables if the variables are found in the template or in a text-based static resource linked to the template. The following tables show which variables are replaced under which conditions, and list the possible values of each variable.

Variable	Value
<code>\${project_name}</code>	The name of the target project converted to a valid identifier.
<code>\${project_type}</code>	Values are: <ul style="list-style-type: none"> ■ war ■ jar

Designer replaces the following variables while processing page template files or linked resources to a page template.

Variable	Value
<code>\${page_filename}</code>	The filename of the page.
<code>\${page_purpose}</code>	Values are: <ul style="list-style-type: none">■ shell■ login■ maintenance
<code>\${page_appFolderAlias}</code>	The alias of the custom application folder.
<code>\${page_applicationName}</code>	The name of the custom application.
<code>\${page_shellAlias}</code>	The alias of the page shell.
<code>\${page_shellName}</code>	The name of the page shell.
<code>\${page_webRelativeFileName}</code>	The page filename as a relative path.
<code>\${page_parentShellAlias}</code>	The alias of the parent shell.
<code>\${page_createFolder}</code>	Values are: <ul style="list-style-type: none">■ true■ false

Designer replaces the following variables when creating a new page.

Variable	Value
<code>\${page_newFolderParentAlias}</code>	The alias of the parent folder under which the new folder is created.
<code>\${page_newFolderName}</code>	The name of the new root folder created for the custom application.

Designer replaces the following variables when creating a new workspace.

Variable	Value
<code>\${page_createWebSpace}</code>	Values are: <ul style="list-style-type: none">■ true■ false
<code>\${page_newWebSpaceAlias}</code>	The workspace alias for the custom application.

Designer replaces the following variables when no new page is created.

Variable	Value
<code>\${page_existingFolderAlias}</code>	The alias of the existing thing with which the shell is associated.

Handling of Existing Files When Applying a Page Template

When you use a custom page template in an OpenUI Shell project that contains existing resources, Designer merges certain files. When particular files exist both in the applied template and at destination paths in the target project, Designer appends the content of the new file to the content of the existing file.

Note:

Merging new and existing files does not remove duplicate elements from the output file. You must ensure that the final file does not contain duplicate elements.

The following table lists the types of files that are merged in the target project and the destination path of each merged file of a particular type.

Type of File	Destination Path
xmlImport files	<ul style="list-style-type: none"> ■ WebContent/WEB-INF/before_upgrade/*/*.xml ■ WebContent/WEB-INF/dataSources.xml ■ WebContent/WEB-INF/xmlImport.xml ■ WebContent/WEB-INF/config/xmlImport.xml ■ WebContent/WEB-INF/xmlImport/*.xml ■ WebContent/WEB-INF/after_upgrade/*/*.xml
faces-config descriptor files	WebContent/WEB-INF/faces-config.xml
web descriptor files	WebContent/WEB-INF/web.xml
portlet descriptor files	<ul style="list-style-type: none"> ■ WebContent/WEB-INF/portlet.xml ■ WebContent/WEB-INF/wm-portlet.xml
web service connector clients descriptor files	WebContent/WEB-INF/wsclients.xml

For all other files that exist at the destination path, Designer ignores the files in the applied page template and uses the existing files in the project instead.

9 Working with Facelets

■ About Facelets in CAF	90
■ Migrating CAF Projects and .view Files to JSF Facelets Format	90
■ Working with the CAF Control Tag Library	93
■ Working with Page Navigation in JSF Facelets Applications	93
■ Embedding Portal Resources in a Page File	94
■ Example of Inserting a Form Into a Portlet	95
■ Java Annotations in CAF and webMethods OpenUI Applications	97

About Facelets in CAF

By default, Composite Application Framework (CAF) provides support for the JSF Facelets standard and the implementation of page views as .xhtml files.

You can create legacy view pages by specifying a .view file name extension when creating a view page.

Facelets provide a number of advantages to facilitate component reuse and ease of development, including:

- **Templating.** A Facelets file can be configured to reference an existing file that can be used as a template. The Facelets file can provide content for placeholders defined in the template file. The Facelets file, with its template file incorporated, can also be used as a template for other Facelets files, enabling you to create a hierarchy of templates.
- **Ability to use XHTML to define pages.** You can use the extended capabilities of XHTML to define your pages. In addition, using *.xhtml as a URL pattern provides convenience when calling pages.
- **Support for Facelets tag libraries.** All CAF controls are available as tag libraries for use in Facelets applications.

You can convert existing CAF portlet projects that use .view files to the JSF Facelets .xhtml format. You can also convert an individual .view file to an .xhtml file.

Migrating CAF Projects and .view Files to JSF Facelets Format

If you have existing Composite Application Framework (CAF) projects implemented with .view files, you can migrate those projects to JSF Facelet/XHTML support. You cannot change the project back to JSF .view file format after you migrate it. In addition, you can migrate individual .view files to .xhtml files.

For more information, see:

- [“Migrating a JSF .view File Project to JSF Facelets Format” on page 90](#)
- [“Migrating .view Files to JSF .xhtml Facelets Format” on page 91](#)
- [“About Migrating Managed Bean Declarations” on page 92](#)

Migrating a JSF .view File Project to JSF Facelets Format

You can migrate a JSF .view file project to JSF Facelets format. Migrating the project updates the project facets to ensure they are using the latest versions and converts all .view files in the project to .xhtml files.

➤ **To migrate a .view file project to JSF Facelets format**

Important:

Migration cannot be undone.

1. Select the project or projects you want to migrate in the Project Explorer view or Navigator view.
2. Right-click the project and click **CAF Tools > Migrate to JSF 2.x**. In the migration wizard, do the following:
 - a. If one or more of the selected projects is not using the latest supported version of the project facet, the Migrate projects panel appears. All projects requiring a facet update are selected by default. Accept the default selections, or clear a check box to omit a project from being migrated. Click **Finish** to migrate the selected projects, all .view files, and any managed bean declarations, or click **Next** to refine your migration selections.
 - b. In the Migrate .view files panel, all .view files are displayed and selected by default. This panel appears first if all projects have up-to-date facets. Accept the default selections, or clear the check box for any .view file you do not want to migrate. Click **Finish** to migrate the .view files and any managed bean declarations, or click **Next** to refine your migration selections.

Note:

If any of the .view files that are to be migrated have unsaved changes in an active editor, the wizard prompts you to select those files for saving before converting them. If you do not select the files for saving before conversion, the unsaved changes will not be included in the conversion.

- c. If one or more of the projects contain bean declarations in the WEB-INF/faces-config.xml file, the declarations available for migration to Java annotations are displayed and selected. For more information about bean annotations, see [“About Migrating Managed Bean Declarations” on page 92](#). Clear the check box for any declaration you do not want to migrate, then click **Finish** to complete the migration.

Migrating .view Files to JSF .xhtml Facelets Format

You can migrate individual .view file within a project into .xhtml files. This is convenient when you have added a JSF .view portlet to a JSF Facelets project and you want to migrate the portlet views individually. While it is permissible to mix both .view files and .xhtml files in a JSF Facelets application, you are advised to migrate the added .view files.

Important:

Be sure to migrate the entire application project *before* you migrate any individual .view files. For more information about migrating a .view project, see [“Migrating a JSF .view File Project to JSF Facelets Format” on page 90](#). Migrating .view files in a non-migrated project will result in missing libraries and runtime errors.

➤ To migrate .view files to JSF .xhtml Facelets format

Important:

Migration of .view files cannot be undone.

1. Locate the project you want to work with in the Project Explorer view or Navigator view.
2. Do one of the following:
 - To migrate an individual .view, right-click the .view file and click **CAF Tools > Migrate to JSF 2.x**.
 - To migrate all .view files in the portlet, right-click the portlet folder and click **CAF Tools > Migrate to JSF 2.x**.
 - To migrate all .view files in the application, right-click the WebContent folder and click **CAF Tools > Migrate to JSF 2.x**.
3. On the Migrate .view panel, the .view files available for migration are displayed and selected. Clear the check box for any .view file you do not want to migrate, Click **Finish** to migrate the .view files and any managed bean declarations, or click **Next** to refine your migration selections.

Note:

If any of the .view files that are to be migrated have unsaved changes in an active editor, the wizard prompts you to select those files for saving before converting them. If you do not select the files for saving before conversion, the unsaved changes will not be included in the conversion.

4. On the Migrate Managed Bean Declarations panel, the declarations available for migration to Java annotations are displayed and selected. Clear the check box for any declaration you do not want to migrate. For more information, see [“About Migrating Managed Bean Declarations” on page 92](#).
5. Click **Finish**.

About Migrating Managed Bean Declarations

When you migrate your project and .view files to Facelets format, qualified managed bean declarations are migrated by default from the WEB-INF/faces-config.xml file to Java annotations.

To qualify for migration, a managed bean declared in faces-config.xml must be a *source type*, meaning that the type is defined in a .java file from the project. *Binary types*, such as those that come from an external jar file, cannot be declared using annotations because the source code is not available to modify.

During the migration, qualified managed beans present in the WEB-INF/faces-config.xml file are displayed in the Migrate Managed Bean Declarations page of the migration wizard, which enables you to view the declarations, and to select any declarations you do not want to migrate.

The migration adds Java annotations for the selected declarations to the source .java file and removes the corresponding <managed-bean> elements from the faces-config.xml file.

Note:

Software AG Designer enables you to declare managed beans using annotations by default. If you want to continue declaring managed beans in the `faces-config.xml` file, see [“Disabling and Enabling Declaration of Managed Beans by Annotation”](#) on page 98.

Working with the CAF Control Tag Library

Composite Application Framework (CAF) version 9.6 and later contains a JavaServer Faces tag library that includes all CAF controls available in the Palette view. When you drag a CAF control to a View Declaration File (VDL) file implemented in `.xhtml` format, the control is added as part of the tag library. For example:

```
<caf_h:inputText id="htmlInputText" width="input20"/>
```

When you convert a JSF project with `.view` files, or an individual `.view` file, to JSF Facelets format, as described in [“Migrating CAF Projects and `.view` Files to JSF Facelets Format”](#) on page 90, the CAF controls contained in the `.view` files are migrated to CAF tag library controls. There is no change in control functionality.

For specific information about the controls in the CAF control tag library, see the VDL doc pages in *webMethods CAF Tag Library Reference*. For more information, see [“Finding Information about CAF Controls”](#) on page 67.

Note:

The structure of the control folders in the Palette view is different for `.view` files and `.xhtml` files. However, the control functionality is the same.

Working with Page Navigation in JSF Facelets Applications

JSF Facelets supports an *implicit navigation* that reduces navigation-related code required in the `faces-config.xml` file. With implicit navigation, you can omit explicit navigation-rule elements in the `faces-config` file and instead have your action handler method return one of the following values:

- To stay on the same page, return `null`.
- To switch to a different view, return the `viewId` of the target view.

The navigation handler first checks the navigation rules in the `faces-config.xml` for a matching navigation case. However, if a matching navigation case is not found, the navigation handler checks to see whether the action outcome is `null` or a value that corresponds to a `viewId`.

For example, the following action handler methods demonstrate how an action handler can take advantage of the implicit navigation:

```
public String doActionThatStaysOnSamePage() {
    //TODO: do your action logic here

    //return null to stay on the same page
    return null;
}
```

```
public String doActionThatSwitchesToPage2() {
    //TODO: do your action logic here

    //return the viewId of the page to switch to
    return "/yourportlet/page2.view";
}
```

For more information about JSF Facelet default navigation handling, see the JSF Facelet standard.

Note:

If you are migrating a JSF .view file project to JSF Facelet format and you have action handlers that return `OUTCOME_OK` or `OUTCOME_ERROR`, you should update the logic. For example, you might want the action handlers to return `null` instead.

Embedding Portal Resources in a Page File

JSF Facelets enables you to use portal resources as template files that can be shared with page files. You do this by adding the `<mws ui:resource>` element to the .xhtml file representing the page at the location in the file.

The `mws_ui:resource` tag is part of the CAF JSF tag library. The tag inserts a portal resource into an .xhtml page.

For information about the `mws_ui:` tags available in the CAF tag library, see the `mws_ui` section of the VDLdoc pages in the *webMethods CAF Tag Library Reference*.

Example: Render the head shell section of the current shell.

```
<mws_ui:resource renderer="shell">
<f:attribute name="shell.subrenderer.layout.name" value="head"/>
</mws_ui:resource>
```

Example: Render a specific portlet (by name) from the head shell section of the current shell.

```
<mws_ui:resource uri='{mwsShell.shellResource("head", "Global Links")}'
renderer="tabularasa" />
```

Example: Render a specific portlet (by alias).

```
<mws_ui:resource uri='portlet.alias.here' renderer="tabularasa" />
```

Example: Render the current resource with the default renderer.

```
<mws_ui:resource />
```

Example: Render some other resource (by alias) with a specific renderer.

```
<mws_ui:resource uri="folder.public" renderer="details"/>
```

For additional examples, see [“Example of Inserting a Form Into a Portlet” on page 95](#).

Example of Inserting a Form Into a Portlet

This topic provides an example of inserting an .xhtml form into an existing portlet using the Facelets templating tags. Typically, a user interface is assembled by rendering a set of portlets that are responsible for drawing a fragment of the larger page. In this case, you can implement a form to import into one of those portlets, providing you with a greater amount of flexibility in creating your portlet page.

This example demonstrates the following files:

form_template.xhtml

This file defines the tags for your form:

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:caf_h="http://webmethods.com/jsf/caf/html"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
  xmlns:h="http://xmlns.jcp.org/jsf/html">

  <caf_h:formattedMessages id="errorMessages"/>
  <caf_h:form id="form">

    <ui:remove> TODO: your generated form elements here </ui:remove>
  </caf_h:form>
</ui:composition>
```

portlet_example.xhtml

This file demonstrates inserting the form_template.xhtml file into the body of a portlet:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<caf_f:view xmlns="http://www.w3.org/1999/xhtml"
  xmlns:caf_f="http://webmethods.com/jsf/caf/core"
  xmlns:caf_h="http://webmethods.com/jsf/caf/html"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  pageManagedBean="ExampleDefaultviewView" preferencesBean="Example">
  <caf_f:design-time-attribute name="portlet" value="example"/>

  <ui:remove> Include the template here </ui:remove>
  <ui:include template="/form_template.xhtml"/>

</caf_f:view>
```

simple_standalone_page_example.xhtml

This file demonstrates inserting the form_template.xhtml file into the body of a simple standalone page without a My webMethods Serverheader/footer:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
```

```

    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:caf_f="http://webmethods.com/jsf/caf/core"
    xmlns:caf_h="http://webmethods.com/jsf/caf/html"
    xmlns:mws_ui="http://webmethods.com/jsf/mws_ui">
<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=8"/>
  <title>#{htmlHead.systemTitle}</title>
  <mws_ui:scripts />
</h:head>
<h:body>
  <mws_ui:pre_body />
  <ui:remove> Include the template here </ui:remove>
  <ui:include template="/form_template.xhtml"/>
</h:body>
</html>

```

mws_framed_standalone_page_example.xhtml

This file is similar to `simple_standalone_page_example.xhtml`, but with the My webMethods Server shell (header/footer/leftnav) rendered around the page content:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:caf="http://webmethods.com/jsf/caf/fn"
  xmlns:caf_f="http://webmethods.com/jsf/caf/core"
  xmlns:caf_h="http://webmethods.com/jsf/caf/html"
  xmlns:mws_ui="http://webmethods.com/jsf/mws_ui"
  >
<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <meta http-equiv="X-UA-Compatible" content="IE=8"/>
  <title>#{htmlHead.systemTitle}</title>
  <mws_ui:scripts />
</h:head>
<h:body>
  <mws_ui:pre_body />
  <div id="page">
    <div id="nonFooter">
      <div id="header">
        <ui:remove> shell header </ui:remove>
        <mws_ui:resource renderer="shell">
          <f:attribute name="shell.subrenderer.layout.name" value="head"/>
        </mws_ui:resource>
      </div>
      <div id="body">
        <div id="leftnav">
          <ui:remove> shell left </ui:remove>
          <mws_ui:resource renderer="shell">
            <f:attribute name="shell.subrenderer.layout.name" value="left"/>
          </mws_ui:resource>

```



```

        </div>

        <div id="rightnav">
            <ui:remove> shell right </ui:remove>
            <mws_ui:resource renderer="shell">
                <f:attribute name="shell.subrenderer.layout.name"
value="right"/>
            </mws_ui:resource>
        </div>

        <ui:remove> page content </ui:remove>
        <div id="portal-content" class="content">
            <ui:remove> wrap with container for caf async code </ui:remove>
            <div id="#{caf:xh(htmlHead.resourceURI)}"
class="page-portlet-container">
                <caf_h:formattedMessages id="shellErrors"/>

                <ui:remove> Include the template here </ui:remove>
                <ui:include template="/form_template.xhtml"/>

            </div>
        </div>
    </div>
</div>
<div id="footer">
    <ui:remove> shell foot </ui:remove>
    <mws_ui:resource renderer="shell">
        <f:attribute name="shell.subrenderer.layout.name" value="foot"/>
    </mws_ui:resource>
</div>
</div>
</h:body>
</html>

```

Java Annotations in CAF and webMethods OpenUI Applications

Java annotations can be used in Composite Application Framework (CAF) applications. By default, Software AG Designer enables you to declare managed beans using annotations. For more information, see [“Disabling and Enabling Declaration of Managed Beans by Annotation” on page 98](#).

The following annotations are supported:

- “Servlet Standard Annotations” on page 98
- “JSF Standard Managed Bean Annotations” on page 100
- “Bean Validation Standard Annotations” on page 101
- “CAF Extended Managed Bean Annotations” on page 103
- “CAF Client-Side Annotations” on page 104
- “CAF Portlet Annotations” on page 106

Disabling and Enabling Declaration of Managed Beans by Annotation

You can choose to declare new managed beans using Java annotations or by declaring them in the WEB-INF/faces-config.xml file. Both methods are valid, and this is generally a personal preference for the developer. Using Java annotations can make it easier to share managed beans between projects.

By default, Software AG Designer enables you to declare managed beans using annotations.

➤ To disable or enable declaration of managed beans with annotations

1. In Designer: **Window > Preferences**.
2. In the preferences tree, expand the **Software AG** node and click **UI Development**.
3. Do one of the following:
 - To disable bean declaration with annotations, clear the **Prefer Declaring Managed Beans via Java Annotations** check box.
 - To enable bean declaration with annotations, select the **Prefer Declaring Managed Beans via Java Annotations** check box. This check box is selected by default.
4. Click **Apply** to apply your changes and continue with other preference settings, or click **OK** to apply the changes and close the Preferences dialog box.

Servlet Standard Annotations

The JSR-315 Java Servlet standard defines a set of annotations that can be used to declare and configure servlets, filters, and listeners directly in a Java class. This is an alternative to declaring the same information in the WEB-INF/web.xml file of your Composite Application Framework (CAF) application. Pertinent code is highlighted in **bold**.

Example: Declare a Servlet

```
package caf.war.testapp1.test1;

import java.io.IOException;
import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class TestServlet1
 */
```

```

@WebServlet("/TestServlet1")
public class TestServlet1 extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public TestServlet1() {
        super();
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        // TODO place your code here
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        // TODO place your code here
    }
}

```

Example: Declaring a Filter

```

package caf.war.testappl.test1;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

```

```

import javax.servlet.annotation.WebFilter;

```

```

/**
 * Servlet Filter implementation class TestFilter1
 */

```

```

@WebFilter("/TestFilter1")
public class TestFilter1 implements Filter {

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletException {
        // TODO place your code here
    }

    /**
     * @see Filter#destroy()
     */
}

```

```
    */
    public void destroy() {
        // TODO place your code here
    }
    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
     */
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        // TODO place your code here

        // pass the request along the filter chain
        chain.doFilter(request, response);
    }
}
```

Example: Declaring a Servlet Context Listener

```
package caf.war.testapp1.test1;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import javax.servlet.annotation.WebListener;

/**
 * Application Lifecycle Listener implementation class
 * TestServletContextListener1
 */

@WebListener
public class TestServletContextListener1 implements ServletContextListener {

    /**
     * @see ServletContextListener#contextDestroyed(ServletContextEvent)
     */
    public void contextDestroyed(ServletContextEvent event){
        // TODO place your code here
    }

    /**
     * @see ServletContextListener#contextInitialized(ServletContextEvent)
     */
    public void contextInitialized(ServletContextEvent event){
        // TODO place your code here
    }
}
```

JSF Standard Managed Bean Annotations

The JSR-344 JavaServer Faces standard defines a set of annotations in the `javax.faces.bean` package that can be used to declare and configure managed beans directly in a Java class. This is an alternative to declaring the same information in the `WEB-INF/faces-config.xml` file of your Composite Application Framework (CAF) application. Pertinent code is highlighted in **bold**.

Example:

```

package caf.war.testapp1.test1;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

/**
 * Sample that demonstrates using the ManagedBean annotations
 * to declare a managed bean
 */

@ManagedBean(name = "TestBean")

@SessionScoped

public class TestBean extends com.webmethods.caf.faces.bean.BaseFacesSessionBean {

    /**
     * Override this method to release any resources associated with this
     * session.
     * Please note, the FacesContext is not valid for this function.
     */
    protected void release() {

    };

}

```

Bean Validation Standard Annotations

The JSR-349 Bean Validation standard defines a set of annotations that can be used to declare and configure validation constraints for fields in beans. This is an alternative to declaring the validators for each input control in the UI templates of your Composite Application Framework (CAF) application. Instead of defining the validators within the .view or .xhtml file, you instead define them on the field in the backing bean that the input controls are bound to. Pertinent code is highlighted in **bold**.

Example 1:

```

package caf.war.testapp1.test1;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

import javax.validation.constraints.Pattern;

import com.webmethods.caf.faces.annotations.ExpireWithPageFlow;
import com.webmethods.caf.faces.bean.BaseFacesBean;

@ManagedBean(name="inputBeanValidatorBean")
@SessionScoped

```

```
@ExpireWithPageFlow
public class InputBeanValidator extends BaseFacesBean {
    @Pattern(regexp = "\\d{3}-\\d{3}-\\d{4}")
    private String value = null;

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

Example 2:

```
@ManagedBean(name = "user")
@SessionScoped
public class UserBean implements Serializable {
    private static final long serialVersionUID = 1L;

    //Check the size of a string.
    @Size(min=3, message="Enter minimum 3 characters!")
    private String name;

    //Check that a value is at least or at most the given bound.
    @Min(10)
    @Max(1000)
    private double amount;

    //Check that a date is in the future.
    @Future
    private Date date;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

CAF Extended Managed Bean Annotations

These annotations provide additional CAF-specific metadata for managed beans defined using the JSF managed bean annotations:

- **@ExpireWithPageFlow**. If this annotation is found on a session-scoped managed bean class, then the managed bean will be stored in the page flow storage instead of as a session attribute, and will expire with the page flow storage object.
- **@DTManagedBean**. If this annotation is found on a managed bean class, then the additional design-time metadata is loaded for Designer to use in the Bindings view and other places.

Pertinent code is highlighted in **bold**.

Example:

```
package caf.war.testappl.test1;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import com.webmethods.caf.faces.annotations.ExpireWithPageFlow;
import com.webmethods.caf.faces.annotations.DTManagedBean;
import com.webmethods.caf.faces.annotations.BeanType;

/**
 * Sample demonstrating the usage of the extended CAF managed
 * bean annotations
 */
@ManagedBean(name = "Test1DefaultviewView")
@SessionScoped
```

@ExpireWithPageFlow

@DTManagedBean(displayName = "Test1/default", beanType = BeanType.PAGE)

```
public class Test1DefaultviewView extends
    com.webmethods.caf.faces.bean.BasePageBean {
    private static final long serialVersionUID = 1L;

    private static final String[][] INITIALIZE_PROPERTY_BINDINGS = new
        String[][] {
    };
    private transient caf.war.testappl.test1.Test1 test1 = null;

    /**
     * Initialize page
     */
    public String initialize() {
        try {
            resolveDataBinding(INITIALIZE_PROPERTY_BINDINGS, null, "initialize",
                true, false);
        } catch (Exception e) {
            error(e);
            log(e);
        }
        return null;
    }
}
```

```
}

    public caf.war.testapp1.test1.Test1 getTest1() {
        if (test1 == null) {
            test1 = (caf.war.testapp1.test1.Test1)resolveExpression("#{Test1}");
        }
        return test1;
    }
}
```

CAF Client-Side Annotations

These annotations provide additional CAF-specific metadata for custom components defined with annotations:

- **@ClientSideModel**. If this annotation is found on a custom JSF renderer class, then the configured client-side-model will be used for the rendered component.
- **@ClientSideValidator**. If this annotation is found on a custom validator class, then the additional client-side validation will be applied to any input control that is bound to the validator.

Pertinent code is highlighted in **bold**.

Example: Client-Side Model

```
package caf.war.testapp 1.test1;

import com.webmethods.caf.faces.annotations. ClientSideModel;
import com.webmethods.caf.faces.annotations. ClientSideScript;
import com.webmethods.caf.faces.render.html.input.BaseHtmlInputRenderer;

/**
 * Sample that demonstrates how to declare the CAF client-side-model
 * that should be used for the rendered component.
 */

@ClientSideModel (
    function = "CAF.TestApp1Custom.Model", //model js function
    base = "testapp1", //app containing the .js resources
    scripts = { //additional .js files to load into the page
        @ClientSideScript (
            resource = "/dyn/j/ui/js/controls/custom.js",
            library = "controls/testapp1"
        )
    }
)

public class CustomRenderer extends BaseHtmlInputRenderer {

    //TODO add your renderer implementation here
}
```


Example: Client-Side Validator

```

package caf.war.testapp1.test1;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

import com.webmethods.caf.common.StringTools;
import com.webmethods.caf.faces.annotations.ClientSideScript;
import com.webmethods.caf.faces.annotations.ClientSideValidator;

/**
 * Demonstrates creating a custom server-side validator from scratch.
 */
@FacesValidator(value="caf.war.testapp1.test1.validator.custom")
// (optionally) register the equivalent client-side validator as below

@ClientSideValidator (
    function = "TestApp1.customValidate", //validate js function
    base = "testapp1", //app containing the .js resources
    scripts = { //additional .js files to load into the page
        @ClientSideScript (
            resource = "/js/validators/custom.js",
            library = "validators/testapp1"
        )
    }
)
public class CustomValidator implements Validator {
    /* (non-Javadoc)
     * @see javax.faces.validator.Validator#validate(javax.faces.context.
     * FacesContext, javax.faces.component.UIComponent, java.lang.Object)
     */
    @Override
    public void validate(FacesContext context,
        UIComponent component,
        Object value) throws ValidatorException {
        if (!"hello".equals(value)) {
            Object o = component.getAttributes().get("label");
            if (o == null || (o instanceof String &&
                ((String) o).length() == 0)) {
                o = component.getValueExpression("label");
            }
            String label = null;
            if (o instanceof String) {
                label = (String)o;
            }
            FacesMessage msg = null;
            if (StringTools.notEmpty(label)) {
                msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                    label + " : Value must be 'hello'", //summary
                    label + " : Value must be 'hello'"); //details
            } else {
                msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                    "Value must be 'hello'", //summary
                    "Value must be 'hello'"); //details
            }
        }
    }
}

```

```
        }
        throw new ValidatorException(msg);
    }
}
```

CAF Portlet Annotations

When a portlet action request is being handled, the framework will first look for a matching method that contains the `@PortletAction` annotation in your portlet preferences bean. If it does not find a match there it will try to find a match in the page bean for whatever is currently the active page of the portlet.

- **@PortletAction.** When attached to an action handler method, this indicates that the method is a portlet action, meaning that it can be invoked as the target action of a portlet URL

Pertinent code is highlighted in **bold**.

Example:

```
package caf.war.testapp1.testportletaction;

import javax.portlet.PortletMode;
import javax.portlet.PortletPreferences;
import javax.portlet.WindowState;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

import com.webmethods.caf.faces.annotations.ExpireWithPageFlow;
import com.webmethods.caf.faces.annotations.DTManagedBean;
import com.webmethods.caf.faces.annotations.BeanType;
import com.webmethods.caf.faces.bean.portlet.PortletAction;
import com.webmethods.caf.portlet.IPortletURL;
import com.webmethods.caf.portlet.PortletURLFactory;

/**
 * Sample demonstrating methods annotated as a PortletAction meaning
 * that the methods can be invoked as the target action of a portlet url.
 */
@ManagedBean(name = "TestPortletAction")
@SessionScoped
@ExpireWithPageFlow
@DTManagedBean(displayName = "TestPortletAction", beanType = BeanType.PORTLET)
public class TestPortletAction extends
    com.webmethods.caf.faces.bean.BaseFacesPreferencesBean {

    private transient caf.war.searchapp1.Searchapp1 searchapp1 = null;

    /**
     * List of portlet preference names
     */
    public static final String[] PREFERENCES_NAMES = new String[] {
        "pref1"
    };

    /**
     * Create new preferences bean with list of preference names
     */
}
```

```

    */
    public TestPortletAction() {
        super(PREFERENCES_NAMES);
    }

    /**
     * Sample of building a portlet action url that can be bound
     * as the href of a link to run the action when the link is
     * clicked.
     *
     * @return href for a link that runs the action
     */
    public String getActionOneLink() { String href = null;
    try {
        String targetPageAlias = "targetPage"; // Base URL -- target page alias
        String targetPortletAlias = getPortletURI(); // Portlet alias or uri to
            target

        IPortletURL targetPageURL =
            PortletURLFactory.createActionUrl(getFacesContext());
        //start with clean URL state (omit to keep the state)
        targetPageURL.clearState();
        targetPageURL.clearParameters();

        //configure the page to start on (omit to stay on the same page)
        targetPageURL.setBaseUrl( targetPageAlias );

        //associate the target component for the action
        targetPageURL.setPortlet(targetPortletAlias);

        //(optionally) pass additional parameters to the target portlet
        targetPageURL.setParameter( "pref1", "new_value1" );

        //define the action to run in the target portlet
        targetPageURL.setTargetAction( "doActionOne" );

        //add the Anti-Cross-Site-Request-Forgery-Token when required
        targetPageURL.setAXSRFT( true );

        //(optionally) add other options for the target portlet
        targetPageURL.setWindowState( WindowState.NORMAL );
        targetPageURL.setPortletMode( PortletMode.VIEW );

        //build the href from the provided details
        href = targetPageURL.toString();
    } catch (Exception e) {
        error(e);
    }

    return href;
}

    /**
     * Portlet action handler. Since it is not otherwise
     * specified, the portlet action requires (by default) the
     * Anti-Cross-Site-Request-Forgery-Token to be present on
     * the action request.
     */

```

@PortletAction

```
public String doActionOne() {
    //TODO: do your action logic here

    return null;
}

/**
 * Another portlet action handler. This one does not
 * require the Anti-Cross-Site-Request-Forgery-Token
 * to be present before running.
 */

@PortletAction(axsrf=false)
public String doActionTwo()
    { //TODO: do your action logic here

        return null ;
    }

/**
 * Call this method in order to persist
 * portlet preferences
 */
public void storePreferences() throws Exception {
    updatePreferences();
    PortletPreferences preferences = getPreferences();
    preferences.store();
}

public caf.war.searchapp1.Searchapp1 getSearchapp1() {
    if (searchapp1 == null) {
        searchapp1 =
            (caf.war.searchapp1.Searchapp1) resolveExpression("#{Searchapp1}");
    }
    return searchapp1;
}

/**
 * The algorithm for this 'smart' preference getter is:
 * 1) Check the Request Map (skip this step if it isn't a 'smart' preference)
 * 2) Check the Member variable
 * 3) Fall back to the PortletPreferences
 */
public String getPref1() throws Exception {
    return (String) getPreferenceValue("pref1", String.class);
}

/**
 * Invoke {@link #storePreferences} to persist these changes
 */
public void setPref1(String pref1) throws Exception {
    setPreferenceValue("pref1", pref1);
}
```

10 Working with Portlet Preferences

■ About Portlet Preferences	110
■ Creating a Portlet Preference	110
■ Modifying a Portlet Preference	111
■ Specifying a Portlet Preference Validator	112
■ Wiring Portlets with Preferences	112
■ Exposing Portlet Preferences in My webMethods Server	113
■ Enabling the Show All Managed Beans Toolbar Button	114
■ About Preferences for the Default Portlet Types	115

About Portlet Preferences

Portlet preferences store values that can be used to define the behavior or appearance of a portlet. You can access a preference through the use of binding expressions. Preferences can be persisted through the use of scopes on a per user basis, meaning that each individual user has a separate preference value for each portlet instance. Composite Application Framework (CAF) provides several preference scope extensions for persisted values:

- Value stored per portlet instance

This scope extension enables storing different instances of the portlet on different portal pages with their own values. All users who access the portlet instance can see the same value. This scope extension is useful when wiring and linking portlets. This is the default portlet persistence preference setting in CAF.

- Value shared by all portlet instances

All instances of the portlet share the same value. This changes the preference value into a static variable. Use this scope when the preference is a configuration parameter that does not vary between portlet instances.

- Transient session value

The transient session value is stored for an individual user until that user logs out of the portal session or until the portlet bean scope expires.

Creating a Portlet Preference

You can create portlet preferences to store customization and configuration information.

➤ To create a portlet preference

1. Open a portlet application project and do one of the following:
 - Double-click a portlet view. In the Bindings view, right-click the portlet and click **Add > Portlet Preference**.
 - Double-click the portlet.xml file. In the Portlet Application Configuration editor, on the **Configuration** tab, click **Preferences** for a portlet and click **Add**.
2. In the Create Portlet Preference wizard, specify values for the following fields:

Field	Description
Name	Required. The name of the preference.
Value	Optional. The default value of the preference. Click Add , and then type a value.

Field	Description
Is Read-Only	Optional. Select the option if you want to prevent the user from modifying the preference.

3. Do one of the following:

- Click **Finish** to create the preference with default settings.
- Click **Next** to specify values for the following additional preference attributes:

Field	Description
Display Name	Optional. The preference name displayed on the Properties page for the portlet in My webMethods Server.
Description	Optional. A brief description explaining the purpose of the preference.
Type	Optional. The data type of the preference. The default type is String . If you do not select a value, Designer uses the default value to create the preference.
Scope	Optional. The scope of the preference. The default scope is Value stored per portlet instance . If you do not select a value, Designer uses the default value to create the preference. For more information about preference scopes, see “About Portlet Preferences” on page 110 .
Hidden	Optional. Clear the check box to display the preference on the Properties page for the portlet in My webMethods Server.

4. Click **Finish**. Optionally, if you are working with a legacy portlet, you can click **Next** to specify additional legacy portlet settings.

Modifying a Portlet Preference

After you create a portlet preference, you can modify its settings. After you modify a portlet preference, you must re-publish the portlet project to the server to apply your changes in the runtime.

➤ To modify a portlet preference

1. Open a portlet application project and double-click the portlet.xml file to open the Portlet Application Configuration Editor.
2. On the **Configuration** tab, expand the **Preferences** node for the portlet with which you want to work.
3. Click the preference that you want to modify.

4. In the **Portlet Preference** and **Portlet Preference (Extended)** areas of the editor, modify the preference settings as required.
5. Save your changes.

Specifying a Portlet Preference Validator

You can specify a validator class for your portlet preferences.

➤ To specify a portlet preference validator

1. Open a portlet application project and double-click the portlet.xml file to open the Portlet Application Configuration Editor.
2. On the **Configuration** tab, for the portlet with which you want to work, click **Preferences**.
3. Do one of the following:
 - Click **Browse** to select a pre-existing validator type. Begin typing the name of the validator in the **Select Entries** field to filter the available choices. The validator class must implement the PreferencesValidator interface.
 - Click the **Validator** link to open the New Java Class wizard. Specify attributes for the class that you want to create as a validator and click **Finish**. The newly created class opens in the Java editor.
4. Save your changes.

Wiring Portlets with Preferences

Portlet *wiring* connects one portlet to another portlet to enable the exchange of data. Typically, a portlet contains data stored as a preference. Through wiring, the portlet pushes the preference to another portlet, resulting in the exchange of the stored data.

You can test your portlet on My webMethods Server by selecting an available portlet and wiring the preference property from the first or source portlet to the destination portlet.

For more information about wiring the property of a portlet to the property of another, see *Administering My webMethods Server*.

➤ To use a preference to wire a value from one portlet to another

1. Open a portlet and create a preference with an internal value. For more information about creating portlet preferences, see [“Creating a Portlet Preference” on page 110](#). Be sure to clear the **Hidden** check box to make the preference available as a portlet property.

For example, create a preference named Postal Code, and for the value type a valid postal code, such as 90210.

2. With a portlet view open in the editor, locate the preference the Bindings view and drag the preference to the design canvas. A Basic Text Output control is added to the interface.
3. Select the Basic Text Output control in the editor and then click the **Value** tab in the Properties view. Observe that the **Value** field contains a binding expression to the preference you created.

For example:

```
{Portlet1DefaultviewView.portlet1.postalCode}
```

4. Publish the portlet to My webMethods Server and add the portlet to a server page. For information about adding a portlet to a server page, see *Administering My webMethods Server*.

When you view the page, the portlet should display the value you defined when you created the preference (in this case, the postal code 90210).

5. Locate a portlet that requires a postal code as an input (such as the Yahoo! Weather portlet), and add it to the page.
6. On the new portlet, click **Tools > Wiring**.
7. On the portlet wiring page, locate the property for the zip code value. For the Yahoo! Weather portlet, the property name is **Zip Code**.
 - a. In the **Source Portlet** column, click the drop-down list and select the portlet that contains the preference you created in Step 1.
 - b. In the **Source Property** column, click **Browse**, click the name of the preference you created in Step 1 (in this case, Postal Code), and then click **Select**.
8. Click **Submit**.

You have now wired the preference value you created in your original portlet in Step 1 to a property of the second portlet (in this case, the Zip Code property of the Yahoo! Weather portlet). When you view the server page that contains the portlets, the postal code value you specified in the first portlet should be used to render the results displayed in the second portlet.

To change the preference value in the first portlet, click **Tools > Properties**, and in the **Portlet Preferences** area, type a new value for the preference and click **Apply**. When you return to the server page, the new value should be used to render the results displayed in the second portlet.

Exposing Portlet Preferences in My webMethods Server

When you create preferences for a JSR 168-compliant portlet, the best practice for viewing and updating those preferences in My webMethods Server is to provide a user interface for portlet preferences. Typically, the solution is to provide an implementation of the edit portlet mode. With

the portlet edit mode, you can provide whatever user interface elements you would like to display the portlet preferences and enable the user to modify them.

You can investigate an implementation of this by looking into the Search Results portlet contained within a standard task application created in Designer. The portlet contains a `default.xhtml` and an `edit.xhtml` file. In this case, the edit view provides controls to enable the user to set various preferences such as number of rows to display and sort order. The values for these controls are set by hidden preferences created within the portlet (`initialPageSize` and `initialSortAscending`, for example).

The controls on the edit view are bound to the preferences through a binding expression, displayed in the **Value** field in the control's properties, for example:


```
#{activePreferencesBean.initialSortAscending}
```

The edit view is then implemented to appear as the **Preferences** tab in the Properties page of the Search Results portlet.

Because task portlets are prepackaged within Designer, you cannot easily modify them if you want to experiment further with this approach. However, you can create a new portlet application project and then create a new Search Results portlet within it. For more information about how to create a portlet, see [“Creating a Portlet” on page 63](#).

In the **Portlet Type** field, select **Search Results**. The new Search Results portlet contains a `default.xhtml` and an `edit.xhtml`, which you can further investigate and experiment with.

Enabling the Show All Managed Beans Toolbar Button

The Bindings view displays managed beans associated with the page bean for the view currently displayed in the design canvas. If you want to see managed beans that are not in the current view page bean, you can enable the **Show All Managed Beans** toolbar button .

When all managed beans are displayed in the Bindings view, you can add references to managed beans in the project, and delete managed beans that are not referenced by the current page bean or by any other managed bean in the project.

➤ To change preferences to show all managed beans

1. In Designer, go to **Window > Preferences** and expand **Software AG**.
2. Expand **UI Development** and click **Bindings View**.
3. On the Bindings View panel, select **Display the ‘Show All Managed Beans’ toolbar button** and click **OK**.

You must close and re-open the Bindings view to display the **Show All Managed Beans** button on the toolbar.

About Preferences for the Default Portlet Types

When you create a portlet, you can select from several predefined Composite Application Framework portlet types. Some of these default portlets are created with predefined preferences:

- **Generic** portlet. This portlet is created with no predefined preferences.
- **Search Bar** portlet. This portlet is created with the following preferences, the values of which are set by controls or properties in the user interface:
 - `initialSearchTab` (hidden)
 - `initialSelectedSavedSearch` (hidden)
 - `runSearchOnDisplay` (hidden)
 - `noMaxResults` (hidden)
 - `maxResults` (hidden)
 - `lastSearchState` (hidden)
 - `savedSearchMap` (hidden)
- **Search Results** portlet. This portlet is created with the following preferences, the values of which are set by controls or properties in the user interface, including the portlet's edit view, which is rendered in the **Preferences** area in the portlet's properties page:
 - `queryString`
 - `initialPageSize` (hidden)
 - `initialSortBy` (hidden)
 - `initialSortAscending` (hidden)
 - `columnDisplay` (hidden)
 - `columnWidths` (hidden)
- **Search Bar + Results** portlet. This portlet is created with the following preferences, the values of which are set by controls or properties in the user interface, including the portlet's edit view, which is rendered in the **Preferences** area in the portlet's properties page:
 - `initialPageSize` (hidden)
 - `initialSortBy` (hidden)
 - `initialSortAscending` (hidden)
 - `columnDisplay` (hidden)
 - `columnWidths` (hidden)
 - `initialSearchTab` (hidden)

- `initialSelectedSavedSearch` (hidden)
- `runSearchOnDisplay` (hidden)
- `noMaxResults` (hidden)
- `maxResults` (hidden)
- `lastSearchState` (hidden)
- `savedSearchMap` (hidden)

You can add additional portlet preferences to these portlets to further control the portlet behavior. For example, if you want to use a tree view for search results, you can add a `treeColumnWidths` preference.

11 Working with Legacy Portlets

■ About Legacy Portlets	118
■ Enabling Legacy Portlet Support	118
■ Importing Legacy Portlets	119
■ Changing Legacy Portlet Preferences	119

About Legacy Portlets

You can import portlet applications that were not developed using Composite Application Framework (CAF). The non-CAF applications are JavaServer Pages (JSP)-based applications, and are referred to as *legacy portlets*.

To work with legacy portlets in CAF, you must enable the MWS Legacy Portlet Support capability. For more information about enabling support for legacy portlets, see [“Enabling Legacy Portlet Support” on page 118](#).

You can log on to the Software AG TECHcommunity website to download legacy portlet code samples included in [webm_CAF_Samples_821.zip](#). You can use these samples to familiarize yourself with importing legacy portlet applications and manipulating the files before you try the same operations with your own applications. For more information about importing legacy portlets, see [“Importing Legacy Portlets” on page 119](#).

After you import your legacy portlet application, you can view the source artifacts in the UI Development perspective’s Navigator or Project Explorer views. The directory structure of legacy portlets reflects the type of application imported. For example, your project might not have a WebContent node in the legacy portlet application.

You can begin modifying the legacy portlet by double-clicking the portlet.xml file to open the portlet.xml and the wm-portlet.xml files in a text editor. Because of their legacy content, the portlet.xml file cannot be opened with the Portlet Application Configuration Editor.

Enabling Legacy Portlet Support

Before you can work with legacy portlets, you must enable the tools to work with legacy portlet applications.

➤ To enable legacy portlet support

1. In Software AG Designer, go to **Window > Preferences**.
2. Expand the **General** node and click **Capabilities**.
3. On the Capabilities panel, click **Advanced**.
4. In the Advanced Capabilities dialog box, expand **Development**, select **MWS Legacy Portlets**, and then click **OK**.
5. Click **OK** to close **Preferences**.

You can now use Composite Application Framework to import and work with legacy portlets.

Importing Legacy Portlets

A legacy portlet is any portlet developed in a platform other than Composite Application Framework (CAF), but primarily refers to JSP-based portlets developed before the common use of JSF. You can experiment by importing legacy projects from the sample files found in [webm_CAF_Samples_821.zip](#). After extracting the samples file, you can find legacy portlets in the following directories:


- /analysis/
- /pagecomponents/
- /security/
- /services/directory/

You must have legacy portlet support enabled to use these samples and instructions. For more information, see [“Enabling Legacy Portlet Support” on page 118](#).

➤ To import legacy portlets into your Composite Application Framework workspace

1. In Software AG Designer, click **File > Import**.
2. In **Import**, expand **Software AG**, expand **Legacy Portlets**, click **Import Existing Component**, and then click **Next**.
3. To populate the **Existing Portlet Source Folder** field, click **Browse** to select the location of the source files to import.

Note:

An error icon  appears in the wizard header to indicate that a selected folder does not contain any legacy portlet projects.

4. Click **Next**. On the **Java Settings** page, define the Java build settings for the project and then click **Finish**.

A new project for the legacy portlet is added to the Navigator and Project Explorer views. The project is not available in the Solutions view.

Changing Legacy Portlet Preferences

Use the following procedure to change the configuration of the My webMethods Server instance to which you want to publish legacy portlets.

Note:

You must enable legacy portlet support before you can change any legacy portlet preferences. For more information about enabling support for legacy portlets, see [“Enabling Legacy Portlet Support” on page 118](#).

➤ **To change preferences for legacy portlets**

1. In Designer: **Window > Preferences > Software AG > UI Development > Legacy Portlets**.
2. Specify values for any or all of the My webMethods Server configuration parameters, listed in the following table:

Field	Description
Target Server URL	The host and port of the target server, for example: <code>http://localhost:8585/</code>
Default Portlet Prefix	The portlet prefix for your legacy portlets. The default value is <code>wm</code> .
HTTP Request Timeout (seconds)	The time in seconds before a request to the server times out. The default value is 60 seconds.
My webMethods Server Location	The location of the server instance in your file system.

3. Click **Apply > OK**.

12 Working with CAF Events and Notifications

■ About CAF Events	122
■ Opening the CAF Events Editor and Enabling CAF Events	123
■ Creating a Custom CAF Event	124
■ Creating an Event Handler	125
■ Creating a Notification	131
■ Creating a Subscription	133
■ Configuring Simple Conditions	135
■ Configuring Advanced Condition and Action Expressions	138

About CAF Events

Composite Application Framework (CAF) events are useful for triggering actions programmatically, as well as for informing users when a change occurs in the system, such as when a user logs in or logs out, uploads a file, or creates or deletes a folder. My webMethods Server generates many different events, but only subset of these events are available for use as CAF events. In addition to the default event types that are available, you can define your own custom event types.

My webMethods Server users can subscribe to and receive e-mail notifications generated by a CAF event, if the event is configured to send notifications. CAF events are created and edited with the CAF Events Editor.

With the CAF Events Editor, you can create:

- A custom event type, if needed.
- One or more event handlers.
- One or more actions to be triggered by an event handler (such as invoke a service).
- One or more conditions that you want to apply to the event (evaluated either as simple conditions or by a binding expression you define).
- A notification for the event, if desired, including a template notification view.
- A subscription for each notification you create, if desired, including a template subscription view.

Note:

CAF events are unrelated to webMethods Event Driven Architecture (EDA) events. For more information about working with EDA events, see *webMethods Event Processing Help*.

About the Default My webMethods Server Event Types

In a Composite Application Framework (CAF) portlet application, you can associate your custom event handler with any one of the default My webMethods Server event types. Custom events that you have previously created can also be associated with a handler.

The following table lists the default My webMethods Server event types:

Event Type	Description
Create	Initiates an event when a user or the system creates an object such as a page, portlet, folder, or file.
Delete	Initiates an event when a user or the system deletes an object such as a page, portlet, folder, or file.
Login	Initiates an event when a user or the system successfully logs in or logs out.

Event Type	Description
Login Failed	Initiates an event when a user's login or logout fails or the system's login or logout fails, or when a login script that is initially run to create a user's folders completes.
Update	Initiates an event when an object is updated such as a page, portlet, folder, or file.
View	Initiates an event when a user or the system accesses an object such as a page, portlet, folder, or file.
Workspace Delete	Initiates an event when a workspace is deleted.
Workspace Share	Initiates an event when a workspace is shared.
Workspace Un-Share	Initiates an event when a workspace sharing is terminated.
Workspace Update	Initiates an event when a workspace is updated.

About My webMethods Server Login Events

The `MWSLoginEvent` has an Action field which fires the following actions at specific stages in the login process:

- `ACTION_FIRST_LOG_IN(0)` fires only once after the user initially logs into My webMethods Server.
- `ACTION_LOGIN_SCRIPT_DONE(5)` fires only once after the login script creates the user personal folders
- `ACTION_LOGGED_IN(1)` fires with each occurrence.
- `ACTION_LOGGED_OUT(2)` fires with each occurrence.
- `ACTION_TIMED_OUT(3)` fires with each occurrence.
- `ACTION_FAILED_LOGIN(4)` fires with each occurrence.
- `ACTION_FIRST_LOGIN_SCRIPT_DONE(6)` fires with each occurrence.

You can collect these actions in the My webMethods Server Events Collector database and implement a page in your application to query and display the collected actions. For more information, see *Administering My webMethods Server*.

Opening the CAF Events Editor and Enabling CAF Events

This procedure describes how to enable the project for CAF events and open the CAF Events Editor for a project for first time. When you do so, the editor creates an `events.xml` configuration file in the project's WEB-INF folder.

When a project is first created, it is not enabled for CAF events. In addition to opening the CAF Events Editor, the following procedure also automatically enables your project for CAF Events.

If you prefer to enable the project manually, right-click the project in the Navigator view, click **Properties**, click **Project Facets**, and then select the **CAF Events** check box.

Tip:

After you open the CAF Events Editor for a project for the first time, you can open the editor for subsequent work in these ways:

- In the Solutions view, expand **User Interfaces**, expand the project, and double-click **Events Configuration**.
- In the Navigator view, expand the project, expand the WEB-INF folder, and double-click the events.xml file.

➤ **To initially open the CAF Events Editor for a project**

1. In the Navigation view, open a project.
2. Right-click the WebContent folder and click **New > Events**.

Tip:

You can also access the CAF Events Editor from any of the locations where the New menu is available. For example, **File > New**, or from the New button on the toolbar.

3. In the New Events wizard, do one of the following:
 - Select the current project in the **Project** list.
 - Click **New** to create a new project.
4. Click **Finish**.

This automatically adds the CAF Events facet to your project and enables the project for CAF events.

Creating a Custom CAF Event

A custom CAF event can describe any arbitrary asynchronous or synchronous event that is supported by a portlet application.

➤ **To create a custom CAF event**

1. In the Navigation view, open a project.
2. Expand the WebContent node, and expand the WEB-INF node, and double-click the events.xml file.
3. In the CAF Events Editor, click the **Events** tab, then click **Add**.
4. In the Add Custom Event Type wizard, type a name for the event in the **Name** text box.

5. (Optional) In the **Description** text box, type an optional description.
6. Click **Finish** to create the custom event with default settings, or click **Next** to modify the default settings.
7. On the Managed Bean page, accept the default scope or select a scope for the event from the **Managed Bean Scope** list box. Click **Finish** to create the custom event, or click **Next** to continue modifying settings.
8. On the Java Type page, accept the default package name, type a name in the **Package** text box, or click **Browse** to select a folder from the **Package Selection** dialog box.
9. (Optional) In the **Interfaces** box, click **Add**, and then select the interface. Type filter text in the **Choose Interfaces** field to help locate the interface you want.
10. (Optional) Click **Generate Comments**.
11. Click **Finish**. The new custom event appears in the **Events Type** list on the **Events** tab.
12. Save your changes.

Note:

You will also find the event type in the **Event Variables** node in the Bindings view (you may have to click the **Events** tab to see the variable in the Bindings view). The event type contains several default properties, and you can add your own custom properties as required.

Creating an Event Handler

Each event handler is associated with a single specific event, and manages the response to that event. When an event occurs, the event handler determines how the event is processed.

When you create an event handler, a variable for the event type associated with the handler is automatically added in the **Event Variables** node in the Bindings view (you must be viewing the Handlers tab to see the variable in the Bindings view).

➤ To create an event handler

1. In the CAF Events Editor, click the **Handlers** tab.
2. In the Handlers area, click **Add**.
3. In the General area, type a name for the handler in the **Name** text box.
4. (Optional) Type a description in the **Description** text box.

5. In the CAF Event Type area, select a My webMethods Server or custom event type in the **Event Type** list box.
 6. Select a **Listener Type**:
 - **Queued** Default. Executes the event handler on only one node in a cluster after the event initiates.
 - **Asynchronous** Executes the event handler on every node in a cluster after the event initiates.
 - **Synchronous** Immediately executes the event handler on every node in a cluster when the event initiates.
- Note:**
Increasing the number of synchronous event handlers can significantly increase the processing time of the action that initiated the event.
7. (Optional) In the **Condition Type** list box, specify a simple or advanced condition type. For more information, see [“Configuring Simple Conditions” on page 135](#) and [“Configuring Advanced Condition and Action Expressions” on page 138](#).
 8. In the Handler Actions area, specify a simple or advanced action type. At least one action is required. For more information, see [“Configuring a Simple Event Handler Action” on page 127](#) and [“Configuring Advanced Condition and Action Expressions” on page 138](#).
 9. Save your changes.

About CAF Event Handler Actions

For each event handler you create, you must include one or more handler actions to be executed by the event handler. The following table lists the available actions:

Handler Action	Description
Fire Ajax Event	Provides Ajax functionality for the event. For more information, see “Configuring a Fire Ajax Event Action” on page 127 .
Invoke Service	Invokes a service for the event. For more information, see “Configuring an Invoke Service Action” on page 129 to configure the service.
Send Notification	Initiates a notification for the event. You must first create the notification before you can specify this action. For more information, see “Configuring a Send Notification Action” on page 129 .

Configuring a Simple Event Handler Action

You can specify the actions that govern the behavior of an event handler in the run-time environment by choosing from a list of predefined (simple) actions.

Note:

If you find you cannot create the action you want with the simple actions that are available, you can create an advanced (custom) action expression. For more information, see [“Configuring Advanced Condition and Action Expressions” on page 138](#).

➤ To configure a simple event handler action

1. Open the project and the CAF Event Editor as described in [“Opening the CAF Events Editor and Enabling CAF Events” on page 123](#).
2. Click the **Handlers** tab and click the handler you want to work with.
3. Expand the Handler Actions area if it is not already visible.
4. Click **Add** to add an action. In the Action Selection dialog box, select one of the following actions:
 - **Fire Ajax Event.** Provides Ajax functionality for the event. See [“Configuring a Fire Ajax Event Action” on page 127](#).
 - **Invoke Service.** Invokes a service or an action method for the event. Select a service or action from the list box. See [“Configuring an Invoke Service Action” on page 129](#).
 - **Send Notification.** You must create a notification before you select Send Notification. See [“Configuring a Send Notification Action” on page 129](#).
5. Click **OK** to add the action.
6. Repeat steps 4-5 if you want to add another action.
7. Save your changes.

Configuring a Fire Ajax Event Action

When you create a Fire Ajax Event action, you do so in conjunction with an Open Ajax control (`com.webmethods.caf.faces.events.OpenAjaxScript`) implemented in a client view. The Open Ajax Event Name you specify when you create the Fire Ajax Event action must match the name of the Open Ajax Topic property of the Open Ajax control placed on the client view.

The Open Ajax control is typically implemented along with the Growler control, which displays client-side notifications. For more information about the Open Ajax and Growler controls, see *webMethods CAF Tag Library Reference*.

When you specify the The Open Ajax Event Name, you can use a wild card character to include multiple event types. For example, if you have several event types that start with `user.` such as, `user.logged.in`, or `user.logged.out`, you can use an asterisk (*) to include all user event types, for example, `user.*`.

➤ **To configure a Fire Ajax Event action**

1. Open the CAF Events Editor as described in [“Creating a Custom CAF Event” on page 124](#).
2. In the CAF Events Editor, click the **Handlers** tab, then click the handler you want to work with.
3. In the Handler Actions area, ensure that **List of Simple Actions** is selected as an action type, then click **Add**.
4. In the Action Selection dialog box, click **Fire Ajax Event**.
5. In the **Open Ajax Event Name** text box, type a name. Refer to the information in the introductory paragraph above regarding this name.
6. Click **OK**.
7. Create a new portlet application or select another portlet application to receive the event.

Note:

You must add the portlet to the applicable shell to notify all users who are configured to use the shell. In addition, place the portlet in the appropriate page to notify users who can access the page. For more information, see [“Embedding Portal Resources in a Page File” on page 94](#).

8. Open the portlet view that you want to work with.
9. In the Palette view, expand **MWS Html > Control > Output**, and then drag an Open Ajax Script control onto the design canvas.
10. Select the control on the canvas.
11. In the Properties view, click the **Value** tab, and type the Open Ajax Event Name that you created in Step 5 into the **Open Ajax Topic** text box.

Note:

If you want to implement a Growler control, you can add it to the portlet as well.

12. Save your changes

Configuring an Invoke Service Action

You can add an Invoke Service action to an existing event handler, or you can create a new event handler and add the Invoke Service action to it.

Note:

You must add at least one service (as a web service descriptor) or an action method to the Events Handler Services managed bean before you can complete this procedure. For more information, see these topics:

- [“Creating a Web Service Descriptor” on page 196.](#)
- [“Creating a New Action Method” on page 156.](#)

➤ To create an Invoke Service action

1. Open the CAF Events Editor as described in [“Creating a Custom CAF Event” on page 124.](#)
2. In the CAF Events Editor, click the **Handlers** tab, then click the handler you want to work with.
3. In the Handler Actions area, ensure that **List of Simple Actions** is selected as an action type, and then click **Add**.
4. In the Action Selection dialog box, click the **Invoke Service** action
5. In the **Service** list box, select a web service or action method.
6. Click **OK** to create the Invoke Service action.
7. Save your changes.

Configuring a Send Notification Action





You can add a Send Notification action to an existing event handler, or you can create a new event handler and add the Send Notification action to it.


Note:


You must create at least one notification before you can create a send notification action. For more information, see [“Creating a Notification” on page 131.](#)


➤ To create a Send Notification action

1. Open the CAF Events Editor as described in [“Creating a Custom CAF Event” on page 124.](#)
2. In the CAF Events Editor, click the **Handlers** tab, then click the handler you want to work with.

3. In the Handler Actions area, ensure that **List of Simple Actions** is selected as an action type, and then click **Add**.
4. In the Action Selection dialog box, click the **Send Notification** action
5. In the **Notification** list box, select a notification.
6. In the Recipients area, click **Add**.
7. In the Recipient Picker dialog box, select a recipient type:
 - **Email Address.** Type the recipient's e-mail address in the **User** text box, or click , and then select an applicable data field that provides an email address.
 - **Folder.** Type the My webMethods Server folder alias in the **Folder** text box, or click , and select the applicable folder from the My webMethods Server taxonomy. You must publish the application to My webMethods Server to be able to see the available folders. Specifying an alias provides portability.
 - **Folder ID.** Type the folder ID in the **Folder ID** text box, or click , and select an applicable data field that provides a folder ID.
 - **Group UID.** Type the group ID in the **Group UID** text box, or click , and select an applicable data field that provides a group ID.

Note:
When the recipient is Group UID, the application sends an email message to the email alias assigned to the group rather than to each member in the group.
 - **Role.** Click  and select the applicable My webMethods Server role.

Note:
When the recipient is role, the application sends an e-mail message to the email alias assigned to the role rather than to each member in the role.
 - **Role UID.** Type the role ID In the **Role UID** text box, or click , and select an applicable data field that provides a role UID.

Note:
When the recipient is Role UID, the application sends an e-mail message to the e-mail alias assigned to the role rather than to each member in the role.
 - **User UID.** Type the user ID in the **User UID** text box, or click , and select an applicable data field that provides a role UID.
8. Repeat steps 6 - 7 if you want to add additional recipients for the notification.
9. After you finish selecting and configuring recipient types, click **OK**.

10. Click **OK** to create the notification action.
11. Save your changes.

Creating a Notification

You can create an event notification to include an e-mail message, an RSS message, or both. When you create a notification, you must specify the My webMethods Server or custom event type that initiates the notification of a change.

Note:

To configure the notification for RSS after you create it, see [“Configuring an RSS Notification” on page 132](#).

➤ To create a notification

1. Open the CAF Events Editor as described in [“Creating a Custom CAF Event” on page 124](#).
2. In the CAF Events Editor, click the **Notifications** tab.
3. Click **Add**.
4. In the **Add New Notification** wizard, type a name in the **Notification Name** text box.
5. (Optional) Type an optional in the **Notification Description** text box.
6. In the **Event Type** list box, select an event type, do one of the following:
 - Click **Next** and follow the instructions in the wizard to customize the notification view locations and name, managed bean name, managed bean scope, and Java attributes.
 - Click **Finish** to accept the remaining configuration parameters as defaults.
7. Save your changes.
8. After you click **Finish**, the wizard creates a notification bean and a notification provider, and opens a template notification view in the editor. The following table lists the properties of the template notification view, that you can customize:

Displayed Text	Control and Description
Email subject text	Text Output. A field for the subject for the e-mail notification.
email@company.com	Text Output. A field for the email address of the sender.
New Notification	HTML Header. A field for the notification header.

Displayed Text	Control and Description
Title	Text Output. A field for the title for the RSS notification.
Link	Text Output. A field for the URL for the RSS notification.
Description	Text Output. A field for the description of the RSS notification.

Configuring an RSS Notification

Note:

You must create a notification view to be able to complete this procedure. For more information, see [“Creating a Notification” on page 131](#).

➤ To configure the RSS notification

1. In the Navigator view, open the portlet application you want to work with and expand it to expose the notification View Declaration Language (VDL) file:
portlet_application/WebContent/viewName.view
2. Double-click the VDL file to open it in the editor.
3. Configure the **Text Output control for the** RSS title. This control must be implemented with an RSS renderer.
 - a. On the design canvas, click the Text Output control, and then, in the Properties view, click the **Value** tab.
 - b. In the **Value** text box, type a title for the RSS notification.
4. Configure the Text Output control for the URL that links to the RSS notification.
 - a. On the design canvas, click the Text Output control and then, in the Properties view, click the **Value** tab.
 - b. In the **Value** text box, type a URL for the RSS notification.
 - c. Click the **Display** tab, and in the **Label** text box, type a display name for the URL.
5. Configure the **Text Output control** for the description of the RSS notification.
 - a. On the design canvas, click the **description** component, and then, in the Text Output control view, click the **Value** tab.
 - b. In the **Value** text box, type a description for the RSS notification.

Tip:

You can use the Atom Feed Icon control to export the contents of a specified table for the RSS notification. For more information about the Atom Feed Icon control, see *webMethods CAF Tag Library Reference*.

Creating a Subscription

Note:

You cannot create a subscription without a notification and you can only create one subscription for a notification.

After you create a notification, you can create a subscription for the notification. When you create a subscription, you specify the conditions for the subscription to be used with the notification.

➤ To create a subscription

1. Open the CAF Events Editor as described in [“Creating a Custom CAF Event” on page 124](#).
2. In the CAF Events Editor, click the **Subscriptions** tab.
3. Click **Add**.
4. In the **Add New Subscription** wizard, type a name in the **Subscription Name** text box.
5. (Optional) Type a description in the **Subscription Description** text box.
6. In the **Notification** drop-down list, select a notification, then do one of the following:
 - Click **Next** and follow the instructions in the wizard to customize the subscription view locations and name, managed bean name, managed bean scope, and Java attributes.
 - Click **Finish** to accept the remaining configuration parameters as defaults.
7. Save your changes.
8. After you click **Finish**, the wizard creates a subscription bean and a subscription provider, and opens a template subscription view in the editor. You can customize this view to your own needs, as described in [“Configuring the Default Subscription View” on page 134](#). The following table lists the components that the view contains:

Displayed Text	Control and Description
Message	Formatted Messages. Displays error messages with a Details button to toggle a display of error contents.
Subscription for Custom CAF Event	Property Group. Provides a label field for the subscription and contains controls for the subscription.

Displayed Text	Control and Description
Subscribed:	Property Line. Contains the Text Output control.
Subscribed	Text Output. Contains a binding expression to the subscription provider in the managed bean.
No text displayed	Submit group. Contains the subscribe/unsubscribe buttons.
Subscribe	Async Command Button. Subscribes to the subscription.
Unsubscribe	Async Command Button. Unsubscribes from the subscription.



Configuring the Default Subscription View


When you create a subscription, a template, or default, subscription view is created and opened in the view editor, as described in [“Creating a Subscription” on page 133](#). Several control properties are pre-configured in the default view, and you can modify other control properties as described in the following procedure.

➤ To configure the default subscription view

1. Locate the portlet application in the tree view and expand it to expose the subscription View Declaration Language (VDL) file:

```
portlet_application/WebContent/subscriptionName.xhtml
```

2. Double-click the VDL file to open it in the editor.
3. Configure a label for the Property Group control.
 - a. Click the Property Group control, and in the Properties view, click the **Display** tab.
 - b. In the **Value** text box, type the text you want to appear for the subscription, or click  and then select a field that will provide the text.
4. Configure the Property Line control.
 - a. Click the Property Line control, and in the Properties view, click the **Display** tab.
 - b. In the **Value** text box, accept the default value of “Subscribed”, type a different label for the Property Line, or click  and then select a field that will provide the text.
5. Configure the Text Output control.
 - a. Click the Text Output control, and in the Properties view, click the **Value** tab.

- b. In the **Value** text box, type the text that you want to appear in the text box, or click  and then select a field that will provide the text.
6. Configure the **Subscribe** button. The key properties of this control are pre-configured as follows, but you can modify them as required:
 - In the Properties view, on the **Display** tab, the **Disabled** property is preset to:
`#{SubscriptionNameView.subscriptionProvider.subscribed}`
 - On the **Action** tab, the **Action** property is preset to:
`#{SubscriptionNameView.subscribe}`
 - On the **Value** tab, the **Value** property is preset to `Subscribe`.
7. Configure the **Unsubscribe** button. The key properties of this control are pre-configured as follows, but you can modify them as required:
 - In the Properties view, on the **Display** tab, the **Disabled** property is preset to:
`#{SubscriptionNameView.subscriptionProvider.subscribed}`
 - On the **Action** tab, the **Action** property is preset to:
`#{SubscriptionNameView.unsubscribe}`
 - On the **Value** tab, the **Value** property is preset to `Unsubscribe`.
8. Save your changes.

Configuring Simple Conditions

You can specify the conditions that govern the behavior of an event handler or subscription. When the event or subscription is triggered in the run-time environment, it will be subject to these conditions.

Note:

If you do not define any conditions, the event or subscription action occurs depending on the basic event action or subscription settings. If you find you cannot create the condition you want with the simple task conditions that are available, you can create an advanced (custom) condition expression. For more information, see [“Configuring Advanced Condition and Action Expressions” on page 138](#).

➤ To configure simple conditions

1. Open the project and the CAF Event Editor as described in [“Opening the CAF Events Editor and Enabling CAF Events” on page 123](#).
 - For event handlers, click the **Handlers** tab and click the handler you want to work with.

- For subscriptions, click the **Subscriptions** tab and click the subscription you want to work with.
2. Expand the Conditions area if it is not already visible. In the **Condition Type** box, ensure that **Joined List of Simple Conditions** is selected. The Simple conditions editor appears immediately below the list.
 3. Specify the following condition settings:

Note:

The **Join** options are disabled unless two or more conditions are defined.

- **Join** — Click one of these three options to specify how the conditions will be evaluated:
 - **ALL are true** — All of the listed conditions must be true to trigger the assignment.
 - **ANY is true** — Any of the listed conditions must be true to trigger the assignment (that is, it can be one, two, or any subset of the condition list).
 - **ONE only is true** — A single listed condition must be true to trigger the assignment.
4. Click **Add** to add a condition. The Add Simple Condition dialog box appears.
 - In the **Field** box, click the browse button and select the data field you want to base the condition on.
 - In the **Operation** box, select the operator you want to apply to the condition.
 - In the **Value** box, type a value you want the Field value to be evaluated against, or click the browse button to select a data field.
 5. Click **OK** to add the condition.
 6. Repeat steps 2-4 if you want to add another condition.
 7. Save your changes.

Modifying a Simple Condition

You can modify the conditions that govern event handlers and subscriptions.

➤ To modify a simple condition

1. Open the project and the CAF Event Editor as described in [“Opening the CAF Events Editor and Enabling CAF Events” on page 123](#).
 - For event handlers, click the **Handlers** tab and click the handler you want to work with.

- For subscriptions, click the **Subscriptions** tab and click the subscription you want to work with.
2. In the **Conditions** box, select the simple condition you want to modify.
 3. Click **Edit**. The Edit Simple Condition dialog box appears.
 4. Modify the condition settings as described in [“Configuring Simple Conditions” on page 135](#).
 5. Save your changes.

Deleting a Simple Condition

You can delete the conditions that govern event handlers and subscriptions.

➤ To delete a simple condition

1. Open the project and the CAF Event Editor as described in [“Opening the CAF Events Editor and Enabling CAF Events” on page 123](#).
 - For event handlers, click the **Handlers** tab and click the handler you want to work with.
 - For subscriptions, click the **Subscriptions** tab and click the subscription you want to work with.
2. In the **Conditions** box, select the simple condition you want to delete.
3. Click **Delete**. The condition is removed from the project.
4. Save your changes.

Rearranging the Order of Simple Conditions

You can rearrange the order of the simple conditions that govern event handlers and subscriptions.

When the Any join option is applied, conditions are evaluated in order, beginning at the top of the condition list, until a condition is matched. To speed up performance, you can, for example, move the conditions most likely to be matched to the top of the list, and place the least likely matches at the bottom.

Because you can call a service as a result of a event handler condition definition, you may want to create several conditions that search for simple matches, and if they fail, you can create an event handler action that calls a service with a more sophisticated matching algorithm.

➤ To rearrange the order of a simple task condition

1. Open the project and the CAF Event Editor as described in [“Opening the CAF Events Editor and Enabling CAF Events” on page 123](#).
 - For event handlers, click the **Handlers** tab and click the handler you want to work with.
 - For subscriptions, click the **Subscriptions** tab and click the subscription you want to work with.
2. In the **Conditions** box, select the simple condition you want to move.
3. Click the **Up** or **Down** button to relocate the selected condition in the list. Although this functionality is available for all join options (All, Any, or One), it has significance only for the Any join option.
4. Save the task to apply your changes.

Configuring Advanced Condition and Action Expressions

If you cannot create the handler or subscription conditions you want with the simple conditions and actions editors, you can create advanced (custom) syntax-based expressions to govern the behavior of a handler or subscription.

➤ To configure advanced condition and action expressions

1. Open the project and the CAF Event Editor as described in [“Opening the CAF Events Editor and Enabling CAF Events” on page 123](#).
 - For event handlers, click the **Handlers** tab and click the handler you want to work with.
 - For subscriptions, click the **Subscriptions** tab and click the subscription you want to work with.

Note:

If you have already created and saved simple conditions for the handler or subscription, they will appear in the advance condition editor in their text-based format. You must delete the simple conditions and save your changes to clear this display.

2. Do one of the following:
 - In the **Condition Type** field, select **Advanced condition expression**.
 - In the **Action Type** box, select **Advanced action expression**.

The Advanced editor appears immediately below the list.
3. Click **Edit** to open the Edit Expression dialog box.
4. Create the advanced condition or action expression you want to apply either by selecting the expression terms with the available buttons, or by typing in the expression directly.

5. Save your changes.

13 E-form Support in Portlet Applications

■ Adding E-form Support to a CAF Portlet Application	142
■ Configuring a CAF Portlet Application Project for E-form Support	142
■ Adding an IS Document Type for E-form Support	142
■ Adding an E-form Template Type to a CAF Portlet View	143
■ Adding an E-form Template from a File System or Web Server	143
■ Adding an E-form Template from a Repository	145
■ Adding a Basic Default E-form Download or Upload User Interface	145
■ Adding an Advanced Default E-Form Download or Upload User Interface	146

Adding E-form Support to a CAF Portlet Application

You can add support for electronic forms (e-forms) to a Composite Application Framework (CAF) generic portlet application. Before you can add e-form support, you must create an Integration Server document type by importing an e-form template. For more information about creating an IS document type from an e-form template, see *webMethods Service Development Help*.

For more information about working with e-forms in general, see *Implementing E-form Support for BPM*.

The general workflow for implementing e-form support is as follows:

- Configure the CAF portlet application project for e-form support.
- Add the e-form-sourced IS document type you want to work with to a portlet view.
- Create and configure the content provider for the e-form instances.
- Create and configure the template provider for the specified e-form template.
- Add download and upload controls to the portlet view.

You can carry out this procedure for each e-form you want to add to the portlet.

Configuring a CAF Portlet Application Project for E-form Support

To support e-forms, your Composite Application Framework (CAF) portlet application project must contain the CAF JCR Client library component.

➤ To configure a CAF portlet application for e-form support

1. Create a new CAF generic portlet project; for more information, see [“Creating a Portlet Application Project” on page 62](#).
2. In the Navigator view, right-click the project name and click **Properties**.
3. Click **Project Facets**.
4. Select the **CAF JCR Client** check box.
5. Click **Apply**.
6. Click **OK**.

Adding an IS Document Type for E-form Support

Important:

This procedure requires an IS document type that was generated from a supported e-form template. For more information about creating an IS document type from an e-form template, see *webMethods Service Development Help*.

➤ To add an IS document type for e-form support

1. Create a new Composite Application Framework generic portlet in your portlet project, or select an existing portlet view in a project configured for e-form support as described in [“Configuring a CAF Portlet Application Project for E-form Support” on page 142](#).
2. Open the generic portlet view in the editor.
3. In the Package Navigator view, drag the e-form-enabled IS document type to the portlet view in the design canvas.
4. In the Add Property wizard, type a name or accept the default value, and click **Finish**.
5. In the Bindings view, expand the portlet view managed bean, right-click the entry for the IS document type, and then click **New > Content Provider**.
6. In the Java Type wizard, click **E-form Content Provider**.
7. Click **Next** to continue defining the content provider values, or click **Finish** to use the default values.
8. Save your changes.

Adding an E-form Template Type to a CAF Portlet View

This procedure requires access to a supported e-form template, as described in *Implementing E-form Support for BPM*. You can add the template from a JCR-170 compliant repository, or from a file location or URL:

- [“Adding an E-form Template from a Repository” on page 145](#)
- [“Adding an E-form Template from a File System or Web Server” on page 143](#)

Adding an E-form Template from a File System or Web Server

➤ To add an e-form template from a file system or web server

1. Create a new Composite Application Framework generic portlet in your portlet project, or select an existing portlet view in a project configured for e-form support. For more information, see [“Configuring a CAF Portlet Application Project for E-form Support” on page 142](#).
2. Open the generic portlet view in the editor.

3. From the Palette view **Data > E-form** folder, drag the **E-form File Template Provider** and drop it onto the portlet view page bean in the Bindings view.
4. On the Choose Data Target wizard, click **Add new shared managed bean and reference**.
5. Click **Next** to continue defining the template provider values, or click **Finish** to use the default values.
6. In the Bindings view, expand the file e-form template provider and click the URL entry.
7. In the Data Binding tab of the Properties view, and enter a URL value in the **Value** field that defines the e-form template's URL, local file path, or web application relative URL. This value must end with the file name extension that identifies the type of the template such as .xsn or .xdp:

- For URL-referenced templates, this URL must be accessible at run time by the server running the portlet application. The URL must allow anonymous access or be protected with basic authentication. In the case of basic authentication, the user name and password must be specified on the URL in a form of:

`http://user:password@server/path`

Any URL accessible resource can be specified here.

- For templates referenced from a file system, use file:// protocol. The file path must be accessible at run time by the server that runs the portlet application.
- For a web application relative URL, start the URL with a slash (/) and place a copy of the template under the portlet applications WebContent folder, for example, for a copy stored under `WebContent/eforms/myform.xsn` use an URL similar to the following:
`/eforms/myform.xsn`.

Note:

If you have not already created an e-form content provided for the view, do so now, as described in [“Adding an IS Document Type for E-form Support” on page 142](#).

8. In the Bindings view, expand the e-form content provider and click **Template Provider**.
9. In the Data Binding tab in the Properties view, click the expression binding button.
10. On the Expression Binding dialog box, click the template provider.
11. Click **OK**.
12. (Optional) In the e-form content provider, click the **Download File Name** entry and define a binding for the e-form download file name.
13. Save your changes.

Adding an E-form Template from a Repository

➤ To add an e-form template from a repository server

1. Create a new Composite Application Framework generic portlet in your portlet project, or select an existing portlet view in a project configured for e-form support. For more information, see [“Configuring a CAF Portlet Application Project for E-form Support” on page 142](#).
2. Open the generic portlet view in the editor.
3. Locate a supported e-form template in the MWS Admin view.
4. Drag the e-form template from the MWS Admin view and drop it onto the portlet view page bean in Bindings View.
5. On the Java Type wizard, click **E-form JCR Template Provider**.
6. Click **Next** to continue defining the template provider values, or click **Finish** to use the default values.

Note:

If you have not already created an e-form content provided for the view, you must do so before continuing. For more information, see [“Adding an IS Document Type for E-form Support” on page 142](#).

7. In the Bindings view, expand the e-form content provider and click **Template Provider**.
8. In the Properties view, click the **Data Binding** tab, and then click the expression binding button.
9. On the **Expression Binding** dialog box, click the template provider.
10. Click **OK**.
11. Optional: In the e-form content provider, click the **Download File Name** entry and define a binding for the e-form download file name.
12. Save your changes.

Adding a Basic Default E-form Download or Upload User Interface

Before you add the default e-form user interface, first add content and template providers to the portlet view as described in the following procedures:

- [“Adding an IS Document Type for E-form Support” on page 142](#)

■ [“Adding an E-form Template Type to a CAF Portlet View” on page 143](#)

➤ **To add a basic default download or upload controls to the portlet view**

1. In the UI Development perspective, open the generic portlet view in the design canvas.
2. From the Bindings view, drag the e-form content provider to a specific control in the portlet view such as the `Property Subgroup` control.

This creates a minimal user interface consisting of a file input control with a browse button, and an **Upload** and **Download** button.

3. Right-click the **Download** button, click **Change Control Type**, and click **Command Button**.
4. Continue customizing the interface as necessary.
5. Save your changes.

Adding an Advanced Default E-Form Download or Upload User Interface

This procedure adds the ability to upload and submit an e-form by including Form Submission controls.

➤ **To add a more advanced default download/upload interface to the portlet view**

1. Open the generic portlet view in the editor.
2. Open the Snippets view.
3. Expand **E-forms**.
4. Drag the **Download and upload UI** snippet onto the portlet canvas and drop it inside a Form control.
5. In the Variables area, specify a value for `providerBeanName` by typing the Java property name of your e-form content provider.

Note:

To see Java proper names in the Bindings View, click the **Show Java Type Names** toolbar button.

6. Click **Insert**.
7. Save your changes.

14 Using a Simple Email Deliverer

■ About the CAF Simple Email Deliverer	148
■ Adding a Simple Email Deliverer	148
■ Specifying the Mail Session	149
■ Specifying Data Binding for a Simple Email Deliverer	149

About the CAF Simple Email Deliverer

You create a view that sends e-mail from a Composite Application Framework application. Add the Simple Email Deliverer control to a form View Declaration Language (VDL) file. The Simple Email Deliverer is a content provider that has no user interface but provides sample fields that you can use in your application. The sample property group of form fields in the Simple Email Deliverer are defined to accept data required to support the JavaMail API for SMTP. You should create a user interface for the Simple Email Deliverer that meets your specific user interface requirements. You can modify or replace the sample user interface elements generated when you drop the Simple Email Deliverer into a view. You must update the Composite Application Framework application project's web.xml with the resource environment variables for the mail session.

If the Composite Application Framework application is deployed to My webMethods Server, verify that the E-mail server settings are properly configured in **Administration > My webMethods > E-Mail Servers**.

If the Composite Application Framework web application is deployed to another instance of My webMethods Server, use the *Administering My webMethods Server* documentation for steps required to set up a mail session. In the server.xml file, add the following code inside the <context> element.

```
<resource-env-ref> <resource-env-ref-name>mail/Session</resource-env-ref-name>
<resource-env-ref-type>javax.mail.Session</resource-env-ref-type>
</resource-env-ref>
```

Adding a Simple Email Deliverer

You must create a view as a container for the **Simple Email Deliverer** user interface control. You can use the Composite Application Framework form template and add the Simple Email Deliverer data provider to a Modal Dialog or Modeless Dialog control.

For information about the control's properties, see the reference documentation for the Simple Email Deliverer control.

➤ To add a simple email deliverer

1. In the UI Development perspective of an opened Web or portlet application, click the Project Explorer or Solutions view, select an existing view to edit or add a new view to the project.
2. In the Palette view, expand the **Data** node, and then drag the **Simple Email Deliverer** to the view in the design canvas.
3. In the Choose Data Target panel, click **Add new simple property to portletname/default**, and then click **Next**.
4. In the Add New Property panel, in the **Property Name** text box, type a name for the property.

5. (Optional) Select **Property is writable (setter will be generated)** check box, and then click **Finish**.

Specifying the Mail Session

When you want to use the **Simple Email Deliverer** in a Composite Application Framework application, you must update the web.xml file to get a JavaMail session for sending messages.

➤ To update the web.xml file for sending email messages

1. In the UI Development perspective, click the Navigation view of the opened Composite Application Framework application that uses the **Simple Email Deliverer**.
2. In the Navigation view, expand the **WebContent > WEB-INF** folder, and then double-click the web.xml file.
3. In the design canvas, click the **Source** tab to add <resource-env-ref> element in the web.xml file.

In the web.xml file, add the following:

```
<resource-env-ref>
<resource-env-ref-name>mail/Session</resource-env-ref-name>
<resource-env-ref-type>javax.mail.Session</resource-env-ref-type>
</resource-env-ref>
```


4. On the **web.xml** tab, click , and then click **Yes** to save the changes to the web.xml file.

Specifying Data Binding for a Simple Email Deliverer

You can define the values for the fields of a **Simple Email Deliverer** anytime after adding the data provider to a view in your Composite Application Framework application. You might want to wait until after you have finalized the user interface for the **Simple Email Deliverer**. The Composite Application Framework sample **Simple Email Deliverer** might contain fields that you do not plan on using in your final design such as the Bcc (blind carbon copy) Addresses field.

➤ To specify data values for the Simple Email Deliverer

1. In the UI Development perspective of an opened Composite Application Framework application, open the view containing the **Simple Email Deliverer** in the design canvas.
2. Click the Binding view, expand the **Managed Bean** node, expand the portlet or Web application node, and expand the **Simple Email Deliverer** node.
3. In the **Simple Email Deliverer** node, click a field to add a data value.

4. Click the Properties view, click the **Data Bindings** tab, and then click **Expression Binding**  for the specific **Simple Email Deliverer** field.
5. In the **Expression Binding** wizard, expand the **Simple Email Deliverer** node, select the field, and then click **OK**.

15 Working with the Bindings View

■ About the Bindings View	152
■ Binding a Managed Bean Property to Another Bean	153
■ Binding Data to a Control	154
■ Binding a User Attribute to an Output Text Control	154
■ Binding a Security Role to a Control	155
■ Binding Security Roles in an Access Control Panel	155
■ Creating a New Action Method	156

About the Bindings View

The Bindings view provides a logical representation of the data available in the object open in the current editor session. In the Bindings view, you can bind data or methods to the user interface (UI) controls enabling the controls to display data or perform some action. The Bindings view facilitates defining data bindings for data elements. The Bindings view is active when a view in the design canvas or another editor contains a control that needs data. When you select a node in the Bindings view, properties for that node are displayed in the Properties view.

The Bindings view contains several types of managed beans and other assets used by the portlet.

The Bindings view filters data to the portlet page visible in the design canvas. You should leave this filter in place to avoid using data from a different portlet. You need to create binding expressions to data associated with the current portlet.

The Bindings view has automatic user interface generation. If you drag an object from the Bindings view to the design canvas, the canvas is populated with controls appropriate to the object. For some objects, such as Web service connectors, there are multiple sets of controls. You can choose among available input controls. Also in the case of complex objects such as Web connectors, you can choose which elements of the object to include. For a simple example of this, see [“Wiring Portlets with Preferences” on page 112](#).

Binding expressions connect data to a property or method to a control. Binding expressions take the following form:

```
# {name_of_managed_bean.property_it_is_getting}
```

You create bindings expressions in Composite Application Framework using a combination of the Bindings view, Properties view, and the design canvas. The Bindings view has automatic user interface generation. If you drag an object from the Bindings view to the design canvas, the canvas is populated with controls appropriate to the object.

You can use the following to help you create binding expressions:

- [“Binding Data to a Control” on page 154](#)
- [“Binding a Managed Bean Property to Another Bean” on page 153](#)
- [“Binding a User Attribute to an Output Text Control” on page 154](#)
- [“Binding a Security Role to a Control” on page 155](#)
- [“Binding Security Roles in an Access Control Panel” on page 155](#)
- [“Creating a New Action Method” on page 156](#)

In the Bindings view, you can access the portlet application project’s managed beans and other assets such as the:

- 

Portlet Application Bean exposes resource bundles, resources, and user attributes. Each portlet application has one application bean.



Portlet Bean exposes preferences and security roles. There is one portlet bean for each portlet in a portlet application.



Page Bean exposes the UI Component Model that is generated when Controls are dropped onto the page. This is the primary Bean that you interact with when creating User Interfaces. There is one Page Bean for each page in a portlet.

The Bindings view displays data only for the portlet that contains the view visible in the design canvas. You should leave this filter in place to avoid using data from a different portlet. You need to create binding expressions to data associated with the current portlet.

Binding a Managed Bean Property to Another Bean

Data binding and data flow assignments enable casting the properties of one managed bean to the properties of another managed bean, such as the properties of one Java type to another Java type. The assignment of different managed beans, or complex types, is not supported by the Java language. Composite Application Framework automatically converts the managed beans on different types using a field-by-field conversion on the field name.


You can replicate a managed bean by assigning the properties of one managed bean to another managed bean. You can also choose to assign a value to the property.

➤ To assign a property of one managed bean to another managed bean

1. In the UI Development perspective, open the View Declaration Language (VDL) file for the view you want to work with in the editor.
2. Click the Bindings view, expand the **Managed Beans** node, and then expand the bean you want to work with and select the bean property, such as a property for a Web service connector.

Note:

By default, the Bindings view displays only those managed beans associated with the page bean for the view currently displayed in the editor. If you want to display all managed beans, see [“Enabling the Show All Managed Beans Toolbar Button” on page 114](#).

3. Click the Properties view, and then click the **Data Binding** tab.
4. On the **Data Binding** tab, next to the **Value** property, click .
5. In the **Expression Binding** wizard, select the binding target and then click **OK**.

Binding Data to a Control


Binding expressions are the means by which you bind data to controls. JSF binding expressions take the following form:

```
# {name_of_managed_bean.property}
```

Properties can have sub-properties, in which case a binding expression might have a more complex form, such as this:

```
# {name_of_managed_bean.property.subproperty1.subproperty2}
```

> To bind data to a control


1. In the **UI Development** perspective, in an open Web or portlet application project, drag the view with the property to the design canvas.
2. In the design canvas, select the control to receive a binding expression.
3. Click the **Properties** view, click **Expression Binding**  next to the property.
4. In the **Expression Binding** wizard, expand the **Page Bean** node, select the data to bind to the control, and then click **OK**.

The resulting binding expression displays in the field associated with the property.

Binding a User Attribute to an Output Text Control

You can bind a user attribute to a control directly from the Bindings view. Expand nodes in that view until you expose the user attributes and drag a user attribute directly to a control container in the design canvas.

> To use a user attribute in a binding expression

1. In the **UI Development** perspective, in an opened Composite Application Framework project, drag the view to update to the design canvas.
2. In the Palette view, expand **CAF Html > Control > Output**, and then drag a Text control to a location in the view on the design canvas.
3. Select the **Text** control in the design canvas, and then click the Properties view.
4. In the **Properties** view, click the **Value** tab, and click .
5. In **Expression Binding**, expand the **Managed Beans** node, select the **User Attribute** to bind to the control, and then click **OK**.

Binding a Security Role to a Control

Before you can bind an application security role to a control, you must create the security role. For more information on creating a security role, see [“Add a Security Role to a Portlet Application” on page 49](#) or [“Add a Security Role to a Web Application” on page 48](#).

The binding expression `#{securityRoles["role1"]}` resolves to true if the user is a member of the privileged role. You can build the logic to fit your needs for the view such as `#{securityRoles["role1"] and securityRoles["role2"] or securityRoles["role3"]}`.

➤ To bind a security role to a control

1. In the **UI Development** perspective of an opened Composite Application Framework project, drag a control to the view in the design canvas.
2. In the design canvas, select the control and then on the **General** tab in the **Properties** view, set the **Rendered** property to true.
3. Click the **Bindings** view, expand **Implicit Variables**, select the security role from the **Application Security Roles** node or the **Portlet Security Roles** node, and drag the security role to the control in the design canvas.
4. Select the control in the design canvas, click the **Properties** view, and then click the **Value** tab.
5. Verify that the security role is set as the value in the control.


Binding Security Roles in an Access Control Panel

Use the Composite Application Framework **Access Control Panel** in a View Declaration Language (VDL) file to create an access relationship by a security role to a specific view. You can deny or allow access to defined security roles. You must have defined security roles before you begin this procedure. For more information, see [“Add a Security Role to a Web Application” on page 48](#) or [“Add a Security Role to a Portlet Application” on page 49](#).

➤ To manage security roles using an Access Control Panel control

1. In the **UI Development** perspective of the opened application, drag the view to contain the **Access Control Panel** to the design canvas.
2. In the Palette view, expand **CAF Core > Control > Panel**, and then drag an Access Control Panel to the view in the design canvas.
3. Click the **Bindings** view, expand **Implicit Variables**, and then expand **Web Security Roles** or **Portlet Security Roles**.

You must have security roles defined in the **Bindings** view to use this procedure.

4. Select the **Access Control Panel** in the design canvas, click the **Properties** view, and then click the **Display** tab.
5. In the **Display** tab, click  next to **Allowed Security Role** or **Denied Security Role**.
6. In **Expression Binding**, expand **Implicit Variables**, and then expand **Web Security Roles** or **Portlet Security Roles**, select a security role, and then click **OK**.

The view shows the updated **Access Control Panel** with the allowed or denied access for the specified security role.

Creating a New Action Method

When designing a user interface in Composite Application Framework, you can use this procedure to create a new action method that is invoked by a command.

When you create the method, you can select the **Implement this Action as Data Flow** option if you want to use binding expressions to assign data from a source to a destination and call other actions without any coding. By default, each page bean has an `Initialize()` method. You can use the Data Flow Implementation to define data flow for initializing the action. The method is called before the page is rendered the first time.

➤ To create a new action method in a page managed bean

1. In the Navigation view, open a project, then open the portlet view you want work with in the view editor.
2. In the Bindings view, right-click the page managed bean, and then select **Add > Action**.
3. In the **Add New Action Method** wizard, type a name for the method in **Action Name** field and do one of the following:
 - Select the **Implement this action as a Data Flow** and click **Next** to specify target and source properties for a data flow, and then click **Finish**.
 - Click **Finish** to create the action method without a data flow.

The method appears as an action on the Bindings view beneath the managed bean.

4. In the Bindings view, double-click the newly created method to open it in the Java editor.
5. In the Java editor, type the method, for example:

```
}  
    public void doSomething()  
    {  
        System.out.println("something");  
    }  
{
```

6. Click **File > Save**.
7. Click the tab for the view editor.
8. To use the method on the page, drag it from the Bindings view to the design canvas.

By default, a Command button appears on the design canvas when you drop the action, with an automatic binding to the method.

16 Working with My webMethods Server

■ Adding a My webMethods Server Instance in Designer	160
■ Editing a My webMethods Server Instance	161
■ Starting a My webMethods Server Instance	161
■ Stopping a My webMethods Server Instance	161
■ Running My webMethods Server in Debug Mode	162
■ Specifying a Default Application Server	162
■ Publishing an Application to My webMethods Server	162
■ Manually Deploying a Web Application to My webMethods Server	163
■ Deploying a Jar File to My webMethods Server	164
■ Exporting from My webMethods Server	164

Adding a My webMethods Server Instance in Designer

Before you can run Composite Application Framework (CAF) and webMethods OpenUI applications in My webMethods Server, your environment must be configured to support connectivity between My webMethods Server and Software AG Designer. In many development environments, it is common for My webMethods Server and Designer to be installed on the same system.

A default installation of Designer and My webMethods Server results in the following configuration:

- My webMethods Server is configured to run on port 8585, resulting in a URL of `http://localhost:8585`.
- Designer is configured to use this URL as the default My webMethods Server instance. Designer automatically detects the default My webMethods Server instance.

➤ To add a My webMethods Server instance at a location that is different than `localhost:8585`

1. In the Servers view of the UI Development perspective, right-click and click **New > Server**.
2. In the **Select the server type** list, select **My webMethods Server (Remote)**.
3. In the **Server's host name** field, type the host name of the server. The default value is `localhost`.
4. (Optional) In the **Server name** field, type a name to identify the server instance in the Servers view.
5. Click **Next**.
6. Specify values for the following fields:

Field	Description
Protocol	The type of server connection. Values are: <ul style="list-style-type: none"> ■ http ■ https
Http(s) Port	The port of the server instance to connect to. The default value is 8585.
Debug Port	The debug port of the server to connect to. The default value is 10033.
Publish Timeout (seconds)	The time in seconds before a request to publish to My webMethods Server times out. The default value is 120 seconds.

7. Click **Finish**.

The new My webMethods Server instance is displayed in the Servers view.

Editing a My webMethods Server Instance

Use the following procedure to edit a My webMethods Server instance in Designer.

➤ To edit the connection information for a My webMethods Server instance

1. In the Servers view of the UI Development perspective, perform one of the following steps:
 - Double-click the server that you want to edit.
 - Right-click the server that you want to edit and click **Open**.
2. Under **Connection Information**, configure the connection by modifying the values of one or more of the following fields :


Field	Description
Protocol	The type of server connection. Values are: <ul style="list-style-type: none"> ■ http ■ https
Http(s) Port	The port of the server instance to connect to. The default value is 8585.
Debug Port	The debug port of the server to connect to. The default value is 10033.
Publish Timeout (seconds)	The time in seconds before a request to publish to My webMethods Server times out. The default value is 120 seconds.

3. Click **File > Save**.

Starting a My webMethods Server Instance

Use the following procedure to start a My webMethods Server instance in Designer.


➤ To start My webMethods Server

1. In the UI Development perspective, click the Servers view.
2. In the Servers view, select a server to start, and click .

Stopping a My webMethods Server Instance

Use the following procedure to stop a My webMethods Server instance.


> To stop an application server that is running

1. In the UI Development perspective, click the Servers view.
2. In the Servers view, select a server to stop, and click .

Running My webMethods Server in Debug Mode

Use the following procedure to start My webMethods Server in debug mode.

> To start a server in debug mode

1. In the UI Development perspective, click the Servers view.
2. Select a server to run in debug mode, and click . If the server is already running, it will be shut down and restarted in debug mode.

Specifying a Default Application Server

You must associate a Composite Application Framework project with a specific server. You can use an instance of My webMethods Server. Your Composite Application Framework application uses the instance of the server specified in this procedure as the runtime application environment.

> To specify a default runtime application server

1. In the UI Development perspective, open the project you want to work with.
2. In the Navigator view, right-click the web or portlet application, and then click **Properties**.
3. In the **Properties** dialog box, click **Server**.
4. In the list **Always use the following server when running this project**, do one of the following:
 - Select the server you want to use as the default server for the project. This list is populated with the servers available in the Servers view.
 - Select **None** if you do not want to associate the project with any server.
5. Click **OK**.



Publishing an Application to My webMethods Server

After you develop your CAF or webMethods OpenUI application project, you must add it to the target runtime environment before you can publish it to My webMethods Server.

Important:

If you make changes to an application after adding it to a server, you must manually publish those changes to the server.

➤ **To add a CAF or OpenUI application to an instance of My webMethods Server**

1. In the UI Development perspective, click the Servers view, right-click the server, and then click **Add and Remove**.
2. In the **Add and Remove** dialog box, select the desired application project in the **Available** list and click **Add** to move the application project to the **Configured** list.
3. If you are not ready to publish to the server, clear the **If server is started publish changes immediately** check box.
4. Click **Finish**.
 - If the server is running, and the check box in step 3 is not cleared, the application is immediately published to the selected server.
 - If the server is stopped, you must click  and then click  to publish the application.

Manually Deploying a Web Application to My webMethods Server

When you develop a web application in the UI Development perspective of Software AG Designer, you can publish the application directly to My webMethods Server as described in [“Publishing an Application to My webMethods Server”](#) on page 162.

However, you might have web applications from other sources that you want to deploy manually to My webMethods Server. You can choose from the following methods for handling web applications developed outside of Software AG Designer:

- You can import the project into Designer, and then publish the web application to My webMethods Server.
- You can manually place the web application archive (.war) file directly into the My webMethods Server folder structure.

➤ **To manually deploy a web application to My webMethods Server**

1. Package the web application as a .war file.
2. Copy the .war file to this directory:

Software AG_directory \MWS\server\server_name\deploy

Tip:

When you deploy a very large (~200 MB) .war file to My webMethods Server, it is possible that My webMethods Server can attempt to start internal deployment of the file before it is fully copied. In this case, change the file name extension of the .war file before you copy it (for example, to *filename.war.tmp*). When the copy procedure finishes, rename the .war file back to its original name.

Deploying a Jar File to My webMethods Server

You can deploy a jar file to My webMethods Server in any of the following ways:

- By publishing it to My webMethods Server as described in [“Publishing an Application to My webMethods Server” on page 162](#). This capability also applies to Java EE Utility jar files.
- By copying the file directly to the *Software AG_directory \MWS\server\instance_name\deploy* folder.
- By logging in to My webMethods Server as sysadmin and installing the component from the **Administrative Folders > Administration Dashboard > Configuration > Install Administration** page.

The deployed project will be picked up by My webMethods Server without having to stop and start the server or run the update command (sometimes referred to as a “hot deployment”).

Note:

When you deploy a very large (~200 MB) file to My webMethods Server, it is possible that My webMethods Server can attempt to start internal deployment of the file before it is fully copied. In this case, change the file name extension of the file before you copy it (for example, to *filename.jar.tmp*). When the copy procedure finishes, rename the file back to its original name.

Exporting from My webMethods Server

You can export portlet pages from My webMethods Server to Composite Application Framework in two forms:

- **XML Import file.** You can export the raw XML that is a serializable representation of a portlet or portal page.

If you have exported an XML import file, you can find it in the Navigator view, in this location:

portlet_application/WebContent/WEB-INF/config/xmlImport.xml


- **Deployable component (CDP file).** You can export a binary representation of a portlet or portal page.

If you have exported deployable component, you can find it on the Navigator view, in this location:

portlet_application/WebContent/WEB-INF/config/component_name.cdp

After exporting, you can deploy this portlet application to another instance of My webMethods Server or to another location on the same server.

➤ **To export a portlet or portal page from the server to Composite Application Framework**

1. In the UI Development perspective, click the Servers view, and then click  to start the My webMethods Server instance.
2. After the server has started, in the MWS Admin view, right-click the host to **Reconnect** or **Browse** My webMethods Server instance.
3. In the **Authentication Request**, type the **UserName** and **Password** for the host, and then click **OK**.
4. In the MWS Admin view, browse to the application that contains the portlet page you want to work with.
5. Right-click the portlet page and click **Import/Export > Export**.
6. In the **Export Data** wizard, select the portlet application to use for the source from the **Project** list.
7. Select the XML import file or the deployable component.

17 My webMethods Server Runtime Assets

■ About Runtime Asset Extraction	168
■ Extracting Runtime Assets	172
■ Creating a Deployable Component	173

About Runtime Asset Extraction

In the MWS Admin view, you can extract runtime assets that reside on an instance of My webMethods Server into a project as file-based representations of these extracted assets. You can then save the extracted asset project into a version control system.

An asset is any object that you created or work with in an instance of My webMethods Server. Runtime assets include tasks, rules, skins, shells, and other configuration elements.

You can extract assets into two project types:

- **MWS content project.** This project type includes any existing web or portlet application projects you have created in Designer. You can also create a new content project when you extract an asset.
- **MWS skin project.** These projects are used exclusively for extracting skin assets. You cannot extract skin assets to a content project.


You can associate an MWS content project and MWS skin project type with a My webMethods Server in the Servers view and publish these projects to the server in the same manner as any other portlet or web application project.








In addition, you can create a deployable component (.cdp file) from an MWS content project or MWS skin project using the Apache Ant Java build tool from a command prompt. The Ant tool builds the deployable component in place. For more information, see [“Creating a Deployable Component” on page 173](#).






For content projects, each extracted asset is saved in an xmlImport.xml file that is placed in a directory under the **WebContent** node. For example, the xmlImport.xml file-based representation of an extracted shell rule is located in the /WebContent/WEB-INF/assets folder. Skin projects have a different structure that you can browse in the Navigator view.






Although the MWS Admin view shows all the available MWS runtime assets deployed on the My webMethods Server instance, avoid extracting every runtime asset or every child asset in the tree of extractable assets. For example, extracting the *everyone* user group from Groups is not useful because the users in the *everyone* group are already stored and captured in the internal directory service system or the external directory service you are using. The **Extract Asset into Project** menu option is not available for elements that are not extractable.








The following table lists the runtime assets that you can view in the MWS Admin view, and information about whether the asset has dependencies on other runtime assets.

Runtime Asset	Asset Icon	Description	Dependencies
Access Privileges		Defines the rights of a user, group, or role to view applications and features in the Navigation panel and access pages associated with them in My webMethods Server. The granting of Access Privileges,	Select the access privileges associated with a user, group, or role. Extract the user, group, or role specified by the access privilege.

Runtime Asset	Asset Icon	Description	Dependencies
		by itself, allows only the capability to view.	
Business Calendars 		Defines the global calendars used in My webMethods Server, for example, US holidays calendar.	None.
CAF Runtime Configuration 		Specifies the runtime configuration for a CAF application; the extract contains the editable variables available through the My webMethods Server Runtime Editor. The actual CAF application is not included in the extract.	None.
Certificates 		Lists public key certificate of a digitally signed certificate from a Certificate Authority, that states the entity in question is valid and trusted.	None.
Data-level Security 		Defines field or record read, write, delete security permissions for a resource used by the Monitoring portlets to control access processes, documents, and KPIs.	Extract the role associated with the data level security.
Data Sources 		Specifies the external data source configuration information used with the runtime assets.	None.
Directory Services 		Defines access to a collection of entries for users with a set of attributes such as name, e-mail address, groups, and roles in an external LDAP server or database.	When using an external directory service such as LDAP, extract the connectivity information to the users and groups including domain, data source, and LDAP provider details for extracting to the project.
Folders 		Defines the layout and file content of page.	Extract the users, groups, and roles included in the page or folder access control.

Runtime Asset	Asset Icon	Description	Dependencies
			In addition, folders and pages have a dependency on the portlets and CAF applications included in the page.
Functional Privileges		Defines the rights of a user, group, or role to make changes within an application or feature, such as to create and modify a workspace.	Extract all functional privileges for a specific role or group; selecting a single functional privilege is not supported. Also, select the applicable access privileges for extraction.
Groups		Defines a collection of users and other groups. Groups are defined and stored in an internal system or external directory service.	Extract the external directory service where the groups are defined.
Locale Rules		Specifies the locale, language and country code, to use when locale information is not specified in the user profile.	Extract users impacted by the locale rule.
Login Page Rules		Specifies the login page to use based on rules such as whether the user is accessing the page from inside or outside of the firewall, and other conditions, such as rules for accessing a sales portal login page.	Extract users, groups, or roles impacted by the login rule, and the login page referenced by the rule.
Rules		Defines a collection of rules for displaying pages, for example, login page rules, start page rules, renderer rules, and locale rules. The MWS Admin view lists additional asset rules such as Task rules and Shell rules.	<p>Select the specific rule to view the applicable dependency.</p> <ul style="list-style-type: none"> ■ See Locale Rules in this table. ■ See Login Page Rules in this table. ■ See Shell Rules in this table. ■ See Skin Rules in this table.

Runtime Asset	Asset Icon	Description	Dependencies
			<ul style="list-style-type: none"> ■ See Start Page Rules in this table. ■ See Task Rules in this table.
Renderer Rules		Defines the user interface formatting capabilities assigned to specific server objects by defining rendering rules. Renderer rules determine the specific renderer to use.	Extract the asset for which the renderer rule applies.
Roles		Defines a collection of users assigned to specific roles defined for any directory service. My webMethods Servers supports static, dynamic, LDAP query, rule-based, and database roles. For more information on a specific role type, see <i>Administering My webMethods Server</i> .	Extract the role dependent on user, groups, and roles contained within the role.
Saved Searches		Defines the queries that are performed regularly by a user. The asset extraction captures the search criteria used in the definition of the global search for the saved search type.	No dependencies; extract the global saved search by type. You cannot extract individual saved searches.
Security Realms		Defines an approach that system administrators use for managing access control lists to resources based on a set of users, groups, and roles on large servers. Security realms apply to a single set of access control lists, and are folders (also called containers).	Extract the users, groups, or roles, and the pages and folders controlled by the security realm.
Shells		Specifies an installable component, that is, a page used to display the header, footer, and title bars.	None.

Runtime Asset	Asset Icon	Description	Dependencies
Shell Rules		Specifies the shell displayed for each page as the user navigates by the shell rule. in My webMethods Server.	Extract the shell referenced
Skins		Specifies an installable component that defines the look and feel of the rendered page. A skin modifies the images, fonts, colors, and other subtle style aspects of HTML content, without functionally modifying the HTML content.	None.
Skin Rules		Specifies the skin to display for a specific user, based on a variety of criteria such as a submit event.	Extract the skin and the users referenced by the skin rule.
Start Page Rules		Defines the page that displays after the user logs into the application.	Extract the users referenced by the start page rule.
Tasks		Defines a solution for repeated activities performed as part of a larger process or as standalone action used in a portlet.	None.
Task Rules		Represents changes made at runtime to task rules.	Extract the task referenced in the task rule.
Users		Specifies the individuals listed in the internal or external directory service.	Extract the external directory service and data source when the internal system directory service is not used.

Extracting Runtime Assets

To extract My webMethods Server runtime assets using the MWS Admin view, you must be connected to the My webMethods Server instance where the application is deployed. The MWS Admin view is available by default on the UI Development perspective.

- You must extract skin assets into an existing skin project, or create a new skin project. You cannot extract skin assets into a content project.
- Extract all other assets into an existing or new content project.

When you select an asset to extract, Designer displays any existing project that matches the asset type in the **Project** field of the Extract Asset into Project dialog box. You can use an existing project or create a new project.

➤ **To extract runtime assets into a project**

1. In the MWS Admin view, expand an asset node to select an asset to extract, right-click, and click **Extract Asset Into Project**.
2. In the Extract Asset into Project dialog box, do one of the following:
 - Select an existing project in the **Project** list. Only projects that match the asset type (content or skin) are displayed.
 - Click **New** and use the wizard to create a new project (either skin or content, depending on the asset type).
3. In the Extract Asset into Project dialog box, click **Finish**.

For content asset types, an xmlImport file is inserted in the WEB-INF/assets directory of the specified project. When a skin is extracted, the results are added to the root directory of the skin project.

Creating a Deployable Component

You can create a deployable component package (.cdp) file from a content project or skin project. For general information about extracting runtime assets, see [“About Runtime Asset Extraction” on page 168](#).

➤ **To create a deployable component package from an extract project**

1. Create a content project or skin project and extract the runtime assets into the project as described in [“Extracting Runtime Assets” on page 172](#).
2. Follow the instructions in the readme.txt file located in the *Software AG_directory\MWS\mycomponents* folder.

18 Performance Monitoring

■ Measuring Application Performance	176
■ Creating Custom Tokens for Performance Monitoring	177

Measuring Application Performance

In your Composite Application Framework application, you can use custom performance tokens, to measure the performance of a specific portlet, bean, class, or method. You might need to create numerous custom token to obtain a thorough performance analysis of your application.

When creating custom tokens, you should define and initialize all custom tokens using the `initialize()` method of the managed bean's code instead of initializing the custom tokens in the methods used before each call. If you want to use custom tokens in non-managed bean code such as a utility class, you can use a static class initialization blocks. You can define custom token and category names. However, to avoid any conflicts during the invocation, you must use unique names. For token names, you should use *portlet_or_class_name/method_name*.

You should also use the portlet application name as a performance category. Later, you can use the performance category name to browse the performance analysis information of the functionality.

Use separate tokens to measure the performance of different areas of functionality. You can use and invoke the same token, however, using the same token loads extra performance data that can lead to incorrect measurements of the concrete functionality. You can split up a particular function or method call into sections that consists of one or more than one line of code for a more thorough performance analysis.

You can collect performance information by inserting `PerfUtil.actionStart(customToken)` in the beginning and `PerfUtil.actionEnd(customToken)` in the end of the business logic which you want to measure and analyze.

```
public String customFunction() {
try {  PerfUtil.actionStart(customToken);          ...
    <custom_business_logic>
    ...
return OUTCOME_OK;
}
finally {  PerfUtil.actionEnd(customToken);      }
}
```

You can invoke multiple nested tokens in an application. Later, you can browse the nested tokens when you analyze the captured performance snapshot using the Performance Analysis tool.

```
public String customFunction() {
try {  PerfUtil.actionStart(customToken);          try {
    PerfUtil.actionStart(customNestedToken1);      ...
    <custom_business_logic>
    ...
}
finally {
    PerfUtil.actionEnd(customNestedToken1);      }
...
<custom_business_logic>
...
try {
    PerfUtil.actionStart(customNestedToken2);      ...
    <custom_business_logic>
    ...
}
finally {
```



```
    PerfUtil.actionEnd(customNestedToken2);    }  
    return OUTCOME_OK;  
}  
finally {    PerfUtil.actionEnd(customToken);    }  
}
```

For more information about creating custom tokens, see [“Creating Custom Tokens for Performance Monitoring” on page 177](#). For more information about measuring the performance using custom tokens, see [Using and Extending the MWS Performance Service](#).

Creating Custom Tokens for Performance Monitoring

You can create and use custom performance tokens to measure the performance of a specific portlet, bean, class, or method. For more information, see [“Measuring Application Performance” on page 176](#).

➤ To create and use custom performance tokens

1. Open the application that you want to extend with custom tokens.
2. Open the Java class source that you want to profile and then define and initialize your custom performance tokens:

```
Object customToken=PerfUtil.actionInit(<category>,<tokenName>);
```

3. Locate the functionality that you want to measure and then collect the performance information.
4. Deploy and run the portlet on an instance of My webMethods Server.
5. Analyze the performance of your application.

19 Using Attachments Providers

■ About Attachments Providers	180
■ Adding an Attachments Provider Property	180
■ Adding a Shared Managed Bean and Attachments Provider Reference	181

About Attachments Providers

Using Composite Application Framework Data controls located in the Palette view, you can add attachment providers to your Web or portlet application. In the View Declaration Language (VDL) file for a view, use an Attachments List Panel control inside of a Form control. For more information about the Attachments List Panel and Form controls, see *webMethods CAF Tag Library Reference*.

In the Attachments List Panel control, you can use one of the following attachments providers:

- **Local Temporary Attachments Provider**,
`com.webmethods.caf.faces.data.attachments.LocalTempAttachmentsProvider`, stores the attached files in a temporary folder in the My webMethods Server file system, in a random folder located at `MWS/server/default/temp`.
- **Portal Container Attachments Provider**,
`com.webmethods.caf.faces.data.attachments.PortalContainerAttachmentsProvider`, stores the attached files in a My webMethods Server specified folder. You must use a portlet application for the Portal Container Provider.
- **Temporary Attachments Provider**,
`com.webmethods.caf.faces.data.attachments.TempAttachmentsProvider`, stores the attached files in a temporary container until the storage location is available in the application.

The attachments providers implement `IAttachmentsProvider`. For more information about `IAttachmentsProvider`, see *webMethods CAF and My webMethods Server Java API Reference*. To use attachments providers in your Composite Application Framework web or portlet application, see [“Adding an Attachments Provider Property” on page 180](#) and [“Adding a Shared Managed Bean and Attachments Provider Reference” on page 181](#).

Adding an Attachments Provider Property

Creates a simple property for the attachments provider. You can use this procedure with any of the available attachments providers. When using the Portal Container Attachments Provider, you must also create a binding expression to the container ID for the attachment storage. For more information, see [“Binding a Managed Bean Property to Another Bean” on page 153](#).

For information about the available attachments providers, see [“About Attachments Providers” on page 180](#). If you want to create a shared managed and reference to the attachments provider, see [“Adding a Shared Managed Bean and Attachments Provider Reference” on page 181](#).

➤ To specify a local temporary attachments provider

1. In a Composite Application Framework portlet project, drag the view to modify to the design canvas.
2. (Optional) In the Palette view, expand **CAF Html > Control > Command**, and then drag a Form control to the view in the design canvas.

If the view already contains a Form control, you can skip this step.

3. In the Palette view, expand **CAF Html > Control > Panel**, and then drag an Attachments List control to the Form control in the design canvas.
4. In the Palette view, expand **Data > Attachments**, and then drag a Local Temporary Attachments Provider to the Attachments List.
5. In **Choose Data Target**, click **Add new simple property to Localtemporaryattachment/default**, and then click **Next**.
6. (Optional) In **Add New Property**, click **Property is writable (setter will be generated)**, and then click **Finish**.

Composite Application Framework adds the attachments provider property to the view Managed Bean.

Adding a Shared Managed Bean and Attachments Provider Reference

This procedure creates an attachments provider shared managed bean and reference in an Attachments List Panel control. For more information about the Attachments List Panel control, see *webMethods CAF Tag Library Reference*.

You can use this procedure with any of the available attachments providers. When using the Portal Container Attachments Provider, you must also create a binding expression to the container ID for the attachment storage. For more information, see [“Binding a Managed Bean Property to Another Bean” on page 153](#).

For information about the available attachments providers, see [“About Attachments Providers” on page 180](#). If you want to create a simple property instead, see [“Adding an Attachments Provider Property” on page 180](#).

➤ To specify a shared managed bean and attachments provider reference

1. In a Composite Application Framework portlet project, drag the view to modify to the design canvas.
2. (Optional) In the Palette view, expand **CAF Html > Control > Command**, and then drag a Form control to the view in the design canvas.

If the view already contains a Form control, you can skip this step.

3. In the Palette view, expand **CAF Html > Control > Panel**, and then drag an Attachments List control to the Form control in the design canvas.
4. In the Palette view, expand **Data > Attachments**, and then drag a Local Temporary Attachments Provider to the Attachments List.

5. In **Choose Data Target**, click **Add new shared managed bean and reference to Localtemporaryattachments/default**, and then click **Next**.
6. (Optional) In **Managed Bean**, use the default managed bean name or in the **Managed Bean Name** field type a new name for the bean.
7. (Optional) Use the default scope or select a scope from the **Managed Bean Scope** list, and then click
8. In **Java Type**, click **Next**.
9. (Optional) In **Add Managed Bean Reference Property to Other Managed Beans**, select a managed bean or page bean.
10. (Optional) Click **Property is writable (setter will be generated)**, and then click **Finish**.

20 Working with JCR Providers

■ About JCR Client Providers	184
■ Enabling the JCR Client for a CAF Application	184
■ Creating a JCR Client Attachments Provider	185
■ Creating a Node Children Table Provider	185
■ Adding Additional Properties to the JCR Provider	186
■ Creating a JCR Client Node Provider	187
■ Creating a Search Result Table Provider	188
■ Creating a JCR Sub-Folder Attachments Provider	188
■ Creating a JCR Client Temp Attachments Provider	190

About JCR Client Providers

The Java Content Repository (JCR) client providers are controls that function in different containers as well as in different implementations of a container.

When you enable the JCR client, the Faces Configuration file automatically includes two managed beans: `repositorySessionManager` and `jcrCredentials`.

The `repositorySessionManager` managed bean provides a connection to the remote repository and obtains a session to use throughout the API. The `repositorySessionManager` references `jcrCredentials` and specifies whether to impersonate a user.

The `jcrCredentials` managed bean provides the necessary login information to access the remote repository, including the `userId` and `password`. The credentials are bound to the environment variables `systemUserId` and `systemPassword`. You need to update the environment variables if you decide to run the application on a container other than My webMethods Server.

The following table lists the available JCR providers.

Provider	Description
Attachments Provider	Uses a JCR Node as an attachment container.
Node Children Table Provider	Displays the children of a JCR Node in a table.
Node Provider	Updates the properties of a folder, or updates the properties of items in a folder.
Search Result Table Provider	Displays the results of a JCR query in a table.
Sub-Folder Attachments Provider	Uses a JCR Node as an attachment container.
Temp Attachments Provider	Uses a JCR Node as a temporary attachment container.

Enabling the JCR Client for a CAF Application

You must enable the JCR client for your Composite Application Framework application before you can use a JCR provider.

To enable the JCR client for a CAF application

1. In the **UI Development Perspective** on an opened application, click the **Project Explorer** view, right-click the Composite Application Framework application and then click **Properties**.
2. In **Properties**, expand **Project Facets**, select **JavaServer Faces**, select **CAF JCR Client** check box, and then click **OK**.

The JCR Client Providers are listed under the **Data > JCR** in the Palette view.

Creating a JCR Client Attachments Provider

The JCR Client Attachments Provider enables you to create a folder in My webMethods Server, or other JCR, that you can use to open, add, delete, or modify files.

You must enable the JCR client for your Composite Application Framework application before you can use a JCR provider. For more information, see [“Enabling the JCR Client for a CAF Application” on page 184](#).

The following table lists the components of a Attachments Provider and the actions that the Composite Application Framework performs when you add a provider:

Component	Description
New provider	A new provider is added to the Bindings view and configured to work with the repository manager in the current session.
Auto Save	The Auto Save setting for the provider is set to <code>true</code> , which automatically saves changes.
Parent ID	When you drag a folder from the My webMethods Server view, the Parent ID is set to the My webMethods Server folder that is used. If you drag a folder from the Palette view, you need to set this property manually.
Attachments List control	An Attachments List control is added to the user interface.
Attachments List control Drag and Drop Available property	The Attachments List control Drag and Drop Available property is set to <code>true</code> , which enables a user to add an attachment by dragging it onto the user interface.
Attachments List control Use WebDav Urls property	The Attachments List control Use WebDav Urls property is set to <code>true</code> .

➤ To use the Attachments Provider

1. In the **UI Development Perspective**, click the MWS Admin view or Palette view, and then drag a folder to the design canvas.
2. In **Drop Candidates**, click **Create Content Provider**, and then click **OK**.
3. In **Java Type**, click **Attachments Provider**, and then click **Finish**.

Creating a Node Children Table Provider

The JCR Client Attachments Provider enables you to access a folder in My webMethods Server, or other JCR that includes detailed information about the contents of the folder. You can also open, add, delete, or modify the files contained in the folder.

You must enable the JCR client for your CAF application before you can use a JCR provider. See [“Enabling the JCR Client for a CAF Application” on page 184](#)

The following table lists the components of a Node Children Table Provider and the actions that the Composite Application Framework performs when you add a provider:

Component	Description
New provider	A new provider is added to the Bindings view and configured to work with the repository manager in the current session.
Auto Save	The Auto Save setting for the provider is set to <code>true</code> , which automatically saves changes.
Parent ID	When you drag a folder from the My webMethods Server view, the Parent ID is set to the My webMethods Server folder that is used. If you drag a folder from the Palette view, you need to set this property manually.
Table	A Table is added to the user interface that includes default columns of Path, Name, and ID.

➤ To use the Node Children Table Provider

1. In the MWS Admin view or Palette view, drag a folder onto the design canvas.
2. In the Drop Candidates dialog box, click **Create Content Provider**, and then click **OK**.
3. In the Java Type dialog box, click **Node Children Table Provider**, and then click **Finish**.

To add additional properties, see [“Adding Additional Properties to the JCR Provider” on page 186](#).

Adding Additional Properties to the JCR Provider

You must enable the JCR client for your CAF application before you can use a JCR provider. For more information, see [“Enabling the JCR Client for a CAF Application” on page 184](#).

➤ To add additional properties to the provider

1. In the Bindings view, under **Controls Scoped Variables**, right-click the provider node, point to **Add**, and then click **Import Property Names**.
2. In the **RepositoryURL** text box, type the URL to the repository.

Note:

The default URL points to My webMethods Server running on localhost:8585. You can set the URL can point to any JCR provider.

3. In the **UserName** text box, type the user name.

4. In the **Password** text box, type the password, and then click **Next**.
5. Select the node type property that you want to add to the provider and display in the table control, and then click **Finish**. The properties are added under the **Controls Scoped Variables** node.
6. Drag the property that you want onto the existing table control to add a new column.

Creating a JCR Client Node Provider

The JCR Client Node Provider enables you to update the properties of a JCR node, for example, the description of a folder, or update the properties of items in the folder.

You must enable the JCR client for your CAF application before you can use a JCR provider. For more information, see [“Enabling the JCR Client for a CAF Application” on page 184](#).

When you create a node provider, you drag a folder from the MWS Admin view or Palette view into the Bindings view to add a provider to a managed bean, instead of dragging a folder onto the design canvas.

➤ To use the Node Provider

1. In the **UI Development** perspective, click the MWS Admin view or Palette view, select drag the folder containing the items with the properties that you want to update to the managed bean in the **Bindings** view.
2. Click **Node Provider**, and then click **Finish**.
3. In the Bindings View, right-click the new provider, click **Add**, and then click **Import Property Names**.
4. In the **RepositoryURL** text box, type the URL to the repository.
5. In the **UserName** text box, type the user name.
6. In the **Password** text box, type the password, and then click **Next**.
7. Select the JCR node type property that you want to update, and then click **Finish**. The properties are added to the Bindings view under the new provider.
8. Drag the property that you want to update into the form on the design canvas.
9. Drag **Node > Save()** into the form on the design canvas.

Creating a Search Result Table Provider

The JCR Client Attachments Provider enables you to access and search for items within a JCR. You can also open, delete, or modify the files obtained by the search results.

You must enable the JCR client for your CAF application before you can use a JCR provider. For more information, see [“Enabling the JCR Client for a CAF Application” on page 184](#).

The following table lists the components of a Search Results Table Provider and the actions that the Composite Application Framework performs when you add a provider:

Component	Description
New provider	A new provider is added to the Bindings view and configured to work with the repository manager in the current session.
Table	A Table is added to the user interface that includes default columns of Path, Name, and ID.
Query language	The default provider query language is xpath. You can use either xpath or SQL.
Query string	The default provider query string includes all children within the parent, for example, all files within a folder.

➤ To use the Search Result Table Provider

1. In the MWS Admin view or Palette view, drag a folder onto the design canvas.
2. In the Drop Candidates dialog box, click **Create Content Provider**, and then click **OK**.
3. In the Java Type dialog box, click **Search Result Table Provider**, and then click **Finish**.

On the Palette view, drag the **CAF Html > Control > Input > Filter Input** control onto the design canvas, and then, if the control is outside of the table, point the control to the table. Completing this step provides type-ahead filtering of the results table.

Creating a JCR Sub-Folder Attachments Provider

The JCR Client Sub-Folder Attachments Provider enables you to create a sub-folder in the My webMethods Server Administrator view that you can use to open, add, delete, or modify files.

This provider can be used to submit a document to start a business, to upload a document to My webMethods Server, create a unique folder for the process, and pass the folder as business data for the process. Tasks can then use the document as an attachment.

You must enable the JCR client for your CAF application before you can use a JCR provider. For more information, see [“Enabling the JCR Client for a CAF Application” on page 184](#).

The following table lists the components of a Sub-Folder Attachments Provider and the actions that the Composite Application Framework performs when you add a provider:

Component	Description
New provider	A new provider is added to the Bindings view and configured to work with the repository manager in the current session.
Sub Folder Name property	A Sub Folder Name property, bound to the user's login name by default, is added to the provider in the Bindings view, which specifies the name of the sub-folder to be created in the target folder. Note: The Sub Folder Name property value must be a unique name.
Auto Save	The Auto Save setting for the provider is set to <code>true</code> , which automatically saves changes.
Parent ID	The Parent ID is set to the My webMethods Server folder that is used.
Attachments List control	An Attachments List control is added to the user interface.
Attachments List control	An Attachments List control is added to the user interface.
Attachments List control Drag and Drop Available property	The Attachments List control Drag and Drop Available property is set to <code>true</code> , which enables a user to add an attachment by dragging it onto the user interface.
Attachments List control Use WebDav Urls property	The Attachments List control Use WebDav Urls property is set to <code>true</code> .

➤ To use the Sub-Folder Attachments Provider

1. In the MWS Admin view or Palette view, drag a folder onto the design canvas.
2. In the Drop Candidates dialog box, click **Create Content Provider**, and then click **OK**.
3. In the Java Type dialog box, click **Sub-Folder Attachments Provider**, and then click **Finish**.
4. In the Bindings view, under the provider node, click **Sub Folder Name**.
5. In the Properties view, click the **Data Binding** tab, and then, in the **Value** text box, type a unique name or bind to an expression that resolves into a unique name.

Creating a JCR Client Temp Attachments Provider

The JCR Client Temp Attachments Provider enables you to move or copy files into a designated folder in the My webMethods Server Administrator view. This provider creates an empty temporary folder to store uploaded files, and then deletes the folder when it is no longer used, typically when the managed bean goes out of scope. If the uploaded files are not copied or moved to the target folder, they are also deleted.

You must enable the JCR client for your CAF application before you can use a JCR provider. For more information, see [“Enabling the JCR Client for a CAF Application” on page 184](#).

The following table lists the components of a Temp Attachments Provider and the actions that the Composite Application Framework performs when you add a provider:

Component	Description
New provider	A new provider is added to the Bindings view and configured to work with the repository manager in the current session.
Migration Provider	Within the new provider in the Bindings view, Composite Application Framework creates a Migration Provider that includes:
Move Attachments()	Moves attached files from a source folder into a designated folder.
Copy Attachments()	Copies attached files in a source folder into a designated folder.
Attachments List control	An Attachments List control is added to the user interface.
Attachments List control Drag and Drop Available property	The Attachments List control Drag and Drop Available property is set to <code>true</code> , which enables a user to add an attachment by dragging it onto the user interface.
Attachments List control Use WebDav Urls property	The Attachments List control Use WebDav Urls property is set to <code>true</code> .

➤ To use the Temp Attachments Provider

1. In the MWS Admin view or Palette view, drag a folder onto the design canvas.
2. In the Drop Candidates dialog box, click **Create Content Provider**, and then click **OK**.
3. In the Java Type dialog box, click **Temp Attachments Provider**, and then click **Finish**.

4. Expand the new provider, expand **Migration Provider**, and then drag **Move Attachments()** or **Copy Attachments()** onto the design canvas. A button to move or copy attached files is added to the user interface.

21 Working with Web Services

■ About Web Services in CAF Applications	194
■ Connecting to webMethods Integration Server	194
■ Specifying a Web Service Connector	195
■ Setting Web Service Connector Preferences	199
■ Specifying a Web Service Connector Socket Timeout	200
■ Migrating a Web Service from Glue to WS-Stack	200
■ Configuring wsclient-socketTimeout at Run Time	202
■ Configuring Global Runtime Timeout	203
■ Web Service Connector Preferences	203
■ About XSD Schema Choice Declarations	204

About Web Services in CAF Applications

When you want to use a web service in your Composite Application Framework web or portlet application, you must implement a web service connector within the application. A web service connector invokes a web service located on a remote server.

The web service connector sends messages using HTTP or HTTPS to the webMethods Integration Server, invoking the web service. The Integration Server hosts packages that contain web services and related files, authenticates clients, and verifies that they are authorized to execute the requested service. For more information about working with web services, see *Web Services Developer's Guide*.

To add Integration Server-based web services into your Composite Application Framework application, Designer must be connected to an instance of the Integration Server containing the resources for the web service.

Connecting to webMethods Integration Server

To implement a web service in your Composite Application Framework application, you must create a web service connector within your application. However, you must create a connection from Designer to an instance of Integration Server before you can create a web service connector.

You must have a valid user account on the Integration Server before you can complete this task. The account must be configured with the Access Control Lists (ACLs) required to access and work with the Integration Server packages and services that you want to use. For more information about working with ACLs, see *webMethods Integration Server Administrator's Guide*.

» To connect to a webMethods Integration Server instance

1. In the Package Navigator view in Designer, click the **Add or modify Integration Servers** button in the view toolbar.
2. In the Preferences dialog box, click **Add** and specify the following information for the server instance:

Field	Description
Name	The display name to use for the Integration Server instance. Note: The name cannot contain control characters, special characters, and characters outside of the basic ASCII character set, such as multi-byte characters.
Host	The host name or IP address of the Integration Server to connect to, for example workstation5.webmethods.com or 132.906.19.22. Do not begin the host name with http://.
Port	The port number of the Integration Server.

Field	Description
User	The name of a valid user account on the Integration Server that is configured with the appropriate access permissions.
Password	The password for the user account.

3. Select any or all of the following options:

Option	Description
Connect immediately	Indicates whether Designer should connect to the Integration Server immediately after you add the server definition.
Connect at startup	Indicates whether Designer should automatically connect to the Integration Server each time you start Designer.
Secure connection	Indicates whether the session will be opened through HTTP or HTTPS. If you want to open an HTTPS session on the Integration Server using the Secure Socket Layer (SSL) protocol, select this check box.

4. To test the connection, click **Verify Server**.

5. Click **OK**.

Specifying a Web Service Connector

When you want to use a web service in your Composite Application Framework web or portlet application, you must create a web service connector and specify a web service client. You can add a web service connector in the following ways:

- Use the **New > Web Services Connector** wizard, as described below. You can use this method to select a WSDL file from your local file system, or to create a web service connector from an Integration Server service or web service descriptor.
- Use the drag and drop method. For more information, see [“Adding a Web Service Connector from Package Navigator” on page 198](#).
- Add a Java class reference to a managed bean. For more information, see [“Generating a Web Service Connector Reference” on page 197](#).

Important:

The user account used to connect to the Integration Server must have appropriate ACL permissions to access and work with the targeted services used in the following procedure.

- To specify a web service client with the New Web Service Client wizard

1. In the UI Development perspective, click the Navigator view and locate the web or portlet application project you want to work with.
 2. Right-click the project and select **New > Web Service Connector**.
 3. In the **New Web Service Client** wizard, select a project from the **Project** list and click **Next**.
 4. To specify a web service, do one of the following:
 - Click the **WSDL** list and select **Browse to a Local File** to select a WSDL file that represents a web service.
 - Click the **WSDL** list and select **Choose a Web Service from a Data Provider** to choose a service or web service descriptor from a data provider:
 1. In the **Select Web Service** dialog box, expand the webMethods Integration Server instance you want to work with until you find the package and service or web service descriptor you want to use, select it, and then click **OK**. For information about creating a web service descriptor, see [“Creating a Web Service Descriptor” on page 196](#).
- The **Client Name** and **Package** fields display the selected object’s information.
5. In the **Authentication Method** list, select the desired authentication mode.

Note:

If you select the **Hybrid** authentication mode, SAML 2.0 is used by default. To use SAML Artifact Profile, set the `wsclient.hybrid.use.samlartifact` property to `true` in the `custom_wrapper.conf` file in the following My webMethods Server directory: *Software AG_directory/profiles/MWS_instanceName/configuration*.

6. In the **Soap Library** list, select **WS-Stack Client** and:
 - If the selected service or descriptor has two or more operations, click **Next** to specify the operation you want to use.
 - Click **Next** to work with additional web service connector settings, or click **Finish** to add the web service connector.

The new web service client is added to the application and can be viewed in the **User Interfaces** node of the Solutions view and in the **Managed Beans** node of the Bindings view. You can also view the `wsclient.xml` file in the project’s **WEB-INF** folder in the Navigator view.

Creating a Web Service Descriptor

If you attempt to create a web service connector by dragging a service from the Package Navigator view and dropping it into a view that is open in the editor (or into the **Managed Beans** node of the Bindings view), you will receive an error message stating that this method is deprecated and is not considered best practice.

Instead, you must first create a web service descriptor (WSD), which you can then drag and drop into your project. For more information about web service descriptors, see *Web Services Developer's Guide*.

➤ To create a web service descriptor

1. In the UI Development perspective, click the Package Management view.
2. In the Package Navigator view, expand the webMethods Integration Server instance, right-click the service you want to use to generate a web service descriptor, and click **Generate Provider WSD**.
3. Accept the default name for the web service descriptor, or type a new name.
4. Click **OK**.

The new web service descriptor is created in the Package Navigator folder that contains the service. You can now drag and drop the descriptor into your project using one of the following methods, as appropriate:

- [“Specifying a Web Service Connector” on page 195](#), to add a web service client to the application.
- [“Generating a Web Service Connector Reference” on page 197](#), to manually create the user interface for a web service by creating a web service connector reference in a managed bean.
- [“Adding a Web Service Connector from Package Navigator” on page 198](#), to add a web service client to a portlet view or managed bean node.

Generating a Web Service Connector Reference

When you want to manually create the user interface for a web service, you can create a web service connector reference in the managed bean. This approach only creates a connector Java class.

If you want to add the web service connector to the user interface, see [“Adding a Web Service Connector from Package Navigator” on page 198](#).

Note:

Dragging and dropping a service from the Package Navigator view into the **Managed Beans** node of the Bindings view is not supported. You must first create a web service descriptor, as described in [“Creating a Web Service Descriptor” on page 196](#).

You must connect to a webMethods Integration Server to use this procedure, and the connecting user account must have appropriate ACL permissions. For more information, see [“Connecting to webMethods Integration Server” on page 194](#).

➤ To generate a web service connector reference

1. In the UI Development perspective, open a web or portlet application project, then open the portlet view you want to work with in the editor.
2. In the Package Navigator view, expand the webMethods Integration Server instance and select the web service descriptor you want to reference in your application.
3. Click the Bindings view and expand **Managed Beans**.
4. Drag the web service descriptor to the web or portlet application listed under the **Managed Beans** node.
5. In the New Web Service Client wizard, select an authentication method and click **Next**.
6. Select the web service operations you want to use and click **Finish**.

The new web service client is added to the application and can be viewed in the Bindings view, as well as in **User Interfaces** node of the Solutions view. You can also view the `wsclient.xml` file in the project's **WEB-INF** folder in the Navigator view.

Adding a Web Service Connector from Package Navigator

When you want to use web services in your Composite Application Framework web or portlet application, you must create a web services connector in the application and select a web services client.

You can create a web services descriptor in the Package Navigator view, and then drag it into a portlet view that is open in the editor. For more information, see [“Specifying a Web Service Connector” on page 195](#). In this case, any user interface elements in the web service are added to the View Declaration Language (VDL) file open in the editor.

Note:

Dragging and dropping a service from the Package Navigator view into a view in the editor is not supported. You must first create a web service descriptor, as described in [“Creating a Web Service Descriptor” on page 196](#).

You must connect to a webMethods Integration Server to use this procedure, and the connecting user account must have appropriate ACL permissions. For more information, see [“Connecting to webMethods Integration Server” on page 194](#).

➤ To create a web service connector by dragging from the Package Navigator view

1. In the UI Development perspective, open the web or portlet application that you want to work with.
2. Open the portlet view you want to work with in the editor.

3. In the Package Navigator view, expand the webMethods Integration Server instance, select the web service descriptor you want to work with and drag it into a view in the design canvas.
4. In the New Web Service Client wizard, select an authentication methods and click **Next**.
5. Select the web service operations you want to use and click **Finish**.

The new web service client is added to the application and can be viewed in the **User Interfaces** node of the Solutions view. You can also view the `wsclient.xml` file in the project's **WEB-INF** folder in the Navigator view. Click the **Preview** tab to view any user interface elements added by the web service.

Adding an Operation from a Referenced Web Service

Some web services contain two or more operations. It is possible for a project to contain a web service connector that represents only one of the operations available in the provider service. In a situation like this, you might want to add to your project another web service connector that represents another operation available in an already-referenced web service provider.

If you attempt to create the new web service connector using the New Web Service Client wizard (as described in [“Specifying a Web Service Connector” on page 195](#)), you will receive an error, “Client with this name already exists.”

➤ To add an operation from a web service that is already referenced in your project

1. Expand the project in the **User Interfaces** node of the Solutions view.
2. In the **Web Services Connectors** folder, expand the existing web service connector so that you can see all of the available operations within it.
3. Right-click the operation you want to add as a new web service connector and click **Generate Web Service Connector**.
4. In the New Web Service Client wizard, click **Finish**.

Setting Web Service Connector Preferences

You can specify preference settings for using web services in your web or portlet application.

➤ To set web service connector preferences

1. In Software AG Designer, click **Window > Preferences**.
2. In the Preferences dialog box, expand **Software AG > UI Development > Web Service Connector**.

3. In the **Web Service Connector** panel, select or clear the check box next to the preference, and then click **OK**.

For explanations of the available preferences, see [“Web Service Connector Preferences” on page 203](#).

A check in the check box indicates that the preference is set. Remove the check to disable the preference.

Specifying a Web Service Connector Socket Timeout

While designing a web service connector, you can specify the timeout interval using the socket timeout property in an individual web service connector. The default value for socket timeout is null (no value). This value is *not* overwritten by the `wsclient-timeout` environment variable. For more information, see [“Configuring wsclient-socketTimeout at Run Time” on page 202](#).

Specify a value in milliseconds to define the socket timeout. For example, 1 minute is 60000 milliseconds. If you specify an empty value, the system administrator cannot update the timeout value at runtime.

➤ To specify the socket timeout for an individual web service connector

1. In the UI Development perspective, open a web or portlet application project, expand the **WebContent** folder, and open the view that uses the web services client in the editor.
2. In the **Bindings** view, click **Show Expert Properties** in the toolbar, expand the **Managed Beans** node, and expand the web service.
3. Click **Socket Timeout**.
4. In the **Properties** view, click the **Data Binding** tab.
5. In the **Value** field, either type a numerical timeout value using milliseconds, or define a binding expression that results in a timeout value.

Migrating a Web Service from Glue to WS-Stack

Use the following procedure to migrate a web service from the Glue Client library, which is deprecated, to the default WS-Stack Client library.

➤ To migrate from Glue WS Client to WS-Stack Client

1. In the UI Development perspective, click the Project Explorer view and locate the CAF application project to migrate.
2. Expand the **WebContent** folder, and then expand the **WEB-INF** folder.

3. Double-click the `wsclients.xml` file.
4. Click the **Source** tab and change the value of the `soapLibrary` attribute from `glue` to `axis`.


```
soapLibrary="axis"
```
5. Delete the Glue files from the project by doing any of the following:
 - Click the **Design** tab, select all `generatedfile` entries for Glue in the **Node** column, then right-click and select **Remove**.
 - In the Navigator view, expand the project, go to **src > caf > war**, and then delete the folder with the Glue files. You can find the file path of the Glue folder next to each file listed on the **Design** tab.
6. In the Package Navigator view, expand the Integration Server instance, right-click the web service descriptor with which you want to work and select **CAF > Generate Web Service Connector**.
7. In the New Web Service Client wizard, select the project that you are migrating and click **Next**.
WS-Stack Client is selected in the **SOAP Library** list.
8. Click **Finish** to generate the web service connector for the project.
9. Modify the portlet view(s) in the project that you are migrating:
 - a. Expand the **WebContent** folder and double-click the portlet folder.
 - b. Right-click the `default.view` file and select **Open With > Text Editor**.

In Glue projects, properties in the `default.view` file are defined with a single qualifier, such as `WebServiceConnectorName.input`, for example:

```
<property name='value' value='#{<PortletName>DefaultviewView.  
WebServiceConnectorName.parameters.  
WebServiceConnectorName.input}' />
```

In WS-Stack projects, properties are defined with an additional qualifier, such as `WebServiceConnectorName.WebServiceConnectorName.input`, for example:

```
<property name='value' value='#{<PortletName>DefaultviewView.  
WebServiceConnectorName.parameters.  
WebServiceConnectorName.WebServiceConnectorName.input}' />
```

- c. Add the additional qualifier for all input properties in the text editor.

The following example shows three input properties with additional qualifiers.

```

    <property name='id' value='propertyLine' />
    <property name='label' value='Input' />
    <control component-type='javax.faces.HtmlInputText' renderer-type='com.webmethods.caf.faces.Text'>
      <property name='id' value='htmlInputText' />
      <property name='value' value='${TestProtletDefaultviewView.testExample.parameters.testExample.testExample.doc1.input}' />
      <property name='width' value='input20' />
    </control>
  </control>
</control>
<control component-type='com.webmethods.caf.faces.panel.PropertyGroup' renderer-type='com.webmethods.caf.faces.panel.PropertyGroup'>
  <property name='id' value='propertyGroup4' />
  <property name='label' value='Doc 2' />
  <control component-type='com.webmethods.caf.faces.panel.PropertyLine' renderer-type='com.webmethods.caf.faces.panel.PropertyLine'>
    <property name='id' value='propertyLine1' />
    <property name='label' value='Input 1' />
    <control component-type='javax.faces.HtmlInputText' renderer-type='com.webmethods.caf.faces.Text'>
      <property name='id' value='htmlInputText1' />
      <property name='value' value='${TestProtletDefaultviewView.testExample.parameters.testExample.testExample.doc2.input1}' />
      <property name='width' value='input20' />
    </control>
  </control>
</control>
<control component-type='com.webmethods.caf.faces.panel.PropertyLine' renderer-type='com.webmethods.caf.faces.panel.PropertyLine'>
  <property name='id' value='propertyLine2' />
  <property name='label' value='Test String' />
  <control component-type='javax.faces.HtmlInputText' renderer-type='com.webmethods.caf.faces.Text'>
    <property name='id' value='htmlInputText2' />
    <property name='value' value='${TestProtletDefaultviewView.testExample.parameters.testExample.testExample.testString}' />
    <property name='width' value='input20' />
  </control>
</control>
</control>

```

10. Replace all instances of `.result.` with `.result.ISServicenameOperationName.Output` in all output properties, for example:

```

<property name='value' value='${TestProtletDefaultviewView.testExample
.result.testExample.outString}' />

```

11. Click **File > Save**.
12. Deploy the project to My webMethods Server.

Configuring wsclient-socketTimeout at Run Time

If you specify a timeout value in the Composite Application Framework (CAF) Application Runtime Configuration using the *wsclient-timeout* environment variable, this timeout values overwrites any timeout values specified in other applications. However, the web service connector Socket Timeout value is *not* overwritten when you specify a *wsclient-timeout* value using the CAF Application Runtime Configuration.

The *wsclient-timeout* environment variable in the `web.xml` file provides the default value for the runtime Web Application Runtime Configuration, as well as default value when no value is provided in the Global Defaults on My webMethods Server.

➤ To configure timeout for deployed CAF applications

1. Log on to the My webMethods Server instance using system administrator credentials.
2. In **Administrative Dashboard**, in the **Configuration** panel, click **CAF Application Runtime Configuration**.
3. In **Keywords**, type search parameters to locate the deployed application to configure.

4. In the search results, click the application name in the **APPLICATION** column.
5. In the **Application Configuration** group, expand the **Web Application** node and click **Environment Entries**.
6. For the *wsclient-socketTimeout* environment variable, type a number in milliseconds in the **Value** column and click **Apply**.

Configuring Global Runtime Timeout

When runtime or design-time timeout values are not specified in a Composite Application Framework web service connector or in the web service client, the global default value is 1 minute or 60000 milliseconds.

You can update the global runtime timeout to use a different value. You might want to use the global runtime environment variable when different applications require different timeout intervals.

For example, when one application calls the local Integration Server services, you can use a short timeout value. If a different application calls remote web services from the third-party provider over the Internet, you might want to define a longer timeout. When web applications require a longer timeout interval, define the socket timeout property. For more information, see [“Specifying a Web Service Connector Socket Timeout” on page 200](#).

You must have system administrator privileges on the My webMethods Server instance to change the global timeout variable.

➤ To configure the global timeout variable

1. Log on to the My webMethods Server instance using system administrator credentials.
2. In **Administrative Dashboard**, in the **Configuration** panel, click **CAF Application Runtime Configuration**.
3. In **CAF Application Runtime Configuration**, click **Configure Global Defaults**.
4. In the **Application Configuration** group, expand **Web Application**, and click **Environment Entries**.
5. For the *wsclient-socketTimeout* environment variable, type a number in milliseconds in the **Value** column and click **Apply**.

Web Service Connector Preferences

Software AG Designer provides several web service connector preferences as described below. For information about how to set these preferences, see [“Setting Web Service Connector Preferences” on page 199](#). The following table lists the available web service connector preferences:

Preference	Description
Silently create Web Service Connector on Drag-and-Drop	Enables you to automatically create a web service connector when you drag an IS service from the Package Navigator view onto the design canvas. Selected by default.
Show warning for creating web service connector from a deprecated source	Displays a message whenever the source for the web service connector is deprecated. A deprecated source is a Flow, Java, or other type of IS service. Selected by default.
Use WSDL schema name space values for generating java package names	Resolves name space conflicts when the same type name exists in multiple XML name spaces. Not selected by default.
Automatically initialize objects for optional input fields	Sets optional input fields to null. If later you need to populate one of the optional input fields, you must manually create the object, and then set it on the parent object using custom Java code. Selected by default.
Default SOAP Library	Sets the library which is used for generating the web services stubs. You can select between WS-Stack Client (default) and Glue WS Client (deprecated).

About XSD Schema Choice Declarations

When a service request element XSD schema contains a complex type with a choice declaration, all potential choice elements are considered optional. When the choice elements are of complex type, apart from populating the fields of the optional element, you must also initialize the optional element itself by using the setter method of the parent bean.

For example:

```
<element name="Parent">
  <complexType>
    <choice>
      <element ref="tns:Red"/>
      <element ref="tns:Blue"/ >
    </choice >
  </complexType>
</element >
<element name="Red" >
  <complexType >
    <sequence >
      <element name="redField" type="string"/ >
    </sequence >
  </complexType >
</element >
<element name="Blue" >
  <complexType >
```

```
        <sequence >
            <element name="blueField" type="string"/ >
        </sequence >
    </complexType >
</element >

Red myRed = new Red() ;
myRed.setRedField("test") ;
parent.setRed(myRed);
```


22 Using the MWS Admin View

■ About the MWS Admin View	208
■ Creating a MWS Data Provider	209
■ Changing the Connection Properties for a Server Instance	210
■ Opening the MWS Administration Dashboard	210

About the MWS Admin View

The MWS Admin view of Software AG Designer enables you to connect to multiple instances of My webMethods Server as a system administrator. To work with a My webMethods Server instance in the MWS Admin view, the server instance must be running. The following table lists the basic operations that you can perform on server resources after adding a server to the view:

Action	Description
Browse	Opens My webMethods Server in an Eclipse browser window and enables you to browse server resources.
Refresh	Refreshes the contents of the view.
Rename	Enables you to modify the name of the server instance in the view.
Reconnect	Attempts to reconnect to the server instance. You can use this command when the Connect at Startup property is set to <code>false</code> .
Delete	Removes the server instance from the view.
Extract Asset into Project	Extracts My webMethods Server runtime assets into a content or skin project. For more information about extracting My webMethods Server assets, see “About Runtime Asset Extraction” on page 168 .

The following table lists further actions that you can perform on server resources in the MWS Admin view:

Action	Instructions
Delete a portlet or a page	Right-click the portlet and click Delete .
Move a portlet	Drag the portlet to another position within the same page, to a different page, or to a different server.
Copy a portlet	Right-click the portlet and click Edit > Copy . Then right-click another page and click Edit > Paste . You can copy resources from one server to another.
Create server resources	Right-click the node where you want to add a resource, such as folders, links, and portlets, and click New . In the tree view, select the resource that you want to create, type a display name for the resource, and click Finish .
Modify properties of a server resource	Select the server resource, such as a portlet or page. Then go to the Properties view and make changes as needed. For example, you can create or modify aliases for server resources, such as pages or portlets.

Action	Instructions
Publish documents to the server	Drag the document from the Navigator view to the MWS Admin view. You can also drag documents from the MWS Admin view to the Navigator view.


Important:

If you have system administrator privileges, making changes indiscriminately can corrupt other portions of the server.

Creating a MWS Data Provider

Use the following procedure to add a My webMethods Server instance as a data provider in the MWS Admin view.

➤ To connect to My webMethods Server

1. In the MWS Admin view of the UI Development perspective, click .
2. The following table lists the required and optional configurations that you specify when creating a MWS data provider:

Field	Description
Data Provider Name	Required. A descriptive name for the My webMethods Server used by the project.
Data Provider URL	Required. The host name and port of the server to connect to, for example: <code>http://localhost:8585/</code>
Default Path	Required. The highest directory level to use in My webMethods Server. Values are: <ul style="list-style-type: none"> ■ folder.root - This is the default value. ■ folder.public ■ folder.system ■ webm.apps
Connect at startup	Optional. Specifies whether to connect to the My webMethods Server instance when Software AG Designer starts.
Remember Authentication	Optional. Specifies whether to remember the credentials used for connecting to My webMethods Server.

3. Click **Finish**.

Changing the Connection Properties for a Server Instance

You can change the connection properties for a My webMethods Server instance in the MWS Admin view.

➤ To change connection properties for My webMethods Server

1. In the MWS Admin view, click the server node to display the server properties in the Properties view.
2. Perform any of the following steps, as required:
 - Click the **General** tab to change the **Root path** or **Connection URL** property.
 - Click the **Connection Info** tab to change the **Remember authentication credentials** or **Connection at startup** properties.

Opening the MWS Administration Dashboard

From the MWS Admin view, you can open a browser to display the Administration Dashboard for the instance of My webMethods Server. In the Administration Dashboard, you can manage user accounts and perform other system administrator activities. For more information about configuring My webMethods Server, see *Administering My webMethods Server*.

➤ To display My webMethods Server in a browser

1. In the MWS Admin view, right-click the server node and then click **Browse**.
2. If required, provide a valid user name and password.

The specific node within My webMethods Server is displayed in an internal web browser. To specify the use of an external web browser, see [“Specifying an External Web Browser” on page 14](#).

23 Views in the Composite Application Framework

■ The UI Development Perspective	213
■ The Bindings View	215
■ Data Source Explorer View	215
■ Connecting to a Database	215
■ Creating a Database Connector	216
■ Outline View	217
■ Package Navigator View	218
■ Properties View	218
■ Properties View Toolbar	218
■ Snippets View	219
■ Customizing the Snippets View	219
■ E-forms Snippets Drawer	221
■ CAF Dialog Patterns	221
■ Solutions View	221
■ Database Connectors in the Solutions View	221
■ Creating a Portlet on the Solutions View	222
■ Using the Design Canvas	223
■ The Palette View	227
■ CAF Events Editor	230

■ Using Snippets	236
■ Customizing UI Controls	241

The UI Development Perspective

Software AG Designer and Composite Application Framework use the Eclipse platform to provide the tools to create Web and portlet applications. The Eclipse platform is composed of a model layer, known as the workspace containing projects, folders, and files, and a folders, and the workbench layer that defines the presentation for resources. The Eclipse workbench uses the term *perspective* to define a set of the actions, resources, and views available to complete a specific task.

In Composite Application Framework, you use the UI Development Perspective to development of Web and portlet applications. The list describes the brief views and editors used in the Composite Application Framework UI Development Perspective for developing Web and portlet applications projects.

■ Project Explorer

Shows resources in the workspace organized by project. You can access Composite Application Framework project repair tools and external strings for localization.

■ Design Canvas

Provides editing functions for building the user interface for Composite Application Framework Web and portlet applications. You can drag the View Declaration Language (VDL) file for a view to the design canvas, and then drag controls from the **Palette** to the VDL file. For more information, see [“About the Design Canvas” on page 223](#).

■ Palette

Contains the JavaServer Faces user interface controls Composite Application Framework Web and portlet applications. For more information, see [“About the Palette View” on page 227](#).

■ Bindings

Lists the managed beans in the Web application, enables creating new parameters, security roles, and other properties. For more information, see [“The Bindings View” on page 215](#).

■ Properties

Shows the properties for the asset selected in the perspective. You can specify values for a control's properties in the design canvas and objects in the MWS Admin view. For more information, see [“Properties View” on page 218](#).

■ MWS Admin

Displays resources on an instance of My webMethods Server, and enables access to environment configuration and monitoring tools. For more information, see [“About the MWS Admin View” on page 208](#).

■ Servers

Lists the runtime environments configured and associated with Composite Application Framework. You can publish a web application to a supported runtime server. Composite

Application Framework portlet applications are only supported on My webMethods Server. For more information, see [“Working with My webMethods Server” on page 159](#).

■ **Package Navigator**

Displays of resources connected to an Integration Server instance. For more information, see [“Package Navigator View” on page 218](#). For information on connecting to an Integration Server instance, see [“Connecting to webMethods Integration Server” on page 194](#).

■ **Data Source Explorer**

Lists data sources and database connections you can associate with components in a Web or portlet application. For more information, see [“Data Source Explorer View” on page 215](#).

■ **Snippets**

Consolidates default complex controls as snippets providing usable components for your applications. For more information, see [“Data Source Explorer View” on page 215](#).

■ **Saved Searches**

Shows a list of saved searches. You can save searches regularly performed for faster access to the search results. For more information about using saved searches, see *Administering My webMethods Server*.

■ **Outline**

Shows the structure of the asset open in the design canvas. For more information, see [“Outline View” on page 217](#).

■ **Solutions**

Shows webMethods Product Suite assets in your workspace, and provides a quick way to see important assets in a Web or portlet application project. For more information, see [“Solutions View” on page 221](#).

■ **Navigator**

Shows the assets in the workspace divided into project node trees. You can add Web and portlet application projects, portlets, views, Web service connectors, and edit resources. This default Eclipse view enables opening files for editing.

■ **Problems**

Displays system-generated errors, warnings, or information associated with a resource in a default Eclipse view. This is default Eclipse view.

■ **Search**

Displays the results of a search in a default Eclipse Search view.

■ **Console**

Shows initialization, information, and other server related activities in a default Eclipse view.

The Bindings View

The Bindings view provides a logical representation of the data available to the portlet in the design canvas. In the Bindings view, you can bind data or methods to the user interface (UI) controls enabling the controls to display data or perform some action. The Bindings view facilitates defining data bindings for data elements. The Bindings view is active when a view in the design canvas or another editor contains a control that needs data. When you select a node in the Bindings view, properties for that node are displayed in the Properties view.

The Bindings view contains several types of managed beans and other assets used by the portlet.

The Bindings view filters data to the portlet page visible in the design canvas. You should leave this filter in place to avoid using data from a different portlet. You need to create binding expressions to data associated with the current portlet.

The Bindings view has automatic user interface generation. If you drag an object from the Bindings view to the design canvas, the canvas is populated with controls appropriate to the object. For some objects, such as Web service connectors, there are multiple sets of controls. You can choose among available input controls. Also in the case of complex objects such as Web connectors, you can choose which elements of the object to include. For a simple example of this, see [“Wiring Portlets with Preferences” on page 112](#).


Data Source Explorer View

The Data Source Explorer is an Eclipse view that allows you to connect to, navigate, and interact with resources associated with a selected connection profile, such as a database. After you have connected to a database, you can drag resources onto the design canvas for use in Web applications. By default, this view is displayed in the UI Development perspective.

Connecting to a Database

To connect to a database in the Data Source Explorer view, do the following:

➤ To connect to a database in the Data Source Explorer

1. If the Data Source Explorer view is not visible:
Window > Show View > DataSource Explorer.
2. In the **Data Source Explorer**, click  **New Connection Profile**.
3. In **New Connection Profile** page, select the **Connection Profile Type** and click **Next**.
4. In Specify Driver and Connection Details, from the **Drivers** list, select a driver, and then click **Next**.
5. In **Select a driver from the dropdown**, choose the appropriate database driver.

6. In the **URL** field, type the URL for the database, including the database name. For example:

```
jdbc:wm:sqlserver://localhost:1433;DatabaseName=Northwind
```

7. In the **User Name** field, type the valid user name for the database.
8. In the **Password** field, type the password.
9. Click **Test Connection** to test the connection to the database.
10. Click **Finish**.
11. Right-click the node, and click **Connect**.

Creating a Database Connector

To create a database connector in the Data Source Explorer view, do the following:

➤ To create a Database connector on the Data Source Explorer view

1. Expand the tree view of the Database Explorer view and locate the table or view for which you want to create a Database connector.
2. Do either of the following:

- To accept all of the default values and populate the design canvas with all columns in the table, drag the table from the **DataSource Explorer** view to the design canvas.

—OR—

- To choose values other than the defaults, do the following:
 1. Right-click the table and click **Generate Database Connector**.
 2. If the project in the **Project** field is correct, click **Next**.
 3. In **Managed Bean Name** field of the DB Client Properties panel, either accept the default name or type anew one.
 4. To allow the database connector to write to the database, clear the **Read-only client** option, and click **Next**.

Connectors are read-only by default.

5. In **Query Columns Selection**, choose the table columns to query.
6. (Optional) To add another table, click **Add Table**, choose a table, and click **OK**.

For tables that have foreign key relations with the main table, the join condition is added automatically.



7. Click **Finish**.

The managed bean appears on the Bindings view. For more information, see [“The Bindings View” on page 215](#).

8. Drag the managed bean for the database connector from the Bindings view to the design canvas.

Outline View

The Outline view is a standard Eclipse view that in the Composite Application Framework provides a tree structure of the portlet currently open in the design canvas. This view is useful in locating specific controls within a complex portlet and for moving controls from one location to another. The following table describes the available viewing modes, which you can select from the Outline view toolbar:

Button	Name	Purpose
	Outline	Displays the controls in a tree view that you can expand and collapse as needed. When you select a control in this view, the design canvas displays the control. The default mode
	Overview	Displays a smaller version of the design canvas. The shaded area corresponds to the visible portion of the design canvas. As you drag the shaded area in the Outline view, the display in the design canvas changes to match it.

➤ To use the Outline view



1. If the Outline view is not visible, in the Composite Application Framework, click **Window > Show View > Outline**.
2. If you need to display a portlet in the design canvas, expand the portlet application and double-click this node:

portlet_application/WebContent/portlet_name/default.view

The tree view of the portlet appears on the Outline view. The top-level node is the Form control that contains all other controls in the portlet.

3. Expand the nodes as needed to display the hierarchical view of the portlet.
4. Perform the following actions on the Outline view as needed.

To do this...	Do this on the Outline view...
Locate a control in the design canvas	Expand nodes until you find the control. When you click the control, it is highlighted in the design canvas.

To do this...	Do this on the Outline view...
Move a control in the design canvas	Drag the control to a new location. You can only drop a control in a valid location. All children of the control are also moved.
Copy a control to another portlet or portlet application	Right-click the control and click Copy . To remove the control from the first portlet, click Cut . Open the new portlet in the design canvas. On the Outline view, right-click the location where the control is to reside and click Paste . All children of the control are included.
Delete a control	Right-click the control and click Delete . All children of the control are also deleted.
Display Overview mode	On the toolbar, click  to change the Outline view to Overview mode.
Display Outline mode	On the toolbar, click  to change the Outline view to Outline mode.

Package Navigator View

The Package Navigator view in the Composite Application Framework UI Development Perspective facilitates connecting to multiple instances of Integration Server, browsing to services, and creating Web service connectors. The Package Navigator view requires installing the ESB and Integration Server feature. For more information, see *webMethods Service Development Help*.





Properties View

The Properties view is an Eclipse view that displays the names and values of properties associated with a resource in Composite Application Framework. When focus is on a the View Declaration Language (VDL) file for a view displayed in the design canvas, you can edit the properties for the resource in the Properties view.

- Shows the server name and URL of the webMethods Integration Server resources available from the Package Navigator view.
- Shows the root path and connection URL properties and other attributes for the My webMethods Server instance in the MWS Admin view.
- Shows the properties associated with the resources selected in the Bindings view.
- Shows the properties associated with the selected resources in the design canvas.
- Shows the properties associated with the selected resource in the Outline view.
- Shows a read-only view properties selected in the Navigator view and Project Explorer views.

Properties View Toolbar

The following table lists the buttons, available on the toolbar at the top of the **Properties** view:

Button	Name	Purpose
	Hide Expert Items	Hides or displays expert properties, which can appear in an additional tab, such as the Client-Side Events tab for the Text Input control, or as additional properties in an existing tab.
	Show Deprecated Items	Displays or hides deprecated properties, which appear as additional properties in a tab.
	Go Up	For any child control, selects the parent control in the design canvas and displays its properties in the Properties view. You can also click the Down arrow immediately to the right of this button and click any control in the menu. The top item of the menu is the parent of the current control, followed by each succeeding parent control in turn.
	Menu	A pop-up menu for the view.




Snippets View





The Snippets view consolidates default controls as snippets for reuse. You can create, edit, delete, and organize your own customized snippets of controls, containers, source code, and clipboard content.

Customizing the Snippets View

➤ To customize the Snippets view

1. If the Snippets view is not visible, in Composite Application Framework, click **Window > Show View > Snippets**.
2. Right-click the **Snippets** view, and then click **Customize**.
3. Customize items on the Snippets view as required by performing the steps, listed in the following table.

Action	Steps
Add a drawer	In the Customize Palette dialog box, click  , click  , and then, under Name , type a name for the drawer in the text box.
Delete a drawer	In the Customize Palette dialog box, select the drawer that you want to delete, and then click  .

Action	Steps
Move a drawer	In the Customize Palette dialog box, select the drawer that you want to move, and then click  or  . The drawer and all snippets in that drawer move to the new location.
Set a drawer to open at startup	In the Customize Palette dialog box, select the customized drawer that you want to open at startup, and then select the Open drawer at start-up check box.
Pin a drawer to open on startup	In the Customize Palette dialog box, select the customized drawer that you want to mark to open at startup, select the Open drawer at start-up check box, and then select the Pin drawer open at start-up check box.
Rename a drawer	In the Customize Palette dialog box, select the customized drawer that you want to rename, and then, under Name , type a new name for the drawer in the text box.
Hide a drawer	In the Customize Palette dialog box, select the drawer that you want to hide, and then select the Hide check box.
Display a hidden drawer	In the Customize Palette dialog box, select the drawer that you want to display, and then clear the Hide check box.
Specify the content types that display a drawer	In the Customize Palette dialog box, select the drawer that you want to display for specific content types, under Show/Hide Drawer , click Custom , click Browse , and then select the content types in which to display the drawer.
Move a snippet	<p>In the Customize Palette dialog box, expand the drawer that contains the snippet, click the snippet, and then click  or . The snippet moves to the new location in the drawer.</p> <p>You cannot move a snippet into a different drawer, but you can copy and paste a snippet into a different drawer.</p>
Rename a snippet	In the Customize Palette dialog box, expand the customizable drawer that contains the snippet, click the snippet, and then, under Name , type a new name for the snippet in the text box.
Hide a snippet	In the Customize Palette dialog box, expand the drawer that contains the snippet, click the snippet, and then select the Hide check box. To display a hidden snippet, clear the Hide check box.
Display a hidden snippet	In the Customize Palette dialog box, expand the drawer that contains the hidden snippet, click the snippet, and then clear the Hide checkbox.

E-forms Snippets Drawer

Use E-forms snippets when developing Task portlet solutions. The E-forms snippets include:

- Download and upload UI
- Download column for Task Inbox

For information on using E-forms snippets in Task development, see [webMethods BPM Task Development Help](#).

CAF Dialog Patterns

Use the CAF Dialog Patterns, Task Modal Dialog when developing Task portlet solutions. The Task Modal Dialog uses a Cascading Style Sheet with a designated title, description, and submit buttons enabling users to submit new content. For more information on developing Task solutions, see [webMethods BPM Task Development Help](#).

Solutions View

The Solutions view provides a quick way to see and perform actions with the assets in a Web or portlet application project. You can:

- Expand the nodes in the **Default Solution** to see the hierarchical views of the Web or portlet application projects in your workspace.
- Open the **Portlet Application Configuration** editor to update the portlet.xml file.
- Open the **Faces Configuration** editor to update faces-config.xml for a portlet application.
- Create a new database connector when the Web application has an established database connection.

Database Connectors in the Solutions View

After you have connected to a database using the DataSource Explorer view, you can create a new database connector or modify an existing one.

In the Solutions view, you can also delete database connections.

Before you can create a new database connector in the Solutions view, you must connect the Composite Application Framework application project to a database and an existing database connector.

To create a new database connector in the Solutions View, do the following.

➤ To create a Database Connector in the Solutions view

1. On the Solutions view, expand the Web application to expose the Database Connectors node, right-click Database Connectors, and click **New Database Connector**.

The Database Connection Configuration wizard appears, pre-configured with information about the database to which Composite Application Framework is connected.

2. If needed, modify the values for the **JDBC Driver**, **URL**, **Username**, and **Password** fields, and click **Next**.
3. In **Managed Bean Name** field of the DB Client Properties panel, either accept the default name or type a new one.
4. To allow the database connector to write to the database, clear the **Read-only client** option, and click **Next**.

Connectors are read-only by default.

5. In the **Table** list, choose a database table and click **OK**.
6. On the Query Columns Selection page, choose the table columns you want to query.
7. (Optional) To add another table, click **Add Table**, choose a table, and click **OK**.

For tables that have foreign key relations with the main table, the join condition is added automatically.

8. Click **Finish**.

Creating a Portlet on the Solutions View

Using the Solutions view, you can create a new portlet.

➤ To create a new portlet on the Solutions view

1. In the **UI Development** perspective, click the **Solutions** view, right-click a portlet application, and then click **New Portlet**.
2. In **New Portlet**, from the **Portlet Type** list, select a portlet type to use, and then click **Next**.
3. In **Create Portlet**, in **Portlet Name**, type a name for the portlet name.
4. (Optional) In **Portlet Modes**, select *Edit* or *Help* to create those portlet mode types, and then click **Finish**.



Using the Design Canvas

About the Design Canvas

The design canvas is a graphical editor used to design the pages of a Web or portlet view. The design canvas does not provide an exact representation of the layout as seen in a My webMethods Server page, on an application runtime server or in an external browser. You can run the application on My webMethods Server and use an external browser to review the user interface design and navigation issues that surface during development.

You create a page layout by dragging and dropping controls from the Palette view into the active view in the design canvas.

Some controls can contain other controls in a parent-child relationship. You can drag controls onto the canvas using the following guidelines:


- A vertical red bar appears at the place where the control is dropped.
- If the  overlay is visible on the cursor, you cannot drop the control.
- If the  overlay is visible on the cursor, you can drop the control.
- You cannot drop an input control outside of a Form control. For more information about input controls, see *webMethods CAF Tag Library Reference*.
- When you drag the control, descriptive hints are displayed to help you determine where to drop it.










The design canvas is fully integrated with the Java build mechanism in Composite Application Framework. As you make changes in the design canvas, the underlying Java code in the managed beans is updated.

Delete, **Cut**, **Copy**, **Paste**, and **Undo** commands from the **File** menu are valid on the design canvas. These commands are also available when you right-click individual controls on either the design canvas or the Outline view.

Design Canvas Toolbar

The design canvas toolbar provides functions to facilitate editing, navigating in the design canvas, and displaying both source and preview content. The following table lists the buttons, available on the toolbar:

Button	Name	Purpose
	Home	Sets the top node of the design canvas to the View Root control. Enabled only after the Go Into action has set another control as the top node.

Button	Name	Purpose
	Navigate back	Sets the top node of the editor to the parent node of the current node. Enabled only after the Go Into action has set a control as the top node.
	Go Into	Sets the top node of the editor as the currently selected control. This action enables you to drill into a specific section of the canvas for reduced clutter.
	Refresh	Redraws the visuals for the design canvas.
	Validate	Executes the validation for the view resource. Any validation errors discovered are available in the Problems view.
	Run	Starts or restarts the selected server listed in the Servers view.
	Debug	Restart the server in debug mode.
	Match Width	Sets the width of the selected controls to the width of the last control selected. Enabled only when more than one control is selected.
	Match Height	Sets the height of the selected controls to the height of the last control selected. Enabled only when more than one control is selected.
	Screen shot	Captures the view in the design canvas and saves it as a graphics image.
Source	Source	Opens the page bean Java class associated with the view. Visible only when the page Managed Bean property of the View Root control is a valid managed bean name.

Using the Design Canvas with Other Views

When designing a page in a Composite Application Framework Web or portlet application, you open the View Declaration Language (VDL) file for a view in the design canvas. If the current focus in Composite Application Framework is in the Solutions view, you can open a project VDL file while in the Solutions view. For more information, see [“Solutions View” on page 221](#).

You add controls to the VDL file by selecting and dragging controls from the Palette view to the VDL file. For more information, see [“The Palette View” on page 227](#).

You can modify properties associated with the VDL file by making changes in the Properties view. For more information, see [“Properties View” on page 218](#).

As the set of controls in the view opened in the design canvas becomes more complex, you can use the Outline view to navigate between controls, drag controls from one location to another, and perform delete, cut, copy, paste, and undo operations. For more information, see [“Outline View” on page 217](#).

You can review an HTML version of the view using the Preview tab of the design canvas.

Manipulating Controls on the Design Canvas

There are several ways to manipulate controls on the design canvas and modify their appearance and position.



Resizing Controls

To resize a control or a column, click the control to display the resizing handles at the corners and the center of each edge. Drag a handle to resize as needed. If you have multiple controls selected, you can resize the controls at the same time.


Moving Controls

To move a control from one place to another, click the control to display the drag handle in the upper left corner.

Place the cursor over the drag handle and drag the control to a new location using these guidelines:

- A vertical red bar appears at the place where the control is dropped.
- If the  overlay is visible on the cursor, you cannot drop the control.
- If the  overlay is visible on the cursor, you can drop the control.
- You cannot drop an input control outside of a Form control. For more information about input controls, see *webMethods CAF Tag Library Reference*.
- As you drag the control, descriptive hints are displayed to help you determine where to drop it.

Minimizing Controls

As the hierarchy of controls becomes more complex, you can minimize a control to hide all of its child controls. Click the control to display  in the upper left corner, and then click the icon.

To restore the control, click the control to display the restore in the upper right corner of the collapsed control and then click the icon. You can also restore the collapsed control by double-clicking the control.

Editing CSS Values

You can edit CSS (Cascading Style Sheet) values for controls without having to know CSS syntax. On the design canvas, right-click a control and then click **Style**. The following table lists the available CSS options:

Menu item	Description
Edit Style	Displays the CSS Style Definition editor. Use this editor to set values for a variety of CSS settings.

Menu item	Description
Reset Style	Click to reset the control to default CSS settings.
Style Classes	Choose a preset CSS style class. This feature requires you to create an external CSS and use an Include Stylesheet control to associate it with the view. For more information about the Include Stylesheet control, see <i>webMethods CAF Tag Library Reference</i> .
Color	From the menu, choose a color to use for foreground objects. The choices are a subset of colors available in the CSS Style Definition editor.
Background Color	From the menu, choose a color to use for the background. The choices are a subset of colors available in the CSS Style Definition editor.

Selecting a Parent or Child Control

With a particular control, you can quickly select its immediate parent control or any of its immediate child controls. Right-click the control, click **Select**, and then click the parent or child control. That control is now selected in the design canvas.

Showing a View

Occasionally a view is hidden because another view has become active. You can display certain views directly from the design canvas. Right-click a control, click **Show**, and click the name of the view to display. You can use this technique to display the Properties, Palette, or Outline views.

Design Canvas Preview

In Composite Application Framework, you can examine an HTML representation of the View Declaration Language (VDL) file loaded in the design canvas. The design canvas Preview tab shows a static representation of the view as it might appear in a deployed application. The design canvas Preview does not include data from user input or from a database. In addition, you should test the view using the external browsers that you expect users to use. You can add external browsers to use for while designing VDL files.



Matching Controls

You can set a group of controls to have the same width or height:

➤ To set a group of controls to have the same width or height

1. Resize one control to the desired width or height.
2. SHIFT+click each of the controls to have the matching width or height, clicking the correctly sized control last.

The last control selected becomes the primary selection.

3. Do one of the following:
 - To match control widths, on the design canvas toolbar click .
 - To match control heights, on the design canvas toolbar click .
4. Click the Preview tab at the bottom of the design canvas for a static view of the control layout.

Changing Control Type

If you need a different control from the one you placed on the canvas, rather than delete the control and drag another one from the Palette view, you can change the control type directly from the design canvas.

If you click a control type that is not compatible with the current control, Composite Application Framework disables the **OK** button.

➤ To change the control type on the design canvas

1. In the **UI Development Perspective**, in the design canvas, select a control in a view.
2. Right-click the control to changed and click **Change Control Type**
3. In the **Control Type Editor**, select the new control type from the tree view, and then click **OK**.

Changing Control Labels in the Design Canvas

You can change the label for a control directly in the design canvas.

➤ To change a control label directly on the design canvas

1. Click the control to select it, and then pause for a second or two.
2. Click the control a second time to open an editing field.
3. Type the new label and press CTRL+ENTER or click away from the control.

The Palette View

About the Palette View






The Palette view contains the objects such as user interface controls, and optionally, providers, data objects, validators, and converters. You use the objects in the nodes of the Palette view to

define the Web or portlet application's user interface. For more information about the available controls, see *webMethods CAF Tag Library Reference*.

Converters and validators are initially hidden from display in the Palette view. You can assign them to controls from the Properties view. For more information of making converters and validators visible in the Palette view, see [“Customizing the Palette View” on page 229](#).

Palette View Toolbar

The toolbar at the top of the Palette view provides mechanisms for navigating in the view. The following table lists the buttons on the toolbar and their descriptions:

Button	Name	Purpose
	Collapse All	Collapse all nodes and show only the top set of nodes. A quick way to clean up the view.
	Home	Sets the top node of the Palette view to the View Root control. Enabled only after the Go Into action has set another node as the top node.
	Back	Sets the top node of the Palette view to the parent node of the current node. Enabled only after the Go Into action has set a node as the top node.
	Go Into	Sets the top node of the editor to the currently selected control. This action enables you to drill into a specific section of the palette for reduced clutter.
	Menu	A pop-up menu for the view.

Adding an Object to the Favorites Node

The Palette view contains a Favorites node in which you can place shortcuts to palette objects you use on a regular basis. You can drag these shortcuts to the design canvas exactly as you would the original object.

➤ To add a palette object to the Favorites node

1. In the **UI Development Perspective**, click the Palette view, expand the node containing the control to add to the Favorites node.
2. Right-click the object and then click **Add To Favorites**.

Removing an Object from the Favorites Node

To remove a palette object from the Favorites node


1. In the **UI Development Perspective**, click the Palette view.
2. In the Palette view, expand the **Favorites** node, select a shortcut, and then click **Delete From Favorites**.

Customizing the Palette View

You can customize the Palette view to:

- Limit the controls of a specific node to contain only the controls and objects to you use most often
- Show converters or validators that are hidden initially
- Add controls that you use most often to the Favorites folder
- Expand the nodes of the Palette view at startup
- Hide controls
- Rename and add descriptions for objects in the Palette view

➤ To customize the Palette view

1. In the **UI Development Perspective**, click the Palette view, click  and then click **Customize**.
2. In the Customize Palette window, expand a node or object and do any of the following:
 - In the **Name** field, type a Display name used for the node or object. This field is not valid for shortcuts in the Favorites node.
 - In the **Description** field, type a description used for Tool Tips in Composite Application Framework. This field is not valid for shortcuts in the Favorites node.
 - Select the **Hide** check box to hide the node or object in the Palette tree view. If you hide a node, all objects nested in the node are also hidden.
 - Select the **Expanded Initially at startup** check box if you want the node expanded when the Palette view is opened. Select this check box to make the parent nodes expandable.
3. Click **Apply** to save a change and continue making additional changes to the Palette view or click **OK** to save and close the **Custom Palette**.

Filter the Palette View

The Palette view contains a large number of objects. If you the name of an object, you can use the filter field at the top of the Palette view to find the object quickly.

➤ To filter controls in the Palette view

In the filter field at the top of the Palette view, type a word that is part of the name of a palette object, using the following rules:

- The word does not have to represent the first word in the name. Typing `group` finds any object with `group` in the title, such as `Radio Group` and `Page Group`.
- Typing a space at the end of a word causes the filter to assume that word is the first word in the name. Typing `group` causes the filter to hide everything because there is no object whose name begins with `group`.
- Always start typing at the beginning of a word. Typing `rou` does not find the word `group`.
- Use wildcards; try typing `*oup` or `?roup` to find the word `group`.
- You can type partial words. Increase the letters to filter more objects from the results. If you type `pa`, names containing `Parameter`, `Panel`, and `Page` are displayed. If you type `pan`, only names containing `Panel` are displayed.

Display Deprecated UI Controls

Deprecated Composite Application Framework user interface controls are hidden by default. You can display deprecated controls by updating the Palette view.

➤ To display deprecated items in the Palette view

1. In the **UI Development Perspective**, click the Palette view, click .

If **Hide Deprecated Items** has a check mark beside it, all deprecated items are hidden.

2. Click **Hide Deprecated Items** to remove the check from the check box.

You can hide deprecated items by clicking the empty check box.

CAF Events Editor

Events Editor UI Reference

The Composite Application Framework Events Editor provides the following tabs:

- [“Introduction tab” on page 231](#)
- [“Overview tab” on page 231](#)
- [“Events tab” on page 231](#)
- [“Handlers tab” on page 232](#)
- [“Subscriptions tab” on page 235](#)
- [“Notifications tab” on page 235](#)

Introduction tab

The **Introduction** tab introduces the Composite Application Framework Events Editor. The following table lists the items, available on the **Introduction** tab:

Component	Description
Start	Provides a link to the Overview page
Help	Opens online help for the Events Editor.

Overview tab

The **Overview** tab consolidates and summarizes the necessary components used to create and configure events. The following table lists the items, available on the **Overview** tab:

Component	Description
Custom Event Types	Provides a summary of the custom event types for the project.
Event Handlers	Provides a summary of the event handlers for the project.
Subscriptions	Provides a summary of the subscriptions created for the project.
Notifications	Provides a summary of the notifications created in the project.

Events tab

The **Events** tab lists the custom events that have been created for and added to the project. The following table lists the items, available on the **Events** tab:

Component	Description
List of event types	Lists the custom event types created for the project.
Add	Adds a custom event type to the list of event types.
Delete	Deletes a selected custom event type from the list of event types.
Up	Decrements the position in the list of a selected custom event type listing.
Down	Increments the position of a selected custom event type listing.

Handlers tab

The **Handlers** tab allows you to create an event handler that specifies the event type, conditions, and actions for an event. The following table lists the items, available on the **Handlers** tab:

Component	Description
Event Handlers	
List of event handlers	Lists the event handlers created for the project.
Add	<p>Adds an event handler to the list of event handlers.</p> <p>Click Add to display the following components for adding event handler parameters: General Information, CAF Event Type, Conditions, Handler Actions</p>
Delete	Deletes a selected event handler from the list of event handlers.
Up	Decrements the run-time position of a selected event handler listing. Event handlers for the same event type are processed in the order that they are defined.
Down	Increments the run-time position of a selected event handler listing. Event handlers for the same event type are processed in the order that they are defined.
General	
Handler Name	Required. Use this text box to name the event handler.
Description	Use this text box to describe the event handler.
CAF Event Type	
Event Type	Required. Enables selection of a My webMethods Server event or custom event type.
Create	A My webMethods Server event type that is fired when an object is created, for example, a page, portlet, folder, file, and so on.
Delete	A My webMethods Server event type that is fired when an object is deleted, for example, a page, portlet, folder, file, and so on.
Login	A My webMethods Server event type that initiates an event when a user successfully logs in or logs out, or when a login script starts and completes.

Component	Description
Login Failed	A My webMethods Server event type that initiates an event when a user's login or logout fails, or when a login script fails to complete.
Update	A My webMethods Server event type that fires when an object is updated, for example, a page, portlet, folder, file, and so on.
View	A My webMethods Server event type that fires when a user accesses an object, for example, a page, portlet, folder, file, and so on.
Workspace Share	A My webMethods Server event type that initiates an event when a workspace is shared.
Workspace Un-Share	A My webMethods Server event type that initiates an event when a workspace is u not shared.
Listener Type	
Queued	Default. Executes the event handler on only one node in a cluster after the event initiates.
Asynchronous	Executes the event handler on every node in a cluster after the event initiates.
Synchronous	Executes the event handler on every node in a cluster immediately when the event initiates.
	Note: Increasing the number of synchronous event handlers can significantly increase the processing time of the action that initiated the event.
Conditions	
Condition Type	
Joined List of Simple Conditions	Provides a list of conditions that you can apply to an event.
Advanced condition expression	Enables you to enter a formula for an advanced condition.
Advanced Condition	
Condition	Use this text box to enter a formula for an advanced condition.
Simple Condition	

Component	Description
ALL are true	Sets the conditions to require all conditions to be true.
ANY is true	Sets the conditions to require any one condition to be true.
One only is true	Sets the conditions to require only one condition to be true.
Add	Adds a field, operation, and optional value for a condition.
Edit	Enables editing of a field, operation, and optional value for a condition.
Delete	Deletes a condition.
Up	Decrements the position in the list of a selected condition listing.
Down	Increments the position in the list of a selected condition listing.
Handler Actions	
Action Type	
List of Simple Actions	Provides a list of handler actions that you can apply to an event.
Advanced action expression	Enables you to enter a formula for an advanced action.
Advanced Action	Use this text box to enter a formula for an advanced action.
Simple Actions	
Add	<p>Adds a Fire Ajax Event, Invoke Service, or Send Notification action to the event.</p> <ul style="list-style-type: none"> ■ Fire Ajax Event. Provides Ajax functionality for the event. See “Configuring a Fire Ajax Event Action” on page 127. ■ Invoke Service. Invokes a service or an action for the event. ■ Send Notification. Initiates a notification for the event. See “Creating a Notification” on page 131. <p>Note: You must create a notification before you can use the Send Notification action.</p>
Edit	Enables editing of a Fire Ajax Event, Invoke Service, or Send Notification action for an event.

Component	Description
	<ul style="list-style-type: none"> ■ Fire Ajax Event. Provides Ajax functionality for the event. See “Configuring a Fire Ajax Event Action” on page 127. ■ Invoke Service. Invokes a service or an action for the event. See “Configuring an Invoke Service Action” on page 129. ■ Send Notification. Initiates a notification for the event. See “Creating a Notification” on page 131.
Delete	Deletes an action.
Up	Decrements the position in the list of a selected action listing.
Down	Increments the position in the list of a selected action listing.

Subscriptions tab

The Subscriptions tab specifies the conditions for each subscription that is to be used with a notification. The following table lists the elements, available on the Subscriptions tab:

Element	Description
List of event subscriptions	Lists the event subscriptions created for the project.
Add	Adds a subscription to the list of subscriptions.
Delete	Deletes a selected subscription from the list of subscriptions.
Up	Decrements the position in the list of a selected subscription listing
Down	Increments the position in the list of a selected subscription listing.

Notifications tab

The **Notifications** tab specifies the My webMethods Server or custom event type that initiates a notification of a change. The following table lists the elements, available on the **Notifications** tab:

Elements	Description
List of event notifications	Lists the event notifications created for the project.
Add	Adds an event notification to the list of event notifications.
Delete	Deletes a selected event notification from the list of event notifications.

Elements	Description
Up	Decrements the position in the list of a selected event notification listing.
Down	Increments the position in the list of a selected event notification listing

Using Snippets

Using a Snippet for Controls or Containers

You can use a snippet to quickly reuse controls, complex controls, containers, or clipboard content.

➤ **To use a snippet for controls, complex controls, or containers**

1. On the Snippets view, expand the drawer that contains the snippet.
2. Drag the snippet onto the design canvas.

Using a Snippet for Source Code

You can use a snippet to quickly reuse source code.

➤ **To use a snippet for source code**

1. On the toolbar of the design canvas, click **Source**.
2. Scroll to the location in the source code to insert the snippet code.
3. On the Snippets view, expand the drawer that contains the snippet.
4. Drag the snippet into the source code, position the cursor at the insertion point, and then drop the snippet.

Creating a Snippet of a Control or Container

You can quickly create a snippet to reuse controls, complex controls, or containers.

➤ **To create a snippet of controls, complex controls, and containers from the design canvas**

1. On the design canvas, right-click the controls, complex controls, or containers that you want to include, and then click **Create Snippet**.

2. In the **Create New Snippet** dialog box, in the **Name** text box, type a name.
3. In the **Description** text box, type a description.
4. In the **Category** pull-down list, select a category for the snippet, and then click **OK**.

Creating a Snippet of Source Code

You can quickly create a snippet to reuse source code.

➤ To create a snippet of source code

1. On the toolbar of the design canvas, click **Source**.
2. Scroll to the location in the source code that contains the code for the snippet, and then select the code.
3. Right-click the selected code, and then click **Add to Snippets**.
4. In the **New Category** dialog box, select a drawer or type the name for a new drawer, and then click **OK**.
5. In the **Customize Palette** dialog box, under **Name**, type a name.
6. In the **Description** text box, type a description, and then click **OK**.

Creating a Snippet of Clipboard Content

You can quickly create a snippet to reuse clipboard content.

➤ To create a snippet of clipboard content

1. Copy content for the snippet into the clipboard.
2. Right-click a customized drawer on the Snippets view, and then click **Paste as Snippet**.

–or–

Right-click a default drawer, click **Paste as Snippet**, in the **New Category** dialog box, select a drawer or type the name for a new drawer, and then click **OK**.

3. In the **Customize Palette** dialog box, under **Name**, type a name.
4. In the **Description** text box, type a description, and then click **OK**.

Editing a Snippet

You can customize a snippet that you created by editing the snippet's parameters. You can only edit a snippet that you created. You cannot edit a default snippet.

> To edit a snippet

1. On the Snippets view, expand the drawer that contains the snippet.
2. Right-click the snippet, and then click **Customize**.
3. In the **Customize Palette** dialog box, make the changes that you want, and then click **OK**.

Copying and Pasting a Snippet

Copy and paste a snippet to quickly create a customizable snippet.

> To copy and paste a snippet

1. On the Snippets view, expand the drawer that contains the snippet.
2. Right-click the snippet, and then click **Copy**.
3. Expand the drawer where you want to paste the snippet.
4. Right-click the drawer, and then click **Paste**.

Cutting and Pasting a Snippet

Cut and paste a created snippet to quickly create customized controls, complex controls, containers, or source code. You can only cut a snippet that you created. You cannot cut a default snippet.


> To cut and paste a snippet

1. On the Snippets view, expand the drawer that contains the created snippet.
2. Right-click the snippet, and then click **Cut**.
3. Expand the drawer where you want to paste the snippet.
4. Right-click the drawer, and then click **Paste**.

Deleting a Snippet

Delete any customized snippet that you no longer need or use.


> To delete a snippet

1. On the Snippets view, expand the drawer that contains the created snippet.
2. Right-click the snippet, and then click **Customize**.
3. Click the snippet that you want to delete, and then click .

Exporting Snippets

You can export the snippets that are contained in a drawer or snippet code, which can then be imported. You can only export all of the snippets contained in a customized drawer. You cannot export default snippets or a single snippet contained in a drawer.


> To export the snippets contained in a drawer or snippet code

1. On the Snippets view, click the drawer that contains the snippets that you want to export.
2. Right-click the drawer, and then click **Customize**.
3. In the **Customize Palette** dialog box, click .
4. In the **File name** box, enter a new name for the file, and then click **Open**.

Importing Snippets

You can import exported snippets contained in a drawer or snippet code.

> To import snippets

1. Right-click the Snippets view.
2. In the **Customize Palette** dialog box, click .
3. Locate the exported snippets XML file, and then click **Open**.

Creating a Variable Placeholder

> To create a variable placeholder

1. On the Snippets view, expand the drawer that contains the created snippet.
2. Right-click the snippet, and then click **Customize**.
3. In the **Customize Palette** dialog box, click **New**.
4. In the **Variables** box, under **Name**, click the inserted variable placeholder, and then type a new name.
5. Under **Description**, click the text box, and then type a description.
6. Under **Default Value**, click the text box, and then type a value for the variable.

Inserting a Variable Placeholder into a Snippet

You can insert a variable placeholder into a snippet to specify the location for inserting a variable. You can use a variable to customize or parameterize the snippet. For Composite Application Framework user interface snippets, inserting a variable placeholder allows you to specify custom values for control properties such as *MyButtonLabel* and *TextInputWidth*.

> To insert a variable placeholder

1. On the Snippets view, expand the drawer that contains the created snippet.
2. Right-click the snippet, and then click **Customize**.
3. In the **Customize Palette** dialog box, in the **Template Pattern** text box, click to place the cursor at the insertion point.
4. Click **Insert Variable Placeholder**, and then double-click the variable placeholder name.

Deleting a Variable Placeholder

> To delete a variable placeholder

1. On the Snippets view, expand the drawer that contains the created snippet.
2. Right-click the snippet, and then click **Customize**.

3. click **New**.
4. In the **Customize Palette** dialog box, in the **Variables** box, click the variable placeholder, and then click **Remove**.

Using MWS Control Patterns Snippets

On the Snippets view, the MWS Control Patterns drawer contains the Select People snippet. Use the Select People snippet to simplify implementation of selecting users, groups, or roles.

> To use the Select People snippet

1. On the Snippets view, expand the **MWS Control Patterns** drawer.
2. Drag the **Select People** snippet onto the design canvas.
3. In the **Insert Template** dialog box, click **Insert**.
4. Resolve the invalid expression for: `#{activePageBean.person.displayName}`.
 - a. In the Palette view, expand **Data > User/Role/Group**, and then drag a User Model control onto the page bean.
 - b. In the **Choose Data Target** dialog box, click **Add new simple property to portlet_name/page_name**, and then click **Next**.
 - c. In the **Property Name** text box, type a name (person is recommended), select the **Property is writable (setter is generated)** text box, and then click **Finish**.

Customizing UI Controls

You can customize these Composite Application Framework user interface controls:

- Attachments List Panel
- Export Table Button
- People Picker Dialog
- People SearchBar
- People SwapBox
- Person Calendar Dialog
- Portal Resource Picker Dialog

You can use the customize functionality with a listed user interface control to see how the control was constructed and then modify the control to fit your needs.

For more information about these controls, see *webMethods CAF Tag Library Reference*.

➤ To customize a user interface control

1. In the UI Development perspective of an opened Web or portlet application project, click the Project Explorer view, expand the **Web Content** node and drag the view to modify to the design canvas.

You can also double-click the view to open it in the design canvas.

2. Click the Palette view and drag a customizable user interface control to the design canvas.
3. Right-click the user interface control, and then click **Customize**.

You can modify the user interface control by editing or deleting the individual user interface control elements that comprise the original user interface control.

24 User Interface Controls Concepts

■ Control ID Reference	244
■ Hideable Controls	246
■ Toggle Controls	248
■ Scriptaculous Effects	249
■ Image URLs	253
■ Skinning	253
■ Client-Side Libraries	254

Control ID Reference

You can assign an ID to every Composite Application Framework control. You can use this ID to reference other controls in the same view, such as the `for` attribute of a One Way Toggle Button control, and the select item values of a Toggle Button control. Both of these buttons use the control ID reference to specify the control or controls to toggle.

For more information about these controls, see *webMethods CAF Tag Library Reference*.

You must define a unique control ID for the element within the control's naming container. The following controls are naming containers:

- Form
- Iterator
- Simple List
- Table
- Tabs
- Tree

Absolute References

Control IDs are not unique outside of naming containers, a single view can contain several different controls with the same ID. To distinguish between these controls, the control ID references can include the control's naming container ancestor IDs in the reference. For example, given a form control with an ID of *myform* and a Hideable Panel control in the form with an ID of *myPanel*, the **absolute** control ID reference to the Hideable Panel is `:myForm:myPanel`. An absolute control ID reference starts with a leading colon (:) and identifies a control from any other control on the page.

Relative References

You can make relative control ID references. A **relative** reference is corresponding to a specified context, usually the context is a portlet application or naming container. The control ID might also act as a relative reference. For example, to reference the show/hide panel control from a specific control such as a button below the form, you would specify the ID of the panel, *myPanel*. When a relative reference consists of only a control ID, the reference is resolved by finding the adjacent control to the referencing control with the specified ID.

If another control with the same control ID is closer to the referencing control, you must qualify the reference with part of the absolute reference. For example, there is a panel sibling with an ID of *myPanel* close to the button, you need to create a unique reference to the first *myPanel* show/hide panel. To create a relative reference the *myPanel* control that is in the form, use a relative reference such as `myForm:myPanel`. Relative control ID references do not start with a leading colon (:), and depend on the location of the referencing control to correctly identify the referenced control.

Absolute Versus Relative

In the relative reference example, the relative control ID reference `myForm:myPanel` is similar to the absolute control ID reference: `myForm:myPanel`. Using the relative reference version removes the need to rename the control when changes are made to the container. If you add a new naming container around both the form and the button by importing the view into another view, you can use the same relative reference for the control ID to reference the panel from the button. If you used an absolute reference for the control ID, you would need to prefix to it the naming container's absolute reference control ID. For example, if the Import View's control ID was `myImportView`, with an absolute reference control ID of `:myImportView`, the panel's absolute reference control ID must change to `:myImportView:myForm:myPanel`.

External Portlet References

Some controls can accept a reference to another control outside of the current view, in an external portlet on the same portlet page. An external portlet reference begins with the URI or an alias to the portlet, followed by a pound sign (`#`), and then by the absolute reference control ID without the leading colon. For example, the `myPanel` control from the first absolute reference is part of a portlet with an alias of `myPortlet`. External portlets can reference the `myPanel` control on the same page as `myPortlet#myForm:myPanel`.

Literal HTML References

Some controls can accept a reference to a raw HTML element instead of a conventional control ID reference. To reference a raw HTML element, specify the element's ID prefixed with a dollar sign (`$`). For example, `$myDiv` references a `div` tag with an ID of `myDiv`. You should not use a literal HTML reference when you can use a regular control ID reference.

Nearest Control Algorithm

The nearest control in a relative reference control ID is the closest control in the \ sub-tree to the referencing or source control. The algorithm for finding the nearest control is the following:

- Test the source control.
- Search the descendants of the source control, depth-first.
- Search the siblings of the source control.
- Search the descendants of the siblings of the source control, depth-first.
- Test the parent of the source control.
- Search the siblings of the parent of the source control.
- Search the descendants of the siblings of the parent of the source control, depth-first.
- Test the grandparent of the source control.
- Search the siblings of the grandparent of the source control.

- Search the descendants of the siblings of the grandparent of the source control, depth-first.

Hideable Controls

Visibility (hideable) controls switch a Composite Application Framework control from visible and hidden through the use of JavaScript code. You manage the visibility of the Composite Application Framework controls with the visibility feature using toggle controls. For more information about using toggle controls, see [“Toggle Controls” on page 248](#).

Server-Side Properties

Visibility controls have a server-side `visible` Boolean property that indicates whether the control is rendered as initially visible when the page containing the control is displayed in response to a user request. You can use the visibility properties to return the page results based on the user request and display only the controls needed to fulfill the request.

Visibility controls also have a `disableWhenHidden` Boolean property that makes controls contained within the parent control invisible. When a Composite Application Framework user interface control is `disableWhenHidden`, the control's values are not submitted to the server.

In addition, the `defaultFocus` property of visibility controls sets the focus on a specified [Control ID Reference](#) referenced in the property's value.

You can also manage a user interface control's visibility with properties called Scriptaculous Effects. Composite Application Framework controls do not use Scriptaculous effects as their default behavior. For more information, see [“Scriptaculous Effects” on page 249](#).

Client-Side Functionality

You can manage the control's visibility on the client side using the Composite Application Framework client-side model. When you change the client-side visibility of a control, you also change the control's server-side visibility property.

You can use the following methods to affect the control's client-side visibility:

- `isVisible()` returns true if the control is visible.
- `setVisible(visible)` sets the control to display as visible or hidden.
- `show()` makes the control visible if hidden.
- `hide()` makes the control hidden if visible.
- `toggle()` makes the control visible if hidden, or hidden if visible.

To use a client-side method to set the value of a `myHideableControlId` check box's `onlick` property, do the following:

```
CAF.model("#{myPageBean.clientIds'myHideableControlId'}').toggle();
```

The result switches a separate *myHideableControlId* hideable control between visible and hidden when the user clicks the check box. For more information, see [“About the Client-Side Model” on page 264](#).

Lazy Loading

Many hideable controls have lazy loading capabilities that enable using a hidden request that loads the control's content asynchronously. The panel's lazy load properties controls this capability. Setting lazy to true enables it, while setting lazy to false, disables lazy loading. The default setting for the lazy loading capability is false. For information on using lazy load, see [“About Ajax” on page 17](#).

refreshOnShow Property

Composite Application Framework provides properties to modify a hideable control's lazy-loading capabilities. These properties have no effect unless the control's lazy property is true. The `refreshOnShow` property refreshes the content contained within the control when the control changes from hidden to visible. When you set `refreshOnShow` to true, it makes the control retrieve its content again, using a secondary request each time visibility changes to visible. When you set `refreshOnShow` to false, the control's content is not updated (refreshed) if the control's visibility changes from hidden to visible.

twoPass Property

The `twoPass` property forces a hideable control to retrieve its content using a secondary request. Setting the `twoPass` property to true prevents the control's content from rendering with the initial page. The control's content is retrieved by a secondary request when the page is loaded by the browser. Setting `twoPass` to false makes the control's content render synchronously with the page when the control is initially visible. If the control is not initially visible, the `twoPass` property has no effect.

supressInputs Property

When a secondary request is made to the server to retrieve the content of a lazy loaded control, and the control is part of a form, the state of all the form's controls are: posted to the server; the controls are validated; and the control values are updated the data models to which they are bound. This behavior is not desirable in all cases. You can use the `supressInputs` property to specify a comma-separated list of IDs of the controls you do not want validated or updated. The descendants of any control in the list of suppressed inputs is also suppressed, for example, if you want to suppress the validation and update processing of all the controls in a property-group control, you can simply specify the ID of the property-group instead of the ID of each individual control in the group. However, any direct ancestor of the hideable control in the list of suppressed inputs is ignored so you cannot add the ID of the form to the suppressed list.

Progress Bar Customization

A progress bar automatically appears when a hideable control's content is loaded asynchronously. You can configure the progress bar's display with the following hideable control properties:

- **progressDelay**: Milliseconds to delay before showing progress bar, defaults to 0. A -, negative one, suppresses the progress bar completely.
- **progressFlashOnComplete**: True to flash control when control's content finishes loading; defaults to true.
- **progressMsg**: Progress bar message; defaults to localized value; in English, for example, Loading...
- **progressUseHideShowEffect**: Use effects specified by `hideEffect`, `hideEffectOptions`, `showEffect`, and `showEffectOptions` properties to hide and show progress bar; defaults to false.

Hideable Control Instances

The server-side component object model of controls that are hideable implement the `com.webmethods.caf.faces.component.IHideablePanel` interface. These controls include the following:

- Hideable Panel
- Inline Hideable Panel
- Modal Dialog
- Modeless Dialog
- Overlay Panel
- Progress Dialog

Toggle Controls

The Composite Application Framework Toggle controls enable managing the client-side visibility of controls that you can hide and the server-side rendering property of controls that you cannot hide.

Only the selected control from the group listed in the `value` property of the toggle control is visible. Selection options are specified using `javax.faces.SelectItem` and `javax.faces.SelectItems` with each value specifying a control ID. Selecting an option whose value does not specify a control ID hides all controls in the group. For more information, see the Option Group and Option controls in *webMethods CAF Tag Library Reference*.

When using a toggle control, if the toggle control has three options, *controlId-one*, *controlId-two*, and *controlId-three* and the toggle control's `value` property is empty, then all of the controls are hidden. If the toggle's control `value` property is set to *controlId-one*, then *controlId-one* is visible and the remaining controls are hidden.

For hideable controls, the toggle control modifies the control's client-side visibility without requiring a page refresh or contact with the server. For more information, see [“Hideable Controls” on page 246](#).

Toggle control options can reference external hideable controls using external portlet references, or raw HTML elements using literal HTML references. For more information on how to construct these references, see [“Control ID Reference” on page 244](#).

You can use a toggle control to change a single control from visible to hidden. For example, if the toggle control has two options, *controlId-one* and no-specified value as the second option leaving the Toggle Button control's value property is empty, then *controlId-one* is hidden. If the toggle's control value property is set to *controlId-one* then *controlId-one* is visible.

For controls that are always visible and cannot toggle between visible and hidden. However, you can use a toggle control to change the control's server-side rendered property, and refresh the page to display the control's updated state from visible to hidden. To use the toggle control to show or hide controls that are usually visible contain the toggle control in Form control. The toggle control updates the current state of any controls in the Form to refresh the page.

When a toggle control is positioned after, top-to-bottom, or left-to-right of any of the controls it toggles, you must add an Initiate Toggle control to the view before the first control, at the same level as the toggle control. For example, if the toggle control is in a table column, the Initiate Toggle control is the first control in the first column of the table. Set the Initiate Toggle control's for property to reference the toggle control.

You can also use an Initiate Toggle control, when a toggle control comes after one of the controls it toggles.

Behavior With Non-Hideable Controls

For controls that are always visible, the toggle control can change the control's server-side rendered property, refreshing the page to display the updated state to visible or hidden. To use the toggle control to show or hide controls that are usually visible, contain the toggle control within a Form control. The current state of all the controls within the form are transferred to the refreshed page.

Toggle Control Instances

The server-side component object model for toggle controls extend the `com.webmethods.caf.faces.component.toggle.Toggle` class. These controls include the following:

- Toggle Button
- Toggle Dropdown
- Toggle Link
- Toggle Radio Button Group
- Toggle Tabs

Scriptaculous Effects

Composite Application Framework uses the [Scriptaculous](#) library for client-side effects. For example, the Tooltip control's fade and appear effect or the Modal Dialog control's grow and shrink effect are from the Scriptaculous library. Some controls, specifically the controls with

show/hide options enable configuring the control's hide and show effects using the `hideEffect` and `showEffect` expert properties. You can specify any standard Scriptaculous effect name, for example, `Effect.Fade`, or any custom Composite Application Framework effect name, for example, `Effect.CAF.SlideOpen`, for these values.

You can specify additional options for these effects using the `hideEffectOptions` and `showEffectOptions` expert properties. Options are specified in JavaScript Object Notation (JSON), for example:

```
direction: "bottom", duration: 0.5, transition: Effect.Transitions.wobble
```

Many effects work with their defaults, and do not require you to specify any options.

Standard Scriptaculous Effects

Show

- **Effect.Appear:** Fade in.
- **Effect.BlindDown:** Reveals from top to bottom.
- **Effect.SlideDown:** Slide open, downward.
- **Effect.Grow:** Grow from center.

Hide

- **Effect.Fade:** Fade out.
- **Effect.BlindUp:** Cover from bottom to top.
- **Effect.SlideUp:** Slide close, upward.
- **Effect.Shrink:** Shrink to center.
- **Effect.Puff:** Expand and fade out.
- **Effect.SwitchOff:** Flicker and drop out.
- **Effect.DropOut:** Move down and fade out.
- **Effect.Squish:** Shrink to corner.
- **Effect.Fold:** Blind up, then shrink to corner.

Highlight

- **Effect.Shake:** Shake back and forth.
- **Effect.Pulsate:** Fade in, fade out, fade in, fade out.
- **Effect.Highlight:** Fade in yellow background, fade back to white background.

Custom CAF Effects

Show

- **Effect.CAF.Appear:** Extends `Effect.Appear` with a default duration of 0.3.
- **Effect.CAF.SlideOpen:** Slide open in any direction; options:
 - *direction:* Direction to slide: left, top, right, or bottom, the default value.
 - *duration:* Duration, in seconds, of slide; defaults to 0.3.
- **Effect.CAF.SlideOpen.Left:** Slide open leftward.
- **Effect.CAF.SlideOpen.Top:** Slide open upward.
- **Effect.CAF.SlideOpen.Right:** Slide open rightward.
- **Effect.CAF.SlideOpen.Bottom:** Slide open downward.

Hide

- **Effect.CAF.Fade:** Extends `Effect.Fade` with a default duration of 0.3.
- **Effect.CAF.SlideClose:** Slide close in any direction; options:
 - *direction:* Direction from which to slide: left, top, right, or bottom, the default value.
 - *duration:* Duration, in seconds of slide; defaults to 0.3.
- **Effect.CAF.SlideClose.Left:** Slide close from the left.
- **Effect.CAF.SlideClose.Top:** Slide close from the top.
- **Effect.CAF.SlideClose.Right:** Slide close from the right.
- **Effect.CAF.SlideClose.Bottom:** Slide close from the bottom.

Highlight

- **Effect.CAF.Shake:** Extends `Effect.Shake` with additional options:
 - *x:* Amount, in pixels to shake horizontally; defaults to 20.
 - *y:* Amount in pixels to shake vertically; default to 0.
 - *duration:* Duration in seconds of a single shake; defaults to 0.05.
 - *count:* Number of shakes; defaults to 5.
- **Effect.CAF.Highlight:** Fade in a halo, fade back to normal; options:
 - *duration:* Duration, in seconds of effect; defaults to 1.0.
- **Effect.CAF.Flicker:** Fade to white, fade back; options:
 - *duration:* Duration, in seconds of flicker; defaults to 1.0.

- *to*: Degree to which to fade, from 1.0 to 0.0, where 1.0 is a complete fade and 0.0 is no fade; defaults to 0.7.
- **Effect.CAF.Flash**: Combination flicker and highlight; options:
 - *duration*: Duration, in seconds of effect; defaults to 1.0.

Common Effect Options

- *duration*: Duration, in seconds of effect; defaults to 1.0.
- *transition*: Transition method to use for effect. This method is passed a number between 0.0 and 1.0, and returns a number between 0.0 and 1.0; defaults to `Effect.Transitions.sinoidal`.
- *from*: Transition start; defaults to 0.0.
- *to*: Transition end; defaults to 1.0.
- *queue*: Advanced queue options.
- *direction*: Transition direction, such as a string *top*; used only by directional effects.

Effect Transitions

You can specify a transition method for most effects using the effect's transition option. This method can modify the behavior of the effect by making the effect appear to speed up or slow down as it progresses, or even oscillate between the start state and the end state. The transition method is passed a number between 0.0 and 1.0, and it returns a number between 0.0 and 1.0. The default method is `Effect.Transitions.sinoidal`, which causes the effect to accelerate in the middle and slow as it ends.

Standard Scriptaculous Transitions

- **Effect.Transitions.linear**: Linear transition from 0 to 1.
- **Effect.Transitions.sinoidal**: Sine wave transition, accelerates in the middle, slows at the end, from 0 to 1.
- **Effect.Transitions.reverse**: Linear transition from 1 to 0.
- **Effect.Transitions.flicker**: Unstable transition between 0.5 and 1.
- **Effect.Transitions.wobble**: Oscillates between 0 and 1, converging on 0.5.
- **Effect.Transitions.pulse**: Five linear oscillations between 0 and 1.
- **Effect.Transitions.none**: Always returns 0.
- **Effect.Transitions.full**: Always returns 1.

Custom CAF Transitions

- **Effect.Transitions.CAF.pulse1**: Smooth single pulse from 0 to 1 and back to 0.
- **Effect.Transitions.CAF.flash1**: Pulses from 0 to 1 quickly, then from 1 to 0 more slowly.

Image URLs

Image controls that render icons or other images enable configuring the image or icon. Specify the image using an absolute URL or as a relative URL. The relative URL is relative to the root of the web application. The image control assumes any value starting with '/' is an image relative to the portlet application context. The following examples are absolute and relative URLs.

- **Absolute URL** `http://www.google.com/images/logo_sm.gif`
- **Relative URL** `/myimages/myicon.gif`

If you need to produce the image URL independently because some client-side code might dynamically modify the image source, you can produce the URL to an image or any other WAR resource with the following java code:

```
String url = "/myimages/myicon.gif";
FacesContext context = getFacesContext();
url = context.getApplication().getViewHandler().getResourceURL(context, url);
url = context.getExternalContext().encodeResourceURL(url);
```

You can automatically replace images with well-known URLs with skinned images.

To make the image URL relative to the My webMethods Server rather than relative to the portlet application context, put `fe:` at the front of your action link as shown in the following example:

```
fe:/meta/default/testmws___por1/activePageBean.exportImageButtonaction
```

Skinning

When you use image or icon controls, My webMethods Server runtime automatically replaces references to the images using your web application's `/skin/images` directory with skinned images, if the images are in the current user's skin. If the images are not in the current user's skin, the default images from your web application are used.

For example, say you have two images in your web application's `/skin/images` directory, `Icon_Error.gif` and `Icon_No-Error.gif`:

```
wm_myapp
  build
    src
      WebContent
        META-INF
          skin
            images
              Icon_Error.gif
              Icon_No-Error.gif
        WEB-INF
```

You have two image controls on your page, one with `/skin/images/Icon_Error.gif` for its URL property value, and one with `/skin/images/Icon_No-Error.gif` for its URL property value. If the current user's skin has an `Icon_Error` image, the first control displays the skin's `Icon_Error` image. If the current user's skin does not have an `Icon_No-Error` image, the second control displays the `Icon_No-Error` image from your web application's `/skin/images` directory.

For more information, see [“Image URLs” on page 253](#).

Client-Side Libraries

The core Composite Application Framework client-side JavaScript library, automatically included into every Composite Application Framework HTML page, is built on the open-source [Prototype](#) library. Prototype provides many convenience functions for manipulating strings, arrays, and DOM objects, as well as for making asynchronous requests. See [Using Prototype.js](#) for documentation on the Prototype library.

The core Composite Application Framework library provides a few extra prototype-style convenience functions, a facility for logging debug messages, a facility for dynamically loading other script files, a facility for displaying alert and progress dialogs, access to the [About the Client-Side Model](#), and access to the [Scriptaculous Effects](#) library.

Strings

Composite applications often use comma-separated value (CSV) strings to serialize lists for transfer between server and client.

The Composite Application Framework library extends the core JavaScript String object with extra methods:

- The `formatMessage(params)` method constructs a parameterized message similar to the `java.text.MessageFormat` class.
- The `escapeJS()` method ends the use of JavaScript blocks within programmatically generated HTML fragments.
- The `escapeRegExp()` ends the JavaScript for use by new `RegExp()` method.
- The `trim()` method deletes the leading and trailing white space.

Logger

The Composite Application Framework Logger sends client-side log messages to registered client-side log service similar to Apache's Log4j service. Each message has a category, an arbitrary string, and a level such as `Logger.ERROR`, `Logger.WARN`, `Logger.INFO`, or `Logger.DEBUG`. `ERROR`. The error messages are displayed using a modal dialog, and the `WARN` messages are displayed using a modeless dialog.

You can log messages using the following static functions:

- `Logger.error()`
- `Logger.warn()`
- `Logger.info()`
- `Logger.debug()`

■ `Logger.log()`

You can log the properties of an object at the DEBUG level using the `Logger.dump()` method.

Logger Bookmarklet

INFO and DEBUG log messages are not displayed in the user interface by default, but you can view them by installing the Composite Application Framework Logger bookmarklet from an instance of My webMethods Server:

```
http://your.server:port/wm_cafshared/ui/js/bookmarklet/
```

From that page, drag the CAF Logger link to your browser's bookmark tool bar. To use the bookmarklet, first navigate to a composite application page, and then click it. A CAF Logger window pops up, containing all the Composite Application Framework messages logged to the main window or tab, if you use Firefox or Internet Explorer.

You can clear the old log messages using the **Clear** button on the CAF Logger window. You can limit the log messages displayed by entering a regular-expression into the **Regex match** field. For example, if you type `Info` into the field and press ENTER, only messages with the string `Info` in them such as messages logged at the INFO level, are displayed. Delete the contents of the field and press ENTER to re-display all messages.

You can evaluate a line of JavaScript in the context of the main window or tab, if you use Firefox or Internet Explorer by entering some JavaScript into the **Quick eval** field at the bottom of the Composite Application Framework Logger window. For example, if you type `document.location.href` into the field and press ENTER, a message is logged with the main window's or tab, if you use Firefox or Internet Explorer page URL. When the **Quick eval** field is focused, you can use the up and down arrow keys to scroll through the list of previously entered commands similar to the history feature of a UNIX shell.

Library

The Composite Application Framework library dynamically loads script files on demand. With the `Library.register(id, url)` method, you can register a named group of script files to load later. The `Library.load(id, onload)` method loads the named group or directly-specified group of script files, and then executes the specified callback method when loading is completed. For example, the scripts needed to execute [Scriptaculous Effects](#) are pre-registered as the `scriptaculous/effects` group. To execute a scriptaculous effect, you must call `Library.load()` to load the scriptaculous effect library, pass a method to execute once the library has finished loading. For example:

```
Library.load("scriptaculous/effects", function() {
  Effect.Fade("myElement");
});
```

Instead of using a registered group name, you also can pass `Library.load()` the URL of a script to load directly. You can specify multiple URLs or named groups as a space-separated string, for example `http://example.com/foo.js http://example.com/bar.js`.

CAF.Dialog Class

A composite application can simulate the core JavaScript `alert()`, `confirm()`, and `prompt()` popup dialogs with a DHTML popup window. The main functional difference, other than the ability to display HTML-formatted messages is that the core JavaScript versions block program flow until the user closes the dialog, whereas the CAF.Dialog versions do not. If you want to perform some action as a result of the user input, you have to pass a callback method. For example, the following code prompts a user for a zodiac sign, and then responds to a click on the prompt's OK button by displaying a second (`alert`) dialog:

```
CAF.Dialog.prompt("What's your sign?", "Capricorn", function(value) {  
CAF.Dialog.alert("I'm a " + value + " too!");  
});
```

You can also apply validation to a `prompt()` dialog such as with the following example that does not let the user click OK without first entering a number between one and ten:

```
CAF.Dialog.prompt(  
'On a scale of 1-10, rate this prompt:',  
'10',  
function(value) {  
CAF.Dialog.alert('You entered ' + value + '.');  
},  
function(value) {  
CAF.Dialog.alert('You canceled!');  
},  
{  
  buttons: {  
    ok: {  
      validate: function(form) {  
        var n = parseInt(form.prompt);  
        if (n < 1 || n > 10)  
          return 'Please enter a number between 1 and 10.';  
        return '';  
      }  
    }  
  }  
}  
);
```

The `CAF.Dialog.show()` method can create more sophisticated dialogs with custom content. The following example displays an account creation dialog, with Username, Password, and Confirm Password fields, as well as a custom "?" button.

```
CAF.Dialog.show({  
  title: "Create an Account",  
  content: "Username: <input name='username' class='input10'>"  
+ "<br>"  
+ "Password: <input name='password' type='password' class='input10'>"  
+ "<br>"  
+ "Confirm Password: <input name='password2' type='password'"  
+ "class='input10'>",  
  buttons: {  
    ok: {  
      label: "Create",  
      "class": "button6",
```



```

defaultCommand: true,
fn: function(form) {
    CAF.Dialog.alert("Created new user " + form.username);
},
validate: function(form) {
    var errors;
    if (form.username.length < 8) {
        if (!errors) errors = {};
        errors.username =
            "Username must be at least 8 characters long.";
    }
    if (form.password.length < 8) {
        if (!errors) errors = {};
        errors.password =
            "Password must be at least 8 characters long.";
    }
    if (form.password != form.password2) {
        if (!errors) errors = {};
        errors.password2 =
            "Confirm Password must match original Password.";
    }
    return errors;
}
},
cancel: {
    label: "Cancel",
    "class": "button6"
},
popup: {
    label: " ? ",
    validate: function(form) {
        CAF.Dialog.alert("I dunno either");
        return {}; // prevent dialog from closing
    }
}
}
});

```

CAF.Progress Class

CAF can display a progress bar over any HTML element. The `CAF.Progress.overlay(element, msg)` method overlays a progress bar over an HTML element, whereas the `CAF.Progress.insert(element, msg)` method replaces the element's content with a progress bar. The `CAF.Progress.show(element, msg)` method decides for you whether to overlay or insert the progress bar. It inserts the progress bar if the element isn't large enough to contain the progress bar, by enlarging the element to the necessary size, or it overlays the bar. The `CAF.Progress.hide(element)` method hides the progress bar.

For example, the following line of JavaScript overlays a progress bar over the entire page:

```
CAF.Progress.overlay(document.body, "Please wait...");
```

The next line hides that progress bar:

```
CAF.Progress.hide(document.body);
```

CAF.Tooltip Class

You can programmatically attach a tool tip to any HTML element. The following code attaches a tool tip with the text "This control does something." to the control with the ID myControlId:

```
CAF.Tooltip.attachHover("#{activePageBean.clientIds['myControlId']}",  
"This control does something.");
```

CAF.Request Class

The `CAF.Request` class extends Prototype's asynchronous request class, `Ajax.Request`, allowing asynchronous JSF requests. Its API is identical to `Ajax.Request`, and you can use it in the same way as `Ajax.Request`. See the [Using the Ajax.Request class](#) Prototype documentation.

You should use `CAF.Request` instead of `Ajax.Request` when making requests to the JSF servlet, adding a scope property to the request options, the client ID value of the JSF view root's client ID. For example, given a URL and the `viewRootClientId`, the following code alerts the response:

```
new CAF.Request(url, {  
  method: 'get',  
  onComplete: function(transport) {  
    alert("response: " + CAF.Request.extractResponseText(transport.responseText));  
  },  
  scope: viewRootClientId  
});
```

To simulate a CAF command request, you can call the `_createFragmentURL()` method on the model object of a CAF control to create a URL that refreshes that control. For more information, see [“About the Client-Side Model” on page 264](#). You can convert the current state of the control's form to a URL parameter string with Prototype's `Form.serialize()` method. With those two strings and after setting the `CAF.Command.field()` value to the ID of the command control to invoke, you can create a new `CAF.Request` that posts the current form state to the server and render the updated control fragment as a result. For example:

```
var m = CAF.model("#{activePageBean.clientIds['myCommand']}');  
// set active command on form CAF.Command.field(m.form).value = m.id;  
// calculate url and form post parameters  
var url = m._createFragmentURL();  
var formParams = Form.serialize(m.form);  
// send request  
var request = new CAF.Request(url, {  
  method: "post",  
  parameters: formParams,  
  scope: CAF.viewRootId(m.id),  
  onComplete: method(transport) {  
    CAF.Dialog.alert(request.extractResponseText());  
  }  
});
```

CAF.Updater Class

The CAF.Updater class extends Prototype's asynchronous request class, Ajax.Updater, allowing updates to the content of an element using an asynchronous JSF request. Its API is identical to Ajax.Updater, and you can use it in the same way as Ajax.Updater. For more information, see the [Ajax.Updater class](#) Prototype documentation.

You should use CAF.Updater instead of Ajax.Updater when making requests to the JSF servlet, adding a scope property to the request options, the client ID value of the JSF view root. For example, given a URL and the ID of the element to update, the following code updates the element with the response content:

```
new CAF.Updater(id, url, {
  method: 'get',
  scope: CAF.viewRootId(id)
});
```

CAF.Draggable Class

The CAF.Draggable class extends the [Scriptaculous Effects](#) Draggable class, providing uniform drag behavior across different types of draggable elements. Unlike the default Draggable behavior, the CAF.Draggable object creates a copy of the element to drag and inserts it into a drag container, ensuring the user drags around the copy of the element and not the original. It uses the CAF.Draggable.duplicateAsHTML() method to generate unique temporary IDs for the dragged element and its children. If you want to override this behavior such as to replace the dragged element content with an image, you could override the startDrag() method to replace the default drag container's content with your custom content:

```
Object.extend(new CAF.Draggable(id), {
  startDrag: function(event) {
    CAF.Draggable.prototype.startDrag.apply(this, arguments);
    var container = this.getDragContainer();
    container.innerHTML = "<img
src='http://www.google.com/intl/en_ALL/images/logo.gif'>";
  }
});
```

The CAF.Draggable startDrag() and endDrag() methods also maintain a CAF.Draggable.current object while the drag is in process. The CAF.Draggable.current object has three useful fields:

- **draggable** - specifies the current draggable object
- **element** - specifies the original draggable element, not the one in the drag container that is the original drag target
- **result** - specifies an open-ended string representing the result of the drag operation

If the drag ends in a successful drop, and the drop handler is able to duplicate the dragged content, the drop handler sets the CAF.Draggable.current.result field to move. This notifies the CAF.Draggable object that it may delete the original draggable element, effectively performing a move operation. The CAF.Draggable object do this if the CAF.Draggable's move option is set to true.

The `CAF.Draggable` class also has two static functions, `model()` and `rows()`. These are especially useful in drop handler functions (see [CAF.Droppable Class](#)) to determine what is in the drag container. The `model()` method creates the drag container's content as a single element, useful for handling the drop of a simple control such as an image or text control. The `rows()` method creates the drag container's content as an array of table rows, useful for handling the drop of a complex drop such as a panel containing multiple controls, or a table or a list row.

Note:

You must load the scriptaculous effects package before referencing the `CAF.Draggable` and `CAF.Droppable` objects. For more information, see [“Scriptaculous Effects” on page 249](#).

You should wrap any effects, or drag and drop JavaScript code in an anonymous method, and pass that method to the `Library.load()` method:

```
Library.load('scriptaculous/effects', function() {  
  new CAF.Draggable(id, { cursor: "move" });  
});
```

CAF.Droppable Class

The `CAF.Droppable` class encapsulates and extends the functionality of the [Scriptaculous Effects](#) `Droppables.add()` method. Instead of using the `Droppables.add()` method and its options to register an element, similar to the following:

```
Droppables.add(id, { onDrop: function() {} });
```

You can use a syntax that mirrors that of the `Draggable` object constructor as in the following example:

```
Object.extend(new CAF.Droppable(id), {  
  onDrop: function(src, dst) {  
    CAF.Droppable.prototype.onDrop.apply(this, arguments);  
    CAF.Draggable.current.result = ""; // cancel move  
  }  
});
```

The `CAF.Droppable onDrop()` method copies the value of sub-controls with the same local ID from the dropped element into the sub-controls of the `CAF.Droppable`'s target element, the drop target.

Note:

You can have multiple controls that use the same ID, as long as each of those controls are in a different naming-container. For more information, see [“Control ID Reference” on page 244](#).

For example, you have two DIV tags, each in a different naming-container, each with three text controls. The first DIV is the target of a `CAF.Draggable` object. The second the target of a `CAF.Droppable`. The IDs of the text controls in the first DIV are name, description, and notes. The IDs of the text controls in the second DIV are item-number, name, and description. If the user drags and drops the first DIV onto the second, the `CAF.Droppable`'s `onDrop()` method copies the first DIV's name and description text control values to the name and description text controls of the second, but it does not modify the second DIV's item-number text control.

If the `CAF.Droppable`'s `onDropSetValue` option is set to `true`, the `onDrop()` method copies the value of the dropped control to the value of the target control, ignoring the sub-controls. For example, you have two image elements, the first image is the target of a `CAF.Draggable` object, the second image is the target of a `CAF.Droppable`, and with the `onDropSetValue` option set to `true`. If the user drags and drops the first image onto the second, the `onDrop()` method copies the first image control's value, the image element's URL, to the second image.

You can customize a `CAF.Droppable` object's drop handling by specifying a method for the `handleDrop` option. This method is invoked by the `CAF.Droppable`'s `onDrop()` method before it does anything else. The `handleDrop` callback method is passed the same two arguments as the original `onDrop()` method, `src` and `dst`. The `src` argument is the dragged (moved) element. The `dst` argument is the `CAF.Droppable`'s target element. The handler should return `true` if it fully handled the drop, and `false` to allow the default `onDrop()` processing to execute that copies the content of the dropped control to the target control.

The following example demonstrates using a custom `handleDrop` handler to override the default `CAF.Droppables` `onDrop()` behavior. It gets the value of the name sub-control in the dropped control, and sets the drop target control's value to the value of the dropped name sub-control:

```
new CAF.Droppable(id, {
  handleDrop: function(src, dst) {
    var rows = CAF.Draggable.rows(src);
    if (rows.length) {
      var nameId = rows[0].getControlId("name");
      if (nameId) {
        CAF.model(dst).setValue(CAF.model(nameId).getValue());
      }
    }
    return false;
  }
});
```

The `allowDrop` option is invoked by the `CAF.Droppable`'s `onActivate()` method. The `allowDrop` callback passes two arguments, `src` and `dst`. The `src` argument is the dragged element. The `dst` argument is the target element, that is the drop target. The handler returns `true` if the `src` element is movable, and `false` if it is not movable.

The following example demonstrates using a custom `allowDrop` method. It returns `true` if the drag container contains the phrase *my test*, and `false` if it does not contain *my test*:

```
new CAF.Droppable(id, {
  allowDrop: function(src, dst) {
    return !/my test/i.test(src.innerHTML);
  }
});
```


25 Understanding the Client-side Model

■ About the Client-Side Model	264
■ CAF.Model	264
■ CAF.Output.Model	264
■ CAF.Link.Model	265
■ CAF.Command.Model	265
■ CAF.Input.Model	266
■ CAF.Checkbox.Model	267
■ CAF.Select.Model	267
■ CAF.Table.Model	269
■ CAF.Tree.Model	271
■ Script Placement in the CAF Model	272

About the Client-Side Model

The JavaServer Faces (JSF) standard provides an abstract model used to interact programmatically with a user interface on the server. Composite Application Framework provides a proprietary client-side model that reflects the JSF server-side model. Use scripting information in Composite Application Framework to develop portlets using the client-side model, `CAF.model`.

CAF.Model

The client-side `CAF.model(id)` creates a model object for each control rendered on a page. All control IDs used by the Composite Application Framework client-side model are client-side IDs that are different from and related to server-side IDs that developers specify at design time. Client-side IDs are generated at run time. You can map client-side IDs in one of two ways:

- In your Java code, by calling the `getClientId()` method of the `javax.faces.component.UIComponent` object. This method gets the client-side ID of the control represented by the `UIComponent` object.
- In a binding expression, by referencing the `clientId` content provider on any page bean, `#{activePageBean.clientIds['myControlId']}`. This approach produces a client-side ID for any given server-side ID.

The client-side model objects produced by the `CAF.model()` enable manipulating controls without knowledge of the DHTML implementation of the controls. The model itself holds no state. Every invocation of `CAF.model()` creates a new model instance. The model simply exposes methods that enable examining and modifying the state of the modeled control.

All models have the following standard, read-only properties:

- `ID`
- `Element` - controls the primary HTML element
- `form.ID` - the client-side ID of the control

Form is the control's containing HTML form element when the control is contained by a form.

For more information about CAF controls, see *webMethods CAF Tag Library Reference*.

CAF.Output.Model

The base model class is `CAF.Output.Model`. Almost all control models are instances of the class or extend the class, including all `UIOutput` controls such as `Text`, and all `UIPanel` controls such as `BlockPanel`.

You can examine and modify the value of a `CAF.Output.Model` using the `getValue()` and `setValue(value)` methods. This is usually just the control element's `innerHTML` property. If the control is a `javax.faces.component.ValueHolder`, the value corresponds to the control's `value` property. For example, the value of an `Image` control is the image's *URL*. The following script example shows how to dynamically set the `Image` control with a control identifier of *ID* to display a company logo.


```
CAF.model(id).setValue("http://www.mycompany.com/logos/Logo_50wht.gif");
```

You can use the following methods to examine and modify the disabled, focused, and visible states of a `CAF.Output.Model`:

- `isDisabled()` and `setDisabled()`
- `isFocused()` and `setFocused()`
- `isVisible()` and `setVisible()`

Setting the disabled and focused states of an `UIOutput` or `UIPanel` control does not affect the control itself, but can affect descendant controls. The `show()`, `hide()`, and `toggle()` methods also modify the control's visibility.

You can refresh the contents of any control from the server using an asynchronous request that calls the model's `refresh()` method. For example, to refresh a panel, *myPanelId*, when the user double-clicks the panel, add a Script Block control to the page, and in its value property use the Prototype library's `Event.observe` method to register the double-click event handler, and call the `refresh()` method in that handler's body. The following snippet shows an example of this approach.

```
Event.observe("#{activePageBean.clientIds['myPanelId']}','dblclick',
function(event) {
CAF.model("#{activePageBean.clientIds['myPanelId']}').refresh();
});
```

You can configure the behavior of the `toggle()` server-side property.

For more information about CAF controls, see *webMethods CAF Tag Library Reference*.

CAF.Link.Model

Link controls extend the `CAF.Output.Model` with the `CAF.Link.Model`. The value of the `CAF.Link.Model` is the URL of the link. The `CAF.Link.Model` has one additional method, the `raise()` method. The method makes the link active and simulates clicking the link.

For more information about CAF controls, see *webMethods CAF Tag Library Reference*.

CAF.Command.Model

UI command controls extend the `CAF.Link.Model` with the `CAF.Command.Model`. The value of a `CAF.Command.Model` is the ID of the control. The `raise()` method is also available in the `CAF.Command.Model`. The `raise()` method makes the link active and simulates clicking the link. The `CAF.Command.Model` has a `go(params)` method that enables you to invoke the command directly, without simulating a user click action on the command button or link with the specified request parameters. For example, create a `HiddenCommand` with a server-side ID of *myCommandId*, and set a simple output button, and set its `onClick` property to the following:

When the user clicks the button, it submits the containing form using the `HiddenCommand`, with *myParam1* and *myParam2* as request parameters.

```
CAF.model("#{activePageBean.clientIds['myCommandId']}').go({
myParam1: "My Value One",
```

```
myParam2: "My Second Value"
});
```

The `CAF.Command.Model` enables registering the client-side action-listeners, which are invoked when the command is initiated after the form has been validated but before it is submitted. A client-side action-listener is a method that accepts one argument, the client-side ID of the command control. If the action-listener returns false, the command aborts and the form is not submitted. For example, the following JavaScript code adds an action-listener to the command with a server-side ID of *myCommandId*, and aborts the processing of the command until the command has been invoked five times:

```
var g_myCount = 0;
CAF.model("#{activePageBean.clientIds['myCommandId']}').addActionListener(function(id)
{
return (g_myCount++ > 5);
});
```

For more information about CAF controls, see *webMethods CAF Tag Library Reference*.

CAF.Input.Model

UIInput controls extend the `CAF.Output.Model` with the `CAF.Input.Model`. In addition to the functionality of the `CAF.Output.Model`, the `CAF.Input.Model` enables registering the client-side value change listeners and validators for a given control.

A client-side value change listener is a method that takes three arguments:

- the control's client-side ID
- the control's old value
- the control's new value

Value change listeners are called when the control's value changes, after the value has already changed.

For example, you create a Dropdown control for which each option value is the client-side ID of a **Hideable Panel** control. You only display the Hideable Panel that is currently selected by the Dropdown control. You can add a value change listener to the Dropdown that triggers when the user changes the Dropdown selection state. The user action hides the previously selected Hideable Panel and displays the newly selected Hideable Panel. The following code shows how you can use a value change listener:

```
CAF.model("#{activePageBean.clientIds['myDropdownId']}').addValueChangeListener
(function(id, oldValue, newValue) {
var oldModel = CAF.model(oldValue);
var newModel = CAF.model(newValue);
if (oldModel)
oldModel.hide();
if (newModel)
newModel.show();
});
```

A client-side validator is a method that takes two arguments: the control's client-side ID, and the control's value. It returns an empty string if the control is valid, and a non-empty, error-message

string if the control's value is invalid. For example, the following JavaScript registers a validator which validates that the input's value does not contain white space:

```
CAF.model("#{activePageBean.clientIds['myInputId']}').addValidator(function(id,
value) {
return (/s/.test(value) ? "Cannot contain whitespace." : "");
});
```

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

CAF.Checkbox.Model

Check boxes are used as part of a larger check box group, which are modeled by the `CAF.Select.Model`. However, the individual check box control, `UISelectBoolean` is modeled by the `CAF.Checkbox.Model`, which extends the `CAF.Input.Model`. With the check box model, `getValue()` returns the string `true` if the check box is checked, and an empty string, if the check box is not checked. You can take advantage of JavaScript's notion of *truthy* and *falsy*, where `true` equals boolean `true`, and `""` equals boolean `false`.

For example, if a `Checkbox` control has an ID of `myCheckboxId`, and next to a `TextInput` control has an ID of `myTextInputId`, you could disable and clear the text input. You want to disable and clear the text input if the check box is toggled unchecked, and enable the text input if the check box is toggled checked. You could apply the following value change listener to accomplish this:

```
CAF.model("#{activePageBean.clientIds['myCheckboxId']}').addValueChangeListener
(function(id, oldValue, newValue) {
var checked = newValue;
var input CAF.model("#{activePageBean.clientIds['myTextInputId']}='');
input.setDisabled(!checked); if (!checked) input.setValue(""); });
```

You can also use the `getLabel()` and `setLabel(label)` methods to dynamically get and set the check box's label.

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

CAF.Select.Model

`UISelectOne` and `UISelectMany` controls extend the `CAF.Input.Model` with the `CAF.Select.Model`. The value of a select-one control such as a dropdown and radio-button group, is the string value of the selected item, or an empty string, if no item is selected. The value of a select-many control such as a list box or a checkbox group, is the array of selected item values, or an empty array if no items are selected.

Individual select items are modeled by `CAF.Select.Item.Model`, which has `value`, `label`, `description`, `disabled`, `style` (inline CSS style), `styleClass` (CSS class name), and `iconURL` properties. Setting these properties does not affect the control directly. Call the `set(x, item)` method on the select control model with the updated select item in order to update the control. The first argument (`x`) of this method is the zero-based index of the item, or the string value of the item.

Symmetric to the `set(x, item)` method, the `get(x)` method retrieves the `CAF.Select.Item.Model` at either the specified zero-based index or with the specified string value. You can get the full list of `CAF.Select.Item.Model` items in the control using the `list()` method. The `selected()` method returns an array of the selected `CAF.Select.Item.Model` items.

You can add items to the control using the `add(x, item)` method. The first argument to this method (`x`) is the index string value, or the `CAF.Select.Item.Model` of the existing item. Use the `remove(x)` method to delete Items. The first argument to this method (`x`) is the index, string value or `CAF.Select.Item.Model` of the item to remove.

The following example appends and selects an Item from a dropdown. The first argument specified for the `CAF.Select.Item.Model`'s constructor is the value of the new item. The second argument is its label.

Note:

When reading the Javadoc for CAF JavaScript objects, the constructor for the object is the method named `initialize()` which is the convention defined by the Prototype library.

```
dropdown.add(new CAF.Select.Item.Model("value-ten", "Item Ten"));
dropdown.setSelected("value-ten", true);
```

You can select Items using the `setSelected(x, selected)` method. The first argument to this method (`x`) is the index, string value, or `CAF.Select.Item.Model` of the item to select. Test the selection state of an item using the `isSelected(x)` method. The first argument to this method (`x`) is the index, string value, or `CAF.Select.Item.Model` of the item to test.

The `CAF.Select.Model` model supports value change listeners that are notified every time the control's selection state changes. The `CAF.Select.Model` provides for selection change listeners. Selection change listeners are notified only when a change occurs to selection state of the specific item for which the selection change listeners are associated. The listener receives three arguments:

- The control ID
- The specific item value to listen for changes
- The new selection state, true for selected, false for not selected

The `addSelectionChangeListener(fn, value)` method enables registering the selection change listeners. The first argument to this method (`fn`) specifies the listener method, and the second argument (`value`) specifies the value of the item for which to listen. The following example adds a selection change listener to the `myDropdownId` control. If the value-on item is selected, the listener shows `myPanelOneId`; if its not selected, it hides `myPanelOneId`:

```
CAF.model("#{activePageBean.clientIds
  ['myDropdownId']}').addSelectionChangeListener(function
  (id, value, selected) {
var panel = CAF.model("#{activePageBean.clientIds['myPanelOneId']}');
if (selected)
panel.show();
else
panel.hide();
}, "value-one");
```

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

CAF.Table.Model

UI data controls extend the `CAF.Select.Model` with the `CAF.Table.Model`. You can manipulate rows in a table model in a manner similar to a select model.

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

CAF.Table.Row.Model

The `CAF.Table.Row.Model`, which models a row in a table, extends the `CAF.Select.Item.Model`, which models a particular item in a select control. The value property of a table row is its server-side row ID, the ID specified for each row by an `IAddressableTableContentProvider`. The label property of a table row is a CSV string of the row's HTML column content. The description, disabled, style, and styleClass properties all apply the same to a table row as they do to a select item. The icon property does not apply to a table row.

The `CAF.Table.Row.Model` also has a values property, which replaces the label property as the second argument to its constructor. The values property is a map; it maps the server-side IDs of the controls contained within the row to the values of those controls. So if a table contained a dropdown with an ID of `myDropdownId` and an output text control with an ID of `myOutputId`, the following example would append a new row to the table with a row-ID of `row-ten`, and set the row's dropdown to the `value-ten` item and the row's output text to display `Row Ten`, assuming the table had a template row, described by the next section:

```
table.add(new CAF.Table.Row.Model("row-ten", {myDropdownId: "value-ten",
myOutputId: "Row Ten"}));
```

A `CAF.Table.Row.Model` also has a few extra methods designed to help manipulate the row's contents. The `getControlId(localId)` method gets the ID of any control contained by the row, given its server-side ID. For example, you have a table with an ID of `myTableId`, and one of the columns within the table has a dropdown with an ID of `myDropdownId`. You can get a reference to the control in a specific table row, by creating a model for the row, and calling its `getControlId()` method with the dropdown's server-side ID:

```
var table = CAF.model("#{activePageBean.clientIds['myTableId']}");
var row = table.get("row-one");
var dropdownId = row.getControlId("myDropdownId");
var dropdown = CAF.model(dropdownId);
```

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

Template Row

If a table is bound to a content-provider that implements the `IUpdateableTableContentProvider` interface, and the content-provider returns true for the interface's `getCanTemplateRow()` method,

then the table automatically renders a hidden template row. If a row is added to the table using the CAF model, and the table does not have a template row, the new row is empty.

You can add content to the new table row using the raw HTML DOM or by setting the label property on the `CAF.Table.Row.Model`, and using the `CAF.Table.Model`'s `set(x, item)` method to update the row with the HTML column content specified by the label.

If a row is added to the table using the CAF model, and the table has a template row, the contents of the template row are copied to the new row. The template row contains the same controls as any other controls in the table, but it is initialized with the values from the content-provider in its template state.

The template state is set after `setTemplateRow()` method is called. With the template content in place, the row's value property is used to set the values of the all the controls in the new row to something other than their template state.

The `CAF.Table.Model`'s `template()` method returns the `CAF.Table.Row.Model` for the template row, if it exists.

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

Paging

The `CAF.Table.Model` has several methods for accessing the table's paging state. The `getFirst()` method returns the zero-based index of the first row on the current page. The `getRows()` method returns the current page size but the size might not equal the actual number of rows.

The `size()` method, inherited from the `CAF.Select.Model` returns the actual number of rows. The `getRowCount()` method returns the total number of rows in the table, or if the total is unknown, returns -1. The `getRowSelectedCount()` method returns the total number of selected rows in the table.

All of the `CAF.Table.Model` row manipulation methods inherited from `CAF.Select.Model`, such as `add()`, `remove()`, and `setSelected()` operate only on the current page. For example, calling `table.setSelected(0)` sets the first item of the table on the current page selected rather than the first item in the entire table. Two special methods, `go(first, rows, sort)` and `move(from, to)` operate across page boundaries, and use row indices relative to the entire table.

The `go(first, rows, sort)` method pages or re-sorts the entire table; it re-sorts the table by the columns, and in the order, specified by the sort string. It allows three arguments:

- The first argument (`first`) is greater or equal to zero, it pages the table to the index specified by the argument
- The second argument (`rows`) is greater than zero, it sets the new page size to the size specified by the argument
- The third argument is non-null and non-empty

The sort string is a comma-separated view (CSV) list of column IDs to sort, in order from the primary sort, to the secondary sort, and continuing using the specified sort order. Follow each

column ID with a plus (+) or minus (-) to indicate ascending order or descending order. For example, the sort string `user+,permissions+,modified-` specifies sorting the table by the user column in ascending order, then by the permissions column in ascending order, and by the modified column in descending order.

The `move(from, to)` method moves a row from the table index specified by the first argument (from) to the table index specified by the second argument (to). It does not page the table, even if one or both of the indices are outside of the current page. Note that the table content-provider must implement the `IReorderableTableContentProvider` interface, and return true for the interface's `isReorderable()` method to persist the row arrangement.

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

Listeners

Custom client-side code can listen for changes to the table by registering table-change and row-change listeners. Table-change events are raised when the table is paged, or when its select-all or none state is toggled. Row-change events are raised when a row is added, removed, updated, or selected using the `CAF.Table.Model`.

Add a table-change listener using the `addTableChangeListener(fn)` method. Its one argument (fn) is the listener method; when a table-change event is raised the listener method is passed two arguments: the table ID and the event type such as `page`, `select-all`, or `select-none`.

Add a row-change listener using the `addRowChangeListener(fn)` method. Its one argument (fn) is also the listener method; when a row-change event is raised the listener method is passed three arguments: the table ID, the row ID, and the event type such as `add`, `remove`, or `update`, or `select`. The following example alerts the user when a row in the table is selected:

```
var table = CAF.model('#{activePageBean.clientIds['myTableId']}');
table.addRowChangeListener(function(tableId, rowId, eventType) {
  if (eventType == "select") {
    var row = CAF.model(rowId);
    if (table.isSelected(row))
      alert(row.getValue() + " selected");
  }
});
```

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

CAF.Tree.Model

The `CAF.Tree.Model` extends the `CAF.Table.Model`, adding methods for expanding and collapsing tree rows. You can expand or collapse individual rows using `collapse(value)`, `expand(value)`, or `toggle(value)`, with their server-side row-ID (the `value` arg). Use the `getRoots()` method to retrieve the `CAF.Tree.Row.Model` roots.

You can determine the state of the tree rows using methods on the `CAF.Tree.Row.Model` that extend the `CAF.Table.Row.Model`. The `getParent()` method of the `CAF.Tree.Row.Model` returns the `CAF.Tree.Row.Model` for the parent of the row. The `getChildren()` method returns the array of

`CAF.Tree.Row.Model` children. The `isOpen()` method returns true if the row is expanded and false if collapsed. The `setOpen(open)` method expands or collapses the row. Unlike other `CAF.Select.Item.Model` and `CAF.Table.Row.Model` methods, the `setOpen()` method takes effect immediately, rather than when the container model's `set()` method is invoked with the changed model.

Add or remove roots from the tree using the `CAF.Select.Model`'s `add()` and `remove()` methods. You can add or remove rows using the `addChild()` and `removeChild()` methods of a row's `CAF.Table.Row.Model`. The `CAF.Select.Item.Model` and `CAF.Table.Row.Model` methods, `addChild()` and `removeChild()` take effect immediately rather than when the container model's `set()` method is invoked with the changed model.

A row template on a tree works the same as with the table. In the following example, if the tree can template rows, and it contains a dropdown with an ID of `myDropdownId` and a output text control with an ID of `myOutputId`, the following script would append a new row with a row-ID of `my-new-row` to the very last node in the tree, and set the row's dropdown to the "value-new" item and the row's output text to display `My New Row`:

```
var tree = CAF.model(treeId);
var node = tree.getRoots().last();
while (node && node.getHasChildren()) node = node.getChildren().last();
node.addChild(new CAF.Tree.Row.Model("my-new-row", { myDropdownId: "value-new",
myOutputId: "My New Row" }));
```

With an async tree, not all rows are present on the client. If a row has children, but its children are not yet on the client, the `CAF.Tree.Row.Model` returns true for `getHasChildren()`, but an empty array for `getChildren()`. Use the `CAF.Tree.Row.Model`'s `isLoading()` method as a shortcut to test for this state. The `isLoading()` method returns false if the row has children but the children are not on the client. To force the tree to request the children from the server, invoke `setOpen(true)` on the `CAF.Tree.Row.Model`, or invoke `expand(value)` on the `CAF.Tree.Model`.

Trees raise expand-all and collapse-all table-change events when the tree's expand-all and collapse-all state is modified. They raise toggle row-change events when individual rows are toggled expanded or collapsed. The async tree raises loaded row-change events when the children of a row are loaded.

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).

Script Placement in the CAF Model

The model wraps a DHTML control, when a page is rendered on the client. The control's HTML element must exist on page. That is, the browser must have parsed the HTML and included it in page's DOM before a model for that control is created. For example, if a script block on a page has some code which models a control as the page is being loaded, the script block must be placed after the modeled control. If the script block that models a control does so in a operation that is executed after the page is loaded such as the result of a mouse click on the control, you can have the script block placed anywhere on the page.

A common scenario for advanced scripting is to extract complicated JavaScript functions into an external script file, using the `IncludeScript` control. Client-side control IDs are generated by a complicated algorithm on the server and are not predictable ahead of time on the client. It is often

necessary to calculate client-side control IDs using the `clientIds` property of the active page bean `#{activePageBean.clientIds['myControlId']}`, while rendering the page, and then storing the IDs in global variables.

Note:

It is a best practice to store the control ID rather than the model instance. The model instance can cache information about the control and over time become out-of-sync with the actual control state.

A good way to prevent global variables defined by one portlet instance's page from colliding with variables from another's is to define a master global object that can be shared among all instances of the portlet on a page or all instances of a set of portlets on a page. For example, all webMethods Task Engine portlets might share a global object called WMTE, conditionally defined in a script block at the top of each portlet page:

```
// if the WMTE object does not exist, create a new empty WMTE object
if (WMTE) var WMTE = {};
```

In JavaScript, you can treat all objects as a generic map that maps arbitrary string keys, the object's property names, to arbitrary object values, the object's property values. You can then use the master global object to map arbitrary global variable names to arbitrary values. For example:

```
WMTE['taskCount'] = 32;
```

You can use a key unique to the portlet instance such as the portlet's view root ID to map a global variable to a dynamic value. For example:

```
WMTE['#{activePageBean.clientIds['myRootId']}' + '.taskCount']
= #{activePageBean.taskCount};
```

Any function defined in an external script file, as long as the function was passed the view root ID, can lookup the `taskCount` global variable specific to the current portlet instance. For example:

```
function wmte_doSomething(rootId) {alert(WMTE[rootId + '.taskCount']);}
```

Use this technique with function names as well. For example, an external script file might contain the following code:

```
if (!WMTE) var WMTE = {}; WMTE.doSomething = function(rootId)
{ alert(WMTE[rootId + '.taskCount']); }
```

Invoke this function as `WMTE.doSomething(rootId)` or as `WMTE['doSomething'](rootId)`.

For more information about CAF controls, see the *webMethods CAF Tag Library Reference*, as described in [“Finding Information about CAF Controls” on page 67](#).


26 Using Converters and Validators

■ About Using Converters and Validators with CAF Controls	276
■ Displaying Converters and Validators	276
■ Adding a Converter to a Control	276
■ Adding a Validator to a Control	277
■ Creating a Custom Converter	277
■ Creating a Custom Validator	278

About Using Converters and Validators with CAF Controls

With *converters*, you can format objects for presentation to the user for reading and modification, or represent objects for persistence in an application. You can apply converters to CAF input and output controls.

With *validators*, you can ensure that users specify correct values when using input controls.

A control can have only one converter but multiple validators. When you select a control on the design canvas in Software AG Designer that has at least one converter or validator, a push pin  is displayed at the upper right corner of the control.


The **CAF Core** node of the Palette view contains the standard JavaServer Faces (JSF) converters and validators, and the converters and validators included in Composite Application Framework. However, the **CAF Core > Converter** and **CAF Core > Validator** nodes are not displayed by default. For instructions about how to display converters and validators in the Palette view, see [“Displaying Converters and Validators” on page 276](#). For more information about the available converters and validators, see *webMethods CAF Tag Library Reference*.

In addition to using the converters and validators available in the Palette view, you can create and apply custom converters and validators.

Displaying Converters and Validators

The Palette view of Software AG Designer does not display converters and validators by default.

> To display converters or validators

1. In the Palette view, click , and then click **Customize**.
2. In the Customize Palette wizard, expand the **CAF Core** node and do the following:
 - To display converters, select the **Converter** node and clear the **Hide** check box.
 - To display validators, select the **Validator** node and clear the **Hide** check box.
3. Click **OK**.


Adding a Converter to a Control

Use the following procedure to add a converter to an input or output control.

> To add a converter to a control

1. Open a CAF view in the design canvas and locate the input or output control that you want to modify.

2. In the Palette view, expand the **CAF Core > Converter** node and select a converter.
3. Drag the converter to the design canvas and drop it onto the control.

If the  overlay is visible on the cursor, you cannot drop the converter.


4. To update properties for the converter:
 - a. In the design canvas, select the control with the converter.
 - b. In the Properties view, click the **Conversion** tab.
 - c. Specify new property values as needed.

Adding a Validator to a Control

Use the following procedure to add a validator to an input control.

➤ To add a validator to a control

1. Open a CAF view in the design canvas and locate the input control that you want to modify.
2. In the Palette view, expand the **CAF Core > Validator** node and select a validator.
3. Drag the validator to the design canvas and drop it onto the control.

If the  overlay is visible on the cursor, you cannot drop the validator.

4. To update properties for the validator:
 - a. In the design canvas, select the control with the validator.
 - b. In the Properties view, click the **Validation** tab.
 - c. Specify new property values as needed.

Creating a Custom Converter

To create a custom converter, you need Java coding experience. Composite Application Framework provides some initial code to get you started. A custom converter is valid only for the portlet in which it is created. If you plan to use a particular converter in many portlets, create your own library and load it through the project.

➤ To create a custom converter

1. Open a CAF view in the design canvas and select the control for which you want to create a custom converter.
2. In the Properties view, click the **General** tab.
3. In the **ID** field, type a unique ID for the converter.
4. In the design canvas, right-click the control, and then click **Lifecycle > Custom Converter**.

Composite Application Framework creates Java code in the managed bean and opens a Java editor to the location of the code. An example of converter code created for an input control is shown here:

```
public javax.faces.convert.Converter getInputID_converter()
{
    return new javax.faces.convert.Converter()
    {
        /**
         * Convert the input Object into a string.
         */
        public String getAsString( javax.faces.context.FacesContext context,
javax.faces.component.UIComponent component, java.lang.Object value)
        {
            // TODO: Convert the Object to a String here.
            return value.toString();
        }
        /**
         * Convert the input String into an Object.
         */
        public Object getAsObject( javax.faces.context.FacesContext context,
javax.faces.component.UIComponent component, java.lang.String value)
        {
            // TODO: Convert the String to an Object here.
            return "New Object: "+ value;
        }
    };
}
```

5. After the TODO comments, type the Java code required for the converter.

Creating a Custom Validator

To create a custom validator, you need Java coding experience. Composite Application Framework provides some initial code to get you started. A custom validator is valid only for the portlet in which it is created. If you plan to use a particular validator in many portlets, create your own library and load it through the project.

➤ To create a custom validator

1. Open a CAF view in the design canvas and select the input control for which you want to create a custom validator.

2. In the Properties view, click the **General** tab.
3. In the **ID** field, type a unique ID for the validator.
4. In the design canvas, right-click the control, and then click **Lifecycle > Custom Validator**.

Composite Application Framework creates some Java code in the managed bean and opens a Java editor to the location of the code. An example of validator code created for an input control is shown here:

```
/**
 * Validator for the control with id='inputID'
 */
public void inputID_validator(javax.faces.context.FacesContext
context, javax.faces.component.UIComponent component, java.lang.Object
value) {
// TODO: Check the value parameter here, and if not valid,
// do something like this:
// throw new ValidatorException(new FacesMessage("Not a valid value!"));
}
```

5. After the TODO comments, type the Java code required for the validator.

