# Developing Web Views for VMware vCenter Orchestrator

vRealize Orchestrator 5.5

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

**VMware, Inc.**
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

# Contents

# Developing Web Views for VMware vCenter Orchestrator

*Developing Web Views for VMware vCenter Orchestrator* provides information about developing Web views for VMware® vCenter Orchestrator.

Orchestrator Web views are Web 2.0 frontends that allow users to access Orchestrator workflows and objects in the Orchestrator inventory by using a Web browser rather than by using the Orchestrator client.

Orchestrator provides a standard Web view that users can use to run workflows, called weboperator. The weboperator Web view provides end users with browser access to all of the workflows in the library, that they can run on all of the objects in the inventory.

The Web components which Orchestrator provides can be used to develop custom Web views.

## Intended Audience

This information is intended for Web designers and developers who want to create or customize Web front ends for the Orchestrator processes, using the Web 2.0 technologies.

# Web View Overview

1

A Web view is a package of Web pages, style sheets, icons, and banners that represent a complete Web site. Web views can contain special Java Web Components (JWC) that add Orchestrator functions to the pages of the Web views. For example, you can add components that allow users to run workflows from a browser.

Orchestrator Web views update content dynamically without obliging users to reload complete pages. Orchestrator provides a library of Tapestry Framework 4.0 components to help you build customized Web views to access Orchestrator functions from a Web browser. Tapestry components provide access to objects in the Orchestrator server, such as the workflows in the library and the virtual machines in the inventory. You can also insert Dojo 0.4.1 components into Web views.

Orchestrator provides a Web view template that you can use as the basis for developing Web views. The Web view template contains skeleton HTML pages and Web view components that you can extend and adapt. You can also export existing Web views to use as templates that you can adapt to create new Web views.

You typically create or modify the pages of a Web view externally by using Web design tools. Creating or modifying Web pages independently of Orchestrator allows you to separate the Web design process from the process of developing Orchestrator Web view components. You import the Web view pages and components into the Orchestrator server and complete the process of creating the Web view in the Orchestrator client.

Developing Orchestrator Web views can require knowledge of some or all of the following Web development technologies and standards. For documentation about the different technologies, consult the Web sites of the organizations that maintain the standards.

- Cascading stylesheets (CSS). See http://www.w3.org/Style/CSS/.
- Ajax platform. See http://www.ajax.org/.
- Dojo toolkit. See http://www.dojotoolkit.org/.
- Java programming language. See http://www.oracle.com/technetwork/java/index.html.
- Java Web Components (JWC) from the Tapestry Framework. See http://tapestry.apache.org/.
- JavaScript Object Notation (JSON). See http://www.json.org/.

- Object-Graph Navigation Language (OGNL). See http://www.opensymphony.com/ognl/.

**Note** Third-party URLs are subject to changes beyond the ability of VMware to control. If you find a URL in VMware documentation that is out of date, notify VMware at docfeedback@vmware.com. You might be able to locate a third-party document by searching from the third-party home page.

# Weboperator Web View

2

Orchestrator provides a standard Web view called weboperator that allows users to run workflows from a browser.

The weboperator Web view provides an example of the orchestration functions that Web views can provide to end users in browsers, without requiring that those users use the Orchestrator client.

## Start the Weboperator Web View

You start the weboperator Web view from the Orchestrator client.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.

   The weboperator Web view and any other Web views that you have imported into Orchestrator appear.
3. Right-click weboperator and select **Publish**.
4. Open a browser and go to `http://orchestrator_server:8280`.

   In the URL, *orchestrator_server* is the DNS name or IP address of the Orchestrator server, and 8280 is the default port number where Orchestrator publishes Web views.
5. On the Orchestrator home page, click **Web View List**.
6. Click **weboperator**.
7. Log in using your Orchestrator user name and password.
8. Expand the hierarchical list of workflows to navigate through the workflows in the Orchestrator library.
9. Click a workflow in the hierarchical list to display information about the workflow in the right pane.

**10** In the right pane, select whether to run the workflow now or at a later time.

| Option | Action |
|---|---|
| **Run the workflow now** | a Click **Start Workflow** to run the workflow.<br>b Provide the required input parameters and click **Submit** to run the workflow. |
| **Run the workflow at a later time** | a Click **Schedule Workflow** to run the workflow at a later time.<br>b Provide the time, date, and recurrence information to set when and how often to run the workflow and click **Next**.<br>c Provide the required input parameters and click **Submit** to schedule the workflow. |

You can use the weboperator Web view to run workflows on objects in your inventory from a Web browser rather than from the Orchestrator client.

### What to do next

If you only need a Web view to access the inventory and run workflows, the standard weboperator Web view should meet your requirements. If you require more complex functionality from a Web view, you can use the Web components and default Web view template that Orchestrator provides to develop custom Web views.

# Web View Development Tasks to Perform in Orchestrator 3

You create the Web pages and Web view components that form an Orchestrator Web view by using Web development tools. You also use the Orchestrator client and configuration interface to perform many of the steps of Web view development.

When you develop Web views, you use the Orchestrator client to perform tasks such as creating skeleton Web views, declaring objects in the Orchestrator server as Web view attributes, exporting and importing files to and from working directories, and creating and using templates to create other Web views. You set the Orchestrator server to Web view development mode by using the Orchestrator configuration interface.

This section includes the following topics:

## Create a Web View Skeleton

You can create a Web view by creating a Web view skeleton. A Web view skeleton contains no HTML files or Web view components, and requires you to create these elements using Web development tools.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 Click the **Web Views** view.

3 Right-click in the white space in the Web view list and select **Add web view**.

4 Type a name for the Web view in the **Create Web View** text box and click **OK**.

5 Right-click the Web view in the Web view list and select **Edit**.

The Web view editor opens.

6 On the **General** tab, set the URL folder value to include a suffix for the URL on which Orchestrator will publish the Web view.

For example, if you set the URL folder to **`MyWebView`**, Orchestrator publishes the Web view at `https://`*`orchestrator_server`*`:8280/vco/vmo/`*`webview_name`*`/`, where *`orchestrator_server`* is the IP address or DNS name of the machine on which the Orchestrator server is running.

By default, the name of the URL folder matches the Web view name, but you can change this value.

**Note** If the Orchestrator server is running in Web view development mode, the URL folder value must match the name of the working folder in which you are developing the Web view.

7 Click the Version digits to increment the version number for the Web view.

The **Version Comment** dialog box opens.

8 Type a comment for this version of the Web view and click **OK**.

For example, type **`Initial creation`** if you created the Web view.

9 On the **General** tab, type a description of the Web view in the Description text box .

10 Click **Save and close** to close the Web view editor.

You created a Web view skeleton that does not yet contain any HTML pages or Web view components. If you export the Web view skeleton to a working folder, the only file it contains is the `VSO-WEBVIEW-INF\.webview.xml` file, which sets the Web view name and ID.

### What to do next

You must add HTML pages and Web view components to the Web view.

## Export a Web View as a Template

You can use an existing Web view as a template. You can export a Web view as a template, and then edit the exported template to create a Web view.

When you export a Web view as a template, Orchestrator creates a ZIP file that contains all the files of the original Web view. You can then create a new Web view that uses these files.

### Prerequisites

You must have an existing Web view to export as a template.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.
3. Right-click the Web view to export as a template and select **Templates > Export as template**.
4. (Optional) Change the name of the ZIP file as appropriate.
5. Select a location on your local system to save the ZIP file and click **Save**.

You exported the contents of an existing Web view to use as a template from which to create other Web views.

### What to do next

Create a new Web view from the template.

## Create a Web View from a Template

You can reduce the amount of development work by creating a Web view from a template.

A Web view template is a ZIP file that contains all the files and components of an existing Web view that you can use as the basis from which to create a new Web view. Orchestrator provides a default Web view template that you can use as the starting point for Web view development.

### Prerequisites

You must have exported an existing Web view to use as a template. Alternatively, you can use the default Web view template that Orchestrator provides.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.
3. Right-click in the white space in the Web view list and select **New from > File template**.
4. Navigate to a Web view template ZIP file and click **Open**.

   Orchestrator provides a default Web view template at the following location on the Orchestrator server.

   | Option | Action |
   |---|---|
   | **If you installed the standalone version of Orchestrator** | Go to `*install_directory*\VMware\Orchestrator\apps\webviewTemplates\default_webview.zip` |
   | **If vCenter Server installed Orchestrator** | Go to `*install_directory*\VMware\Infrastructure\Orchestrator\apps\webviewTemplates\default_webview.zip` |

5 Type a name for the new Web view in the **Create Web View** dialog box and click **OK**.

By default, the new Web view name is `web_view_template_name_FromTemplate`.

6 Right-click the Web view in the Web view list and select **Edit**.

The Web view editor opens.

7 On the **General** tab, set the URL folder value to include a suffix for the URL on which Orchestrator will publish the Web view.

For example, if you set the URL folder to **`MyWebView`**, Orchestrator publishes the Web view at `https://orchestrator_server:8280/vco/vmo/webview_name/`, where *`orchestrator_server`* is the IP address or DNS name of the machine on which the Orchestrator server is running.

By default, the name of the URL folder matches the Web view name, but you can change this value.

**Note** If the Orchestrator server is running in Web view development mode, the URL folder value must match the name of the working folder in which you are developing the Web view.

8 Click the Version digits to increment the version number for the Web view.

The **Version Comment** dialog box opens.

9 Type a comment for this version of the Web view and click **OK**.

For example, type **`Initial creation`** if you created the Web view.

10 On the **General** tab, type a description of the Web view in the Description text box .

11 Click **Save and close** to close the Web view editor.

You created a new Web view from a Web view template.

### What to do next

Export the contents of the new Web view to a working folder to modify them, and edit the Web view settings and attributes in the Orchestrator client.

## Define a Web View Template as a Resource Element

Instead of exporting a Web view to your local system for use as a Web view template, you can define a Web view template as a resource element in the Orchestrator server.

Defining a Web view template as a resource element makes it available to all Web view developers who connect to the Orchestrator server.

### Prerequisites

You exported a Web view template ZIP file to define as a resource element.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Design**.

2 Click the **Resources** view.

3 Right-click a resource folder in the hierarchical list or the root and select **New folder** to create a folder in which to store the resource element.

4 Right-click the resource folder in which to import the resource element and select **Import resources**.

5 Select the resource to import and click **Open**.

Orchestrator adds the resource element to the folder you selected.

You defined a Web view template ZIP file as a resource element that all Web view developers who connect to the Orchestrator server can use.

### What to do next

Create a Web view from a resource element template.

## Create a Web View from a Resource Element Template

Instead of creating a Web view from the beginning, you can create a Web view from a resource element template that you or another developer has imported to the Orchestrator server.

A Web view template ZIP file that a developer has defined as a resource element is available to all developers who connect to the Orchestrator server.

### Prerequisites

Orchestrator must define a Web view template that you or another developer has imported to the server as a resource element.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 Click the **Web Views** view.

3 Right-click in the white space in the Web view list and select **New from > Resource template**.

4 Press the Enter key in the **Filter** text box to display a list of all the resource elements that the Orchestrator server defines.

5 Select the Web view template ZIP file from the list of resource elements and click **Select**.

6 Provide an appropriate name for the new Web view in the **Create Web View** dialog box and click **OK**.

7 Right-click the Web view in the Web view list and select **Edit**.

The Web view editor opens.

8 On the **General** tab, set the URL folder value to include a suffix for the URL on which Orchestrator will publish the Web view.

   For example, if you set the URL folder to **`MyWebView`**, Orchestrator publishes the Web view at `https://`*`orchestrator_server`*`:8280/vco/vmo/`*`webview_name/`*, where *`orchestrator_server`* is the IP address or DNS name of the machine on which the Orchestrator server is running.

   By default, the name of the URL folder matches the Web view name, but you can change this value.

   **Note** If the Orchestrator server is running in Web view development mode, the URL folder value must match the name of the working folder in which you are developing the Web view.

9 Click the Version digits to increment the version number for the Web view.

   The **Version Comment** dialog box opens.

10 Type a comment for this version of the Web view and click **OK**.

   For example, type **`Initial creation`** if you created the Web view.

11 On the **General** tab, type a description of the Web view in the Description text box .

12 Click **Save and close** to close the Web view editor.

You created a new Web view from a Web view template that you or another developer has defined as a resource element.

### What to do next

Export the contents of the new Web view to a working folder to modify them, and edit the Web view settings and attributes in the Orchestrator client.

## Export Web View Files to a Working Folder

When you create a new Web view, either as a skeleton or from a template, you export the Web view files to a working folder on your local system for editing.

### Prerequisites

You must have created a new Web view in the Orchestrator client, either as a skeleton or from a template.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 Click the **Web Views** view.

3 Right-click the Web view in the Web view list and select **Export to directory**.

4 Select the working folder in your local system in which to develop the Web view and click **Export**.

The working folder in your local system now contains all the HTML, Web view component, image, and other files of the Web view.

### What to do next

You can edit and adapt the Web view files by using Web development tools.

**Note** To preview the Web view from the working folder while you develop it, set the Orchestrator server to Web view development mode.

## Configure the Server for Web View Development

During the Web view development process, you can configure the Orchestrator server to publish the Web view from a working folder rather than from the Orchestrator server.

When the server runs in development mode, you can preview the Web view as you develop it, without having to import it to the Orchestrator server to view it. You set the Orchestrator server to Web view development mode in the Orchestrator configuration interface.

**Note** Because Orchestrator publishes Web views from the working folder, you cannot access Web views that you have not exported to the working folder when the server is in development mode.

### Prerequisites

To enable Web view development mode, your working folder must be on the same machine as the Orchestrator server.

### Procedure

1. Log in to the Orchestrator configuration interface by using the your Orchestrator configuration username and password.

   For example, go to https://*orchestrator_server_DNS_name_or_IP_address*:8283 or https://localhost:8283 in a Web browser.

2. On the **General** tab click **Advanced Configuration**.
3. Select the **Enable Web view development mode** check box.
4. Type the path to the root of your working folder in the text box.

   Make sure you provide the path to the root of the working folder. Do not include the name of the folder that contains the Web view in the path.

   For example, if you are working on a Web view in the folder `C:\Documents and Settings\`*`username`*`\Desktop\MyWebView\`, type `C:\Documents and Settings\`*`username`*`\Desktop\` as the path.

5. Click **Apply changes**.
6. On the **Startup Options** tab, click **Restart Service** to restart the Orchestrator server in Web view development mode.
7. After the Orchestrator server has restarted, start the Orchestrator client and log in.
8. Click **Web Views**.

9. Verify that your Web view's URL folder value matches the name of your working directory.

   For example, if you created the working folder `C:\Documents and Settings\`*`username`*`\Desktop\MyWebView\`, set the URL folder to `MyWebView`.

   a. If the Web view is running, right-click the Web view and select **Unpublish**.
   b. Right-click the Web view and select **Edit**.
   c. On the **General** tab of the Web view editor, type the name of the working folder in the **URL folder** text box, and click **Save and Close** to close the Web view editor.

10. Right-click the Web view and select **Publish**.

You set the Orchestrator server to Web view development mode, in which you can preview a Web view from your working folder while you develop it.

# Import Web View Files from a Working Folder

After you edit the files of a Web view in the working folder, you must import them back to the Web view in the Orchestrator server.

### Prerequisites

Verify that you exported the files of a Web view to a working folder and edit them using Web development tools.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.
3. Right-click the Web view in the Web view list and select **Edit**.
4. Click the **Elements** tab in the Web view editor.
5. Click **Import from directory**.
6. Select the working folder in your local system from which to import the modified Web view files and click **Import**.
7. Click **Save and Close** to exit the Web view editor.

You imported to the Web view in the Orchestrator server the Web view files that you modified on your local system.

### What to do next

Create Web view attributes.

# Create a Web View Attribute

With Web view attributes, you can pass objects to Web view components. The functions that the Web view components define act on these objects to perform the orchestration actions that you run from the Web view.

A Web view attribute can be an object of any type that the Orchestrator API supports. For example, a Web view attribute can be a `VC:VirtualMachine` object. A Web view component can define a function that requires this object as an attribute. For example, when a user clicks a button in a Web view, a Web view component associated to that button runs a workflow that starts a virtual machine. A Web view attribute provides the virtual machine object to the workflow that the Web view component starts.

### Prerequisites

Create or import a Web view in the Orchestrator client.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.
3. Right-click the Web view and select **Edit**.
4. Click the **Attributes** tab in the Web view editor.
5. Right-click in the **Attributes** tab and select **Add attribute**.
6. Click the attribute name and type a name.
7. Click the attribute **Type** link and select the attribute type from the list.
8. Click the attribute **Value** link and type or select the value of the attribute.

   You type or select the attribute value depending on the type of the attribute.
9. Click **Save and Close** to exit the Web view editor.

You defined attributes that direct the Web view to the objects in the Orchestrator server on which it performs tasks.

### What to do next

Add a resource element to a Web view.

# Add a Resource Element to a Web View

Resource elements are external objects that you can import into the Orchestrator server for Web views to use as Web view attributes. Web view attributes identify objects with which Web view components interact.

### Prerequisites

Verify that you have the following objects in your Orchestrator server:

- An image, script, XML, or HTML file, or any other type of object imported into Orchestrator as a resource element.
- A Web view that requires the resource element as an attribute.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.
3. If the Web view is running, right-click the Web view to which you want to add the resource element and select **Unpublish**.
4. Right-click the Web view and select **Edit**.
5. Click the **Attributes** tab.
6. Right-click within the **Attributes** tab and select **Add attribute**.
7. Click the attribute name and type a new name for the attribute.
8. Click **Type** to set the attribute type.
9. In the **Select a type** dialog box, type `resource` in the **Filter** box to search for an object type.

| Option | Action |
|---|---|
| **Define a single resource element as an attribute** | Select `ResourceElement` from the list. |
| **Define a folder that contains multiple resource elements as an attribute** | Select `ResourceElementCategory` from the list. |

10. Click **Value** and type the name of the resource element or category of resource elements in the **Filter** text box.
11. From the proposed list, select the resource element or a folder containing resource elements and click **Select**.
12. Click **Save and close** to exit the editor.

You added a resource element or folder of resource elements as an attribute in a Web view.

# Disable Web View Development Mode

If you set the Orchestrator server to Web view development mode during the development process, you must set the Orchestrator server back to its normal mode before you can publish the Web view.

### Prerequisites

You must have set the Orchestrator server to Web view development mode and finished modifying the Web view files in your working folder.

### Procedure

1. Log in to the Orchestrator configuration interface by using the your Orchestrator configuration username and password.

   For example, go to https://*orchestrator_server_DNS_name_or_IP_address*:8283 or https://localhost:8283 in a Web browser.

2. On the **General** tab click **Advanced Configuration**.
3. Deselect the **Enable Web view development mode** check box.
4. Click **Apply changes**.
5. On the **Startup Options** tab click **Restart Service** to restart the Orchestrator server in normal mode.

You disabled Web view development mode. Orchestrator now publishes Web views from the Orchestrator server, rather than from the working folder.

### What to do next

Publish the Web view.

# Publish a Web View

When you finish Web view development and import the modified files to the Web view in the Orchestrator server, you can publish the Web view.

### Prerequisites

You must have a Web view that is ready for publishing. You must also have disabled Web view development mode.

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. Click the **Web Views** view.
3. Right-click the Web view to publish and select **Publish**.

Orchestrator publishes the Web view at `https://`*`orchestrator_server`*`:8280/vco/vmo/`*`web_view_url_folder/`*. The IP address or DNS name of the machine on which the Orchestrator server is running is *orchestrator_server*. The name of the Web view URL folder is *web_view_url_folder*.

# File Structure of a Web View

4

When you develop a Web view, you must save the collection of Web pages and Web view components that comprise the Web view to a working folder. The Web view working folder must conform to basic file-naming and file-structuring rules.

You can name the working folder in which you develop the Web view pages and components any name that is appropriate. The working folder must contain the following file and folder at its root.

Table 4-1. Web view file structure

| File | Description |
|---|---|
| `<WebView_Folder>\default.html` | The home page of the Web view. All Web views must include a `default.html` file at the root of the working folder. |
| `<WebView_Folder>\components\` | Contains the JWC files and the associated HTML templates of the Web view components. The `components` folder must be at the root of the working folder. |

**Important** If you create more than one Web view to run in the same Orchestrator server, you must save the Web view components in subfolders inside the `components` folder, to avoid conflicts between identically named components. Alternatively, create all Web view components with a unique name.

The `default.html` file and the `components` folder are the only mandatory elements that a Web view must contain. You can add other files and folders in the Web view folder and organize the files and folders in any way. You can include HTML files that are not Web view component templates anywhere in the Web view folder.

# Web View Home Page

5

All Web views must contain a file named `default.html`, that you must save at the root of the Web view working folder. The `default.html` file is the home page of the Web view.

The `default.html` file is the point of entry to a Web view. The default Web view template that Orchestrator provides contains a skeleton `default.html` file that you can adapt and extend. The following code extract shows the contents of the `default.html` file from the default Web view template.

```
<vco jwcid="@layout/MyBorder" section="literal: home" title="Home">
    <!--  Content of the homepage -->
        <h2 style="margin-left: 16px; margin-top: 0px; padding-top:18px;">
                Welcome to the default Web view template
        </h2>
        <p style="margin-left: 16px;">
                This Web view template is a base for your own Web view development.
        </p>
</vco>
```

### Table 5-1. Contents of the Web View Template default.html File

| Code | Description |
|---|---|
| `<vco></vco>` tags | You can insert Web view components in any type of HTML tag. The Web view template wraps all its content in `<vco>` tags, to show that this code is specific to Orchestrator. |
| `jwcid="@layout/MyBorder"` | A reference to the `MyBorder` Web view component from the default Web view template. The `MyBorder` component defines the borders of the Web view pages. |
| `title="Home"` | The title that appears in the title bar of the Web view home page. |

The rest of the code in the `default.html` file is standard HTML. You can extend and adapt the content of the home page by adding HTML code and add functions to the page by adding Web view components.

# Web View Components 6

Web view components add Orchestrator functions to Web pages. For example, you can add Web view components to Web pages that allow users to run workflows from a Web page in a browser.

You build Orchestrator Web views by adding JWC components to HTML Web pages. Orchestrator provides a library of JWC Web view components that add predefined orchestration functions to Web views. The JWC Web view components that Orchestrator provides conform to the Tapestry Framework 4.0 standard.

In addition to the library of Web view components that Orchestrator provides, you can use every standard component from the Tapestry Framework 4.0 in Web views.

- Tapestry Web View Components

  With the Tapestry Web view components in the Orchestrator Web view component library, you can add orchestration functions to Web views. The Tapestry Web components in the Orchestrator library define actions that access objects in the Orchestrator server.

- Creating Tapestry Web View Components

  With Orchestrator, you can create custom Tapestry Web view components to perform orchestration functions from Web pages. A Tapestry Web view component conforms to the Tapestry Framework standard version 4.0.

- Orchestrator Tapestry Component Library

  Orchestrator has a library of Tapestry components that you can reference in Web views. You can also use all of the components that the Tapestry Framework 4.0 standard defines.

## Tapestry Web View Components

With the Tapestry Web view components in the Orchestrator Web view component library, you can add orchestration functions to Web views. The Tapestry Web components in the Orchestrator library define actions that access objects in the Orchestrator server.

The Tapestry Web view components that Orchestrator provides add functions to Web views such as obtaining and displaying the properties of an object in the server, starting workflows, or obtaining information from the user.

You add Tapestry components to a Web view by adding a `jwcid` attribute to an HTML tag in a Web page. When you reference a Web view component, you prefix the name of the component with the `@` character. Certain Web view components require you to set additional properties when you set the `jwcid` attribute.

## Add a Tapestry Component in an HTML Page

You add Tapestry components to a Web view by adding a `jwcid` attribute to an HTML tag in a Web page. The `jwcid` attribute references a Web view component.

You can add a `jwcid` attribute to any HTML tag. You can add references to components from the Orchestrator Web view component library, to components from the Tapestry Standard, or to custom components that you create.

### Prerequisites

Create a Web view in the Orchestrator client and exported its contents to a working directory.

### Procedure

1 Open an HTML page of a Web view in an HTML editor.

2 Add an arbitrary tag to the HTML file, in the position at which the Web view component is to appear in the page.

For example, add the following arbitrary tag in the appropriate position in the HTML file:

```
<vco>
```

3 Add to the arbitrary tag a `jwcid` attribute that references a Web view component.

For example, the following `jwcid` attribute adds the `vco:DisplayProperty` component from the Orchestrator library to the Web view.

```
<vco jwcid="@vco:DisplayProperty">
```

The `vco:DisplayProperty` component obtains and displays the properties of an object that is in the server in the Web view.

4 Add the additional properties that the component requires to the arbitrary HTML tag.

For example, the following Web view component displays the `MyVirtualMachine` virtual machine `Name` property in a Web view.

```
<vco jwcid="@vco:DisplayProperty" name="Name" property="MyVirtualMachine"/>
```

You added a reference to a Web view component to a Web page in a Web view.

# Creating Tapestry Web View Components

With Orchestrator, you can create custom Tapestry Web view components to perform orchestration functions from Web pages. A Tapestry Web view component conforms to the Tapestry Framework standard version 4.0.

A Tapestry Web view component must contain a component specification file and a component template.

**Important** The Tapestry component template file and the component specification must have the same name. For example, if you name a component template `MyComponent.html`, you must name the associated component specification `MyComponent.jwc`. Web view components that you use in different Web views that run in the same server must have unique names.

You must save the component files in the `components` folder in the Web view file structure. If you create subfolders in the `components` folder, you must specify the full path to a component when you set the `jwcid` attribute in HTML pages. For example, if you include a `MyBorder` component in a *<WebView_Folder>*`\components\layout\` subfolder, you must set the `jwcid` attribute, as the following example shows:

```
<div jwcid="@layout/MyBorder">
```

You can precede the `@` character with a unique identifier. With the unique identifier, you can reuse the class throughout the HTML page, by referencing the unique identifier.

In the following example, the component is `Border` and the unique identifier is `myBorderComponent`.

```
<div jwcid="myBorderComponent@MyBorder">
```

- Tapestry Component Specification File

  A Tapestry component specification file is a JWC file that refers to the Tapestry DTD definition and to the Java class that specifies the behavior of the component.

- Tapestry Component Template File

  A Tapestry component template file is an HTML file that defines the layout of a Web view component.

- WebviewComponent Class

  The `ch.dunes.web.webview.WebviewComponent` class is the main class for Web view components. All Web view component specification JWC files must implement this class.

- WebviewPage Class

  The `ch.dunes.web.webview.WebviewPage` class provides methods that you call in OGNL expressions in Web view component template HTML files.

- WebObjectComponent Class

  The ch.dunes.web.webview.components.WebObjectComponent class provides methods to obtain information from objects in the Orchestrator server. The WebObjectComponent class extends WebviewComponent.

## Tapestry Component Specification File

A Tapestry component specification file is a JWC file that refers to the Tapestry DTD definition and to the Java class that specifies the behavior of the component.

The JWC file can also set the initial values of the Web view component properties.

Orchestrator Web views implement the following Java classes:

- ch.dunes.web.webview.WebviewComponent
- ch.dunes.web.webview.WebviewPage
- ch.dunes.web.webview.components.WebObjectComponent.html

The name of the Tapestry component specification file must match the name of the component specification JWC file.

### Example: Web View Template Access.jwc File

The following example shows a component specification file that implements the WebviewComponent Java interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE component-specification PUBLIC
"-//Apache Software Foundation//Tapestry Specification 4.0//EN"
"http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">
<component-specification class="ch.dunes.web.webview.WebviewComponent">
</component-specification>
```

## Tapestry Component Template File

A Tapestry component template file is an HTML file that defines the layout of a Web view component.

The name of the Tapestry component template file must match the name of the component specification JWC file.

### Example: Web View Template Access.html File

The following example shows a component template file that displays the user's username and adds a **Logout** link to a Web view page.

```
<strong>
<vco jwcid="@Insert" value="ognl:page.user.displayName" />
</strong>
<vco jwcid="@Insert" value="ognl:' (' + page.getUsername() + ')'" />
&nbsp;|&nbsp;
<a jwcid="@Any" href="ognl:page.webviewUrl + '?logout'">Logout</a>
```

## WebviewComponent Class

The `ch.dunes.web.webview.WebviewComponent` class is the main class for Web view components. All Web view component specification JWC files must implement this class.

Implementing the `WebviewComponent` class in an Orchestrator Web view component allows you to call methods in a Web view page to perform various functions in the Orchestrator server, such as retrieving attributes, making queries, getting and setting parameters and attributes, and implementing Dojo widgets in the Web view component.

The `WebviewComponent` class extends the `org.apache.tapestry.BaseComponent` Tapestry class. The `BaseComponent` class provides the implementation for all Tapestry components that implement an HTML definition file.

The `WebviewComponent` class defines the following methods.

Table 6‑1. Methods of the `WebviewComponent` class

| Method | Returns | Description |
|---|---|---|
| `getWebviewPage()` | `ch.dunes.web.webview.WebviewPage` | Returns the `WebviewPage` object of the page that contains this component. |
| `getWebview()` | `ch.dunes.model.webview.WebView` | Returns the `WebView` object that represents the current Web view. |
| `getRequestCycle()` | `org.apache.tapestry.IRequestCycle` | The `IRequestCycle` object is the Tapestry object that controls every access to the server. |
| `getWebVisitor()` | `ch.dunes.web.webview.WebVisitor` | The `WebVisitor` object contains information about the Web view user for the server to use. |
| `getParameter(java.lang.String parameterName)` | `java.lang.Object` | Returns a query parameter value, or null if no query parameter is provided in the request. If multiple values are provided, it returns the first value. |
| `getParameters(java.lang.String parameterName)` | `java.lang.Object[]` | Retrieves an array of values for a query parameter. |

### Table 6-1. Methods of the `WebviewComponent` class (Continued)

| Method | Returns | Description |
|---|---|---|
| `objectToJson(java.lang.Object object)` | `java.lang.String` | Returns a JSON representation of the object as a parameter. |
| `objectToParam(java.lang.Object object)` | `java.lang.String` | Returns a parameter string to identify an object. |
| `getAttribute(`<br>`java.lang.String attributeName,`<br>`java.lang.Object defaultValue)` | `java.lang.String` | Returns a Web view attribute, or the default value if the attribute is null or not set. |
| `getAttribute(java.lang.String attributeName)` | `java.lang.Object` | Returns a Web view attribute. |
| `translateParameters(java.util.List parametersValues)` | `java.lang.Object[]` | If the object value is a string that begins with "`attribute:`", `translateParameters` translates an array of objects into a `FinderResult` object. If the string does not begin with "`attribute:`", it does nothing. |
| `getClientId()` | `java.lang.String` | Obtains the ID of the client. |
| `setClientId(java.lang.String id)` | void | Sets the ID of the client. |
| `getDojoSource()` | `org.apache.tapestry.IAsset` | Returns the source of any Dojo widgets in the Web view as a Tapestry `IAsset` object. |
| `getDojoPath()` | `org.apache.tapestry.IAsset` | Returns the path to any Dojo widgets in the Web view as a Tapestry `IAsset` object. |
| `getBrowser()` | `ch.dunes.web.Browser` | Returns a `Browser` object that contains information about the browser in which the user accesses the Web view. |
| `addQueryParameter(`<br>`java.lang.String url,`<br>`java.lang.String parameterName,`<br>`java.lang.Object parameterValue)` | `java.lang.String` | Adds a parameter to a server query. |
| `addQueryParameter(`<br>`boolean condition,`<br>`java.lang.String url,`<br>`java.lang.String parameterName,`<br>`java.lang.Object parameterValue)` | `java.lang.String` | Adds a parameter to a server query if a given condition is met. |

The `WebviewComponent` class inherits the following methods from `class java.lang.Object`:

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`,`wait`, `wait`, `wait`

### Constructor

```
public WebviewComponent()
```

## WebviewPage Class

The `ch.dunes.web.webview.WebviewPage` class provides methods that you call in OGNL expressions in Web view component template HTML files.

The `WebviewPage` class extends the Tapestry class `org.apache.tapestry.html.BasePage`.

The `WebviewPage` class defines the following methods:

### Table 6-2. Methods of the `WebviewPage` class

| Method | Returns | Description |
|---|---|---|
| `addQueryParameter(`<br>`boolean condition,`<br>`java.lang.String url,`<br>`java.lang.String parameterName,`<br>`java.lang.Object parameterValue)` | `java.lang.String` | Adds a parameter to a server query if a given condition is met. |
| `addQueryParameter(`<br>`java.lang.String url,`<br>`java.lang.String parameterName,`<br>`java.lang.Object parameterValue)` | `java.lang.String` | Adds a parameter to a server query. |
| `executeAction(java.lang.String actionAttributeName)` | `java.lang.Object` | Runs an action in the server and returns the result. |
| `executeAction(`<br>`java.lang.String`<br>`actionAttributeName,`<br>`java.lang.Object actionParameters)` | `java.lang.Object` | Runs an action in the server and returns the result. |
| `getAbsoluteUrl(java.lang.String relativeUrl)` | `java.lang.String` | Returns the absolute URL from a relative URL. |
| `getAttribute(java.lang.String attributeName)` | `java.lang.Object` | Returns the Web view attribute of the specified name. Use this method instead of `getAttributes().get(String)` because it returns an exception if it does not find the attribute. |
| `getAttribute(`<br>`java.lang.String attributeName,`<br>`java.lang.Object defaultValue)` | `java.lang.Object` | Returns the Web view attribute of the specified name. |
| `getAttributes()` | `java.util.HashMap` | Returns a hash map containing the Web view attributes. |
| `getBaseURL()` | `java.lang.String` | Returns the URL of the Web view. |

### Table 6-2. Methods of the `WebviewPage` class (Continued)

| Method | Returns | Description |
|---|---|---|
| `getBrowser()` | `ch.dunes.web.Browser` | Returns a `Browser` object that contains information about the browser in which the user accesses the Web view. |
| `getDojoPath()` | `org.apache.tapestry.IAsset` | Returns the path to any Dojo widgets in the Web view as a Tapestry `IAsset` object. |
| `getDojoSource()` | `org.apache.tapestry.IAsset` | Returns the source of any Dojo widgets in the Web view as a Tapestry `IAsset` object. |
| `getPageUrl()` | `java.lang.String` | Returns the URL of the current page without the URL parameters. |
| `getPageUrlWithQuerryString()` | `java.lang.String` | Returns the URL of the current page with the URL parameters. |
| `getParameter(java.lang.String parameterName)` | `java.lang.Object` | Returns a query parameter value, or null if no query parameter is provided in the request. If multiple values are provided, it returns the first value. |
| `getParameters(java.lang.String parameterName)` | `java.lang.Object[]` | Retrieves an array of values for a query parameter. |
| `getRequest()` | `Abstract HttpServletRequest` | Returns HTTP servlet requests. |

The `WebviewPage` class inherits the following methods from `class java.lang.Object`:

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`,`wait`, `wait`, `wait`

## Fields

- `public static java.lang.String DEFAULT_FLASH_TYPE`
- `public static java.lang.String LOGIN_MESSAGE_ATTRIBUTE`
- `public static java.lang.String DEFAULT_LOGIN_MESSAGE`

## Constructor

`WebviewPage()`

# WebObjectComponent Class

The `ch.dunes.web.webview.components.WebObjectComponent` class provides methods to obtain information from objects in the Orchestrator server. The `WebObjectComponent` class extends `WebviewComponent`.

You use the `WebObjectComponent` class in conjunction with `vmo:ListPane` components.

The `vmo:ListPane` component inserts a list of objects into a Web view. To display information about an object in the list in another Web view page or panel, the HTML file that displays that information must contain a Web view component that implements the `WebObjectComponent` class.

The `WebObjectComponent` class defines the following methods that obtain properties from objects in the Orchestrator server.

Table 6-3. Methods of the `WebObjectComponent` class

| Method | Retruns | Description |
|---|---|---|
| `get(java.lang.String name)` | `java.lang.Object` | Obtains the property of the given name. |
| `get(`<br>`java.lang.String name,`<br>`java.lang.String valueIfNotFound` | `java.lang.Object` | Obtains the property of the given name. |
| `toParam()` | `java.lang.String` | Obtains the output parameter of an `Action` or `Workflow` object. |

The `WebObjectComponent` class inherits the following methods from `class java.lang.Object`:

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`,`wait`, `wait`, `wait`

## Constructor

`WebObjectComponent()`

## Example: Using WebObjectComponent to Display Object Information

The following `vmo:ListPane` component displays information about the objects it lists in an HTML page called `panel.html`:

```
<p jwcid="@vmo:ListPane"
                    action="getVirtualMachineList"
                    actionParameters="attribute:vmFolder"
                    detailUrl="./panel.html"/>
```

The `panel.html` file contains a reference to a Web view component called `DisplayVmInfo`:

```
<vmo jwcid="@DisplayVmInfo" urlParameter="itemId" />
```

The `DisplayVmInfo.jwc` component specification file implements the `WebObjectComponent` Java class:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE component-specification PUBLIC
"-//Apache Software Foundation//Tapestry Specification 4.0//EN"
"http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">
<component-specification class="ch.dunes.web.webview.components.WebObjectComponent">
</component-specification>
```

The `DisplayVmInfo.html` component template file uses `WebObjectComponent.get()` methods in OGNL statements to display object properties in `panel.html`:

```
<table width="200" border="1">
  <tr>
    <td>Virtual machine name</td>
    <td><vmo jwcid="@Insert" value="ognl:get('name')"/></td>
  </tr>
  <tr>
    <td>Object ID</td>
    <td><vmo jwcid="@Insert" value="ognl:get('id')"/></td>
  </tr>
</table>
```

## Orchestrator Tapestry Component Library

Orchestrator has a library of Tapestry components that you can reference in Web views. You can also use all of the components that the Tapestry Framework 4.0 standard defines.

All of the Tapestry components in the Orchestrator library have the prefix `vco:`, to distinguish these components from the components that the standard Tapestry framework provides.

The components in the Orchestrator Web view component library require different properties to display different types of information in the Web view. In the property tables for each component, asterisks (*) denote mandatory properties.

### vco:DisplayProperty Component

The `vco:DisplayProperty` component displays the names and values of the properties of objects in the Orchestrator inventory.

#### Properties

The `vco:DisplayProperty` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `name` * | String | Property name |
| `property` * | String | Property value |

#### Example: vco:DisplayProperty Component

The following example shows how to use the `vco:DisplayProperty` component to display the details of a virtual machine in a Web view.

```
<div jwcid="@vco:DisplayProperty" name="Name" property="MyVirtualMachine"/>
<div jwcid="@vco:DisplayProperty" name="Id" property="vm_89575"/>
<div jwcid="@vco:DisplayProperty" name="State" property="poweredOff"/>
```

# vco:IfMemberOf Component

The `vco:IfMemberOf` component includes a block of content if the current user is a member of a given LDAP group.

If you pass an array of LDAP groups to this component, the Web view displays the content if the current user is a member of at least one of the groups in the list.

## Properties

The `vco:IfMemberOf` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `attribute` * | String | An attribute of the `LdapGroup` type, or an array of `LdapGroup` objects. |

## Example: vco:IfMemberOf Component

The following example shows how to use the `vco:IfMemberOf` component to add information about a user's LDAP group membership to a Web view.

```
<span jwcid="@vco:IfMemberOf" attribute="ognl:'adminGroup'">
You are a member of the group that the adminGroup attribute defines.
</span>
<span jwcid="@Else">
You are not a member of the group that the adminGroup attribute defines.
</span>
```

# vco:IncludeJavascript Component

The `vco:IncludeJavascript` component inserts a `<script>` tag that defines a URL to a JavaScript file, to add a JavaScript function to a Web view.

You insert the `vco:IncludeJavascript` component in the `<head>` tags of a Web view page.

## Properties

The `vco:IncludeJavascript` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `src` | String | The path to a Javascript file, with or without the `.js` extension |

### Example: vco:IncludeJavascript Component

The `.js` file extension is optional. The following example code lines both return the same `src` value.

```
<script jwcid="@vco:IncludeJavascript" src="dojo"/>
```

```
<script jwcid="@vco:IncludeJavascript" src="dojo/dojo.js"/>
```

## vco:IncludeStylesheet Component

The `vco:IncludeStylesheet` component inserts a `<link>` tag that links to an external CSS file in a Web view.

You insert the `vco:IncludeStylesheet` component in the `<head>` tags of a Web view page.

### Properties

The `vco:IncludeStylesheet` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `href` or `src` | String | Name of a CSS file, with or without the `.css` extension. |
| `attribute` | String | Name of a Web view attribute that contains a path to the CSS files. |

The `href` or `src` properties are unnecessary if you provide an `attribute` property. If you store CSS files in a dedicated folder in the Web view file structure, you can include the folder name in the path you provide to the `href` or `src` property.

### Example: vco:IncludeStylesheet Component

The `.css` file extension is optional. The following example code lines both return the same `href` or `src` value.

```
<link jwcid="@vco:IncludeStylesheet" href="default.css"/>
```

```
<link jwcid="@vco:IncludeStylesheet" href="default"/>
```

The following example code line references a Web view attribute that contains a path to a CSS file. You create Web view attributes in the Orchestrator client.

```
<link jwcid="@vco:IncludeStylesheet" attribute="cssDefault"/>
```

# vco:IncludeWorkflowHeader Component

You use the `vco:IncludeWorkflowHeader` component with the `vco:WebformContainer` to display a Web form in a Web view.

You can insert a Web form in a Web view to request information from the user when they start a workflow, or to prompt the user for information during the workflow run.

## Properties

The `vco:IncludeWorkflowHeader` component defines the following property.

| Name | Type | Description |
|---|---|---|
| `debug` | Boolean | Set the Dojo debugging mode |

You insert the `vco:IncludeWorkflowHeader` component in the `<head>` tags of a Web view page. You insert the associated `vco:WebformContainer` component in the `<body>` tags of the Web view page.

## Example: vco:IncludeWorkflowHeader Component

The following example shows how to use the `vco:IncludeWorkflowHeader` component in a Web view.

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

<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <span jwcid="@vco:IncludeWorkflowHeader"/>
</head>

<body jwcid="@Body">
    <span jwcid="@WebformContainer"/>
</body> </html>
```

# vco:ListPane Component

The `vco:ListPane` component displays a list of objects in a Web view, and displays an object's details when the user selects the object in the list.

## Properties

The `vco:ListPane` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `attribute` | String | A Web view attribute that references an array of objects that you define in the Orchestrator client. The array of objects appears in the `vco:ListPane` component in the Web view. |
| `action` | String | A Web view attribute that references an action. The action must return an array of objects. |
| `actionParameters` | Array of objects | Array of parameters that an action requires. If the parameter is a Web view attribute, use the syntax `attribute:myAttribute`. |
| `url` | String | The URL of the content of the list. Returns a JavaScript Object Notation (JSON) string. |
| `detailUrl` | String | The URL of the HTML page in which to display the details of an object in the list. The HTML page that `detailUrl` refers to must include a reference to a Web view component that implements the `WebObjectComponent` class. |
| `detailParameterName` | String | Name of the parameter that stores the ID of the item. Default is `itemId`. |
| `updateFrequency` | Integer | Time in milliseconds before the list of objects refreshes. Default is 0. If you do not set the property, the list never refreshes. |
| `showHeader` | Boolean | Displays the filtering table header. Default is `false`. |
| `jsonIdKey` | String | Name of the JSON ID key. |
| `jsonStateKey` | String | Name of the JSON state key. Default is `state`. |
| `jsonNameKey` | String | Name of the JSON name key. Default is `name`. |
| `jsonTypeKey` | String | Name of the JSON type key. Default is `type`. |
| `sizerWidth` | Integer | Width of the sizer method. Default is 9. |
| `sizeShare` | Integer | Width or height of a child of a `SplitContainer`. The value is relative to the `sizeShare` properties of other children. Default is 6, with the other columns set to 10. |
| `showState` | Boolean | Shows a state column if set to `true`. Default is `false`. |

You must set at least one and only one of the `attribute`, `url`, or `actions` properties. If you do not set an `attribute`, `url`, or `actions` property, or if you set more than one of these properties together, the Web view returns an error.

### Example: vco:ListPane Component

The following example displays a list of virtual machines in a pane of a Web view.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    version="-//W3C//DTD XHTML 1.1//EN" xml:lang="en">

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>
        <span jwcid="@Insert" value="ognl:webview.name">Page Title</span>
         | Home
    </title>
</head>

<body>
    <div jwcid="@Border">
        <h1>Virtual Machine List</h1>
    <div jwcid="vmList@vco:ListPane"
            action="getVirtualMachineList"
            actionParameters="attribute:vmFolder"
            detailUrl="system/partials/virtual_machine.html" >
                Select a virtual machine on the left to display it.
    </div>
    </div>
</body>
</html>
```

## vco:Login Component

With the `vco:Login` component, you can customize the login page of a Web view.

The login page of a Web view must adhere to the following rules.

- You must name the login page `login.html`.
- You must save the login page at the root of the Web view file structure.

### Properties

The `vco:Login` component has no properties.

### Example: vco:Login Component

The following example code line adds a login link to a `login.html` page.

```
<span jwcid="@vco:Login">login here</span>
```

## vco:PageAccessControl Component

The `vco:PageAccessControl` component allows or denies users access to the Web view page that contains this component. The `vco:PageAccessControl` component checks the membership of the Web view user to an LDAP group.

If the user is a member of at least one group that the `deny` attribute defines, the Web view denies the user access to the page. If the user is not a member of a group in the `deny` attribute, the component checks the `allow` attribute.

If the user is a member of at least one group that the `allow` attribute defines, the user can access the page. Otherwise, the Web view does not display the page.

You set the LDAP groups of users who can view the page as Web view attributes in the Orchestrator client.

### Properties

The `vco:PageAccessControl` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `deny` | String | A Web view attribute of the type `LdapGroup`, or an array of `LdapGroup` objects. |
| `allow` | String | A Web view attribute of the type `LdapGroup`, or an array of `LdapGroup` objects. |
| `redirectUrl` | String | A URL to which to redirect the user if they are not authorized to view the page. If `redirectUrl` is not set, the Web view returns a 403 error. |
| `message` | String | If `redirectUrl` is set and `message` is set, the URL of the page to which the Web view redirects the user contains a `msg`query parameter and the contents of the `message` property. For example `*error_page.html*?msg=*message_content*`. |

### Example: vco:PageAccessControl Component

The following example code line allows access to a page to users who are members of the group that the `adminGroup` Web view attribute defines.

```
<span jwcid="@vco:PageAccessControl" allow="adminGroup"/>
```

The following example code line denies access to a page to users who are members of the group that the `partnerGroup` Web view attribute defines.

```
<span jwcid="@vco:PageAccessControl" deny="partnerGroup"/>
```

The following example code line redirects users who are members of the `partnerGroup` LDAP group to an error page. The `error401.html` file is at the root of the Web view file structure.

```
<span jwcid="@vco:PageAccessControl" deny="partnerGroup" redirectUrl="error401.html"/>
```

## vco:TaskAction Component

The `vco:TaskAction` component displays the scheduled action from a `Task` object. A user selects the task from a list that a `vco:ListPane` component generates.

### Properties

The `vco:TaskAction` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `stringValue` | String | The `stringValue` of the task that the user selects. Every object in the Orchestrator server has a `stringValue` string representation. |
| `type` | String | The type of the task that the user selects. |
| `attribute` | String | A Web view attribute. The `vco:TaskAction` displays the possible actions to perform on the object that corresponds to this Web view attribute. |
| `action` | String | A Web view attribute of type `Action`. The `vco:TaskAction` component displays the possible actions to perform on the object that this action returns. |
| `actionParameters` | List | A list of parameters for the action. |
| `object` | Object | An object. The `vco:TaskAction` component displays the possible actions to perform on this object. |
| `urlParameter` | String | The parameter name in the URL that represents the task the user selects. |

You can only set the following parameters or combinations of parameters. Setting other combinations of parameters results in an error.

- `stringValue` and `type`
- `attribute`
- `action` and `actionParameters`

- `object`
- `urlParameter`

### Example: vco:TaskAction Component

The following example code line shows how to use the `vco:TaskAction` component with the `stringValue` and `type` parameters, if the Java interface that your Web view component or page references defines a `getMyStringValue()`method.

```
<div jwcid="@vco:TaskAction" type="ognl:myType" stringValue="ognl:myStringValue"/>
```

The following code line shows how to use the `vco:TaskAction` component with the `object` parameter, if the Java interface that your Web view component or page references defines a `getMyObject()`method.

```
<div jwcid="@vco:TaskAction" object="ognl:myObject"/>
```

## vco:WebformContainer Component

The `vco:WebformContainer` component adds a Web form to a Web view, for users to complete when they run a workflow or to allow users to provide information to a workflow during its run.

### Properties

The `vco:WebformContainer` component defines no properties.

You can use the `vco:WebformContainer` component with the `vco:WorkflowLink` component.

### Example: vco:WebformContainer Component

To add a `vco:WebformContainer` component to a Web view, you must also include a `vco:IncludeWorkflowHeader` component in the `<head>` tag of the Web view page, as the following example shows.

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

<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <span jwcid="@vco:IncludeWorkflowHeader"/>
</head>

<body jwcid="@Body">
    <span jwcid="@WebformContainer"/>
</body> </html>
```

## vco:WorkflowLink Component

The `vco:WorkflowLink` component adds a link to a Web view to allow users to run a workflow. You can also use this component to display a link to schedule a workflow.

### Properties

The `vco:WorkflowLink` component defines the following properties.

| Name | Type | Description |
|---|---|---|
| `workflow` | String | Web view attribute of type `Workflow`. |
| `action` | String | Web view attribute of type `Action`. |
| `actionParameters` | List | A list of parameters for the action. |
| `object` | Object | A `Workflow` object. |
| `workflowId` | String | ID of the `Workflow` object. |
| `returnUrl` | String | URL to which to redirect the user after starting the workflow.The default is the URL of the page that contains the component. |
| `onReturn` | String | Run a JavaScript method when the workflow starts. For example, to display information about the running workflow in the Web view. |
| `cancelUrl` | String | URL to which to redirect the user if they cancel a workflow. The default is the URL of the page that contains the component. |
| `onCancel` | String | Run a JavaScript method if the user cancels the workflow. |
| `isSync` | Boolean | If `true`, the workflow runs in synchronous mode. Default is `false`. |
| `returnUrlAttribute` | String | Workflow attribute containing a URL to which to redirect users after the workflow finishes its run. Default is `returnUrl`. |
| `webformPage` | String | URL of a page that contains a `vco:WebformContainer` component, to open a Web form when a user starts a workflow. Default is `system/form.html`. |
| `isDialog` | Boolean | If `true`, the Web form opens in a dialog box. Default is `false`. |
| `width` | Float | Width of the dialog box. Values less than 1 define a ratio in relation to the width of the window. Values greater than 1 define the size in pixels. Default is `0.5`. |

| Name | Type | Description |
|---|---|---|
| `height` | Float | Height of the dialog box. Values less than 1 define a ratio in relation to the height of the window. Values greater than 1 define the size in pixels. Default is `0.9`. |
| `isScheduled` | Boolean | If `true`, when the workflow starts, the user is prompted for a time and date at which to run the workflow. Default is `false`. |
| `isAutostart` | String | If `true`, the workflow runs with preset default parameters without displaying them to the user. If set to `never`, the workflow never runs in autostart mode. If false, the workflow runs according to the `Autostart` property you set in the workflow presentation. Default is `false`. |
| `defaultValues` | `HashMap<String, String>` | A hashmap of default parameter values. If `isAutostart` is `true`, these parameters are not seen by the user when the workflow runs. If `isAutostart` is `false`, the workflow opens a parameters dialog box containing these default parameters. The key is the name of the parameter and the value is its string representation. |

The properties of the `vco:WorkflowLink` component must conform to the following rules:

- Set only one of the `workflow`, `action`, `object`, or `workflowId` properties.
- If you set the `returnUrlAttribute` property, you must set `isSync` to true.
- Use the syntax `#{'element1','element2'}` to construct a list of properties in OGNL.
- Use the syntax `#{"foo":"foo value", "bar":"bar value"}` to construct a map in OGNL.

## Example: vco:WorkflowLink Component

The following code example adds a link to a Web view that starts an action when the user clicks it.

```
<span jwcid="@vco:WorkflowLink"
 action="myAction"
    actionParameters="ognl:{'attribute:myParameter',
            'attribute:myParameter2'}">
            Click here
</span>
```

# 7 Accessing Server Objects from URLs

You can add URLs to Web view pages to access objects in the Orchestrator server, without having to implement a Web view component. For example, you can add URLs to Web view pages that run an action in the Orchestrator server or URLs that retrieve resource elements from the Orchestrator server.

- Running Actions from URLs

  Running actions from URLs rather than by implementing Web view components allows you to run operations directly in the Orchestrator server without requiring any input parameters from the Web view user. Running actions from URLs also allows you to define how to process the action results.

- Accessing Resource Elements from URLs

  Workflows and Web views can require as attributes objects that you create independently of Orchestrator. To use external objects as attributes in workflows or Web views, you import them into the Orchestrator server as resource elements.

## Running Actions from URLs

Running actions from URLs rather than by implementing Web view components allows you to run operations directly in the Orchestrator server without requiring any input parameters from the Web view user. Running actions from URLs also allows you to define how to process the action results.

When you run a workflow from a Web view component, the Web view prompts you for input parameters before it runs. Web view components also apply formatting to any data they receive from the server. If you retrieve objects from the server from a URL rather than from a Web view component, you can apply your own formatting or processing to the data that the URL retrieves.

For example, certain actions return JavaScript Object Notation (JSON) data. If you call such an action from a URL in a Web view, you can write Ajax functions to process and format the action results. Similarly, if the action returns an object from the Orchestrator server, you can write JavaScript functions to act on this object in the Web view.

To run an action from a URL in a Web view, you must declare that action as a Web view attribute and reference the attribute in the URL.

Web views access actions that you have defined as Web view attributes at the following URL:

```
https://orchestrator_server:8280/vco/vmo/web_view_url_folder/system/execute/action/action.html?
action=action_attribute_name
```

To avoid hard-coding the *orchestrator_server* address into the URL, you can provide relative paths to the action that start from the directory below the Web view URL folder, as the following path shows:

```
<a href="./system/execute/action/action.html?action=attribute_name"></a>
```

## Setting Action Parameters in a URL

You pass input parameters to the action by setting the `actionParameters` property in the URL. Depending on the action, you might need to declare the input parameters for the action as Web view attributes. If the action parameters are Web view attributes, you must prefix the Web view attribute name of the parameter with `attribute:`, as the following example shows.

```
https://orchestrator_server:8280/vco/vmo/web_view_url_folder/system/execute/action/action.html?
action=action_attribute_name&actionParameters=attribute:parameter_attribute_name&actionParameters=param
eter_value
```

## Writing Action Results to a File

You can use a URL to run an action that returns its results as a `MimeAttachment` file. You provide the name and type of the file to which to write the results in the URL. You must define the action to run as a Web view attribute.

Web views access actions that you have defined as Web view attributes and obtain their results as `MimeAttachment` files at the following URL:

```
https://orchestrator_server:8280/vco/vmo/web_view_url_folder/system/execute/action/
action_attribute_name/filename.file_extension?actionParameters=parameter_value&mimetype=mime_type
```

The *filename.file_extension* file you specify for the output file can be any type of file. If you set the optional `mimeType` property, the file type must be a valid `MimeAttachment` file, for example an EML or PDF file.

To avoid hard-coding the *orchestrator_server* address into the URL, you can provide relative paths to the action that start from the directory below the Web view URL folder, as the following path shows:

```
<a href="./system/execute/action/generateReport/annualReport.pdf?actionParameters="Annual
Report"&mimetype=application/pdf"></a>
```

The preceding example URL performs the following tasks:

- Runs an action that you have declared as the Web view attribute `generateReport`, which returns a `MimeAttachment` object.
- Creates a PDF file called `annualReport.pdf`
- Returns the PDF file

## Run an Action from a URL

You can add URLs to Web view pages to run actions in the Orchestrator server. Running actions by using direct URLs rather than by using Web view components allows you to specify the parameters with which to run the action in the URL, and allows you to provide your own formatting to the action results.

### Prerequisites

You have created a Web view in the Orchestrator client and have Web view page in which to insert a URL to run an action.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 In the **Web views** view, right-click the Web view to which to add the URL and select **Edit**.

   You must unpublish a running Web view to edit it.

3 Right-click in the **Attributes** tab of the Web view editor and select **Add attribute**.

4 Create a Web view attribute, of type `Action` with the value set to the action of your choice.

   For example, name the Web view attribute **`MyAction`**.

5 Create Web view attributes for each of the input parameters that the action requires.

   For example, name a Web view attribute **`ActionParameterAttribute`**.

6 Click **Save and close** to exit the Web view editor.

7 Open the HTML page in which to insert the URL in an HTML editor.

8 Add a link to `https://`*`orchestrator_server`*`:8280/vco/vmo/`*`web_view_url_folder`*`/system/vmo/pages/action.html` at the appropriate place in the HTML file.

   Add a relative link to the action URL, that starts below the Web view URL folder value, rather than hard-coding the address of the Orchestrator server in the URL, as the following example shows:

```
<a href="./system/vmo/pages/action.html?action=myAction
      &actionParameters=attribute:ActionParameterAttribute
      &actionParameters=action_parameter_value">
      Click here to run an action</a>
```

   The URL uses the Web view attributes you set in the Orchestrator client to start the action `myAction` and to set the `ActionParameterAttribute`parameter. The second parameter, `ActionParameterValue`, is not a Web view attribute so you add the parameter value directly in the URL.

You added a link to run an action from a URL. When users click the link in the published Web view, the action runs with the parameters you reference in the URL. JavaScript functions that you define can process the results of running the action.

# Accessing Resource Elements from URLs

Workflows and Web views can require as attributes objects that you create independently of Orchestrator. To use external objects as attributes in workflows or Web views, you import them into the Orchestrator server as resource elements.

You can add URLs to Web view pages to retrieve resource elements from the Orchestrator server. You can then add JavaScript functions to the Web view to process the resource elements. To access a resource element from a URL in a Web view, you must declare that resource element as a Web view attribute and reference the attribute in the URL.

Web views access resource elements at the following URL:

`https://`*`orchestrator_server`*`:8280/vco/vmo/`*`web_view_url_folder`*`/system/resources/attributes/`*`resource_attribute_name`*`.att`

To avoid hard-coding the *`orchestrator_server`* address into the URL, you can provide relative paths to the action that start from the directory below the Web view URL folder, as the following path shows:

```
<a href="./system/resources/attributes/resource_attribute_name.att"></a>
```

## Obtain a Resource Element from a URL

Resource elements are files that are imported into the Orchestrator server for use by Orchestrator applications. By accessing a resource element from a URL in a Web view, you make that resource element available to processing that you define in the Web view.

Objects that workflows and Web views can use as resource elements include image files, scripts, XML templates, HTML files, and so on. Any workflows or Web views that run in the Orchestrator server can use any resource elements that you import into Orchestrator.

### Prerequisites

You must have performed the following tasks:

- Imported a file into the Orchestrator server to use as a resource element
- Created a Web view in the Orchestrator client
- Created a Web view page in which to insert a URL to run an action

### Procedure

1. From the drop-down menu in the Orchestrator client, select **Administer**.
2. In the **Web views** view, right-click the Web view to which to add the URL and select **Edit**.

   You must unpublish a running Web view to edit it.
3. Right-click in the **Attributes** tab of the Web view editor and select **Add attribute**.

4. Create a Web view attribute, of type `ResourceElement` with the value set to the resource element of your choice.

   For example, name the Web view attribute **`MyImage`** if the resource element is an image file.

5. Click **Save and close** to exit the Web view editor.

6. Open the HTML page in which to insert the URL in an HTML editor.

7. Add a link to `https://orchestrator_server:8280/vco/vmo/web_view_url_folder/system/resources/attributes/resource_attribute_name.att` at the appropriate place in the HTML file.

   Add a relative link to the resource element URL, that starts below the Web view URL folder value, rather than hard-coding the address of the Orchestrator server in the URL, as the following example shows:

   ```
   <a href="./system/resources/attributes/MyImage.att">
         Click here to obtain an image</a>
   ```

The URL uses the Web view attribute you set in the Orchestrator client to obtain the resource element in the Web view.

# Create a Simple Web View Using the Default Template 8

The easiest way to create a Web view is to use a template. Orchestrator provides a default Web view template to help you create Web views.

You can use the default template to create a simple Web view called Virtual Machine Manager. You can find a ready-made version of the Virtual Machine Manager sample Web view in the bundle of Orchestrator examples.

**Note** If you install the Virtual Machine Manager Web view from the Orchestrator samples bundle, you must edit the `vmFolder` attribute to point to a virtual machine folder in your vCenter Server before you publish the Web view.

### Procedure

1 Import the Default Web View Template

To create a Web view by using the default Web view template, you must import the template to the Orchestrator client.

2 Export the Virtual Machine Manager Web View to a Working Folder

You edit the HTML files and Web view components of the Virtual Machine Manager Web view on your local system, using Web development tools.

3 Provide Unique Component Names

To avoid conflicts if you run multiple Web views in the same Orchestrator server, you must make sure that all Web view components have unique names.

4 Configure the Server for Web View Development

During the Web view development process, you can configure the Orchestrator server to publish the Web view from a working folder rather than from the Orchestrator server.

5 Edit the Virtual Machine Manager Web View Home Page

You create the home page of the Virtual Machine Manager Web view in the `default.html` file. The `default.html` file must be at the root of the working directory.

6 Add a vco:ListPane Component to the Web View Home Page

You can add a `vco:ListPane` component to the Virtual Machine Manager Web view home page to display a list of vCenter Server objects.

7 Define the Web View Attributes for the vco:ListPane Component

The `vco:ListPane` component refers to Web view attributes that provide an action to obtain an array of virtual machines from the server and the virtual machine folder from which to obtain them.

8 Create a Web View Component to Display Virtual Machine Information

The `vco:ListPane` component lists virtual machines in the left side of the Web view. You can create a Web view component to show information about each virtual machine in a Web view panel on the right side.

9 Create a Web View Tab by Using the Menu Component

The default Web view template provides a `Menu` component that you can use to create navigation tabs in a Web view.

10 Add Links to Run Workflows from a Web View by Using the vco:WorkflowLink Component

You add links to run workflows from a Web view by using the `vco:WorkflowLink` component. You define the workflows to run by setting Web view attributes.

11 Customize the Web View Interface

You can customize the appearance of the Web view by adjusting the `custom.css` style sheet and changing the images in the `\images` folder.

12 Publish the Virtual Machine Manager Web View

After you finish developing the Virtual Machine Manager, you must disable Web view development mode, import the Web view from your working directory to the Orchestrator server, and publish the Web view.

# Import the Default Web View Template

To create a Web view by using the default Web view template, you must import the template to the Orchestrator client.

When you create Web views, it is easier to customize an existing template than to start from the beginning.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 Click the **Web Views** view.

3 Right-click in the white space under the Web views list and select **New from > File template** to import a Web view template to the Orchestrator server.

4 Click **Browse** and browse to the appropriate folder.

| Option | Action |
|---|---|
| **If you installed the standalone version of Orchestrator** | Go to `C:\Program Files\VMware\Orchestrator\apps\webviewTemplates\default_webview.zip`. |
| **If you installed Orchestrator with the vCenter Server installer** | Go to `C:\Program Files\VMware\Infrastructure\Orchestrator\apps\webviewTemplates\default_webview.zip`. |

5 Select the `default_webview.zip` file and click **Open**.

6 Name the Web view **`Virtual Machine Manager`**.

The Virtual Machine Manager Web view now appears in the list in the **Web Views** view in the client.

7 Right-click **Virtual Machine Manager** and select **Publish** to preview the empty template.

8 In a browser, go to `http://`*`orchestrator_server`*`:8280/vmo/` to see the list of Web views running on the Orchestrator server.

9 Click **Virtual Machine Manager** and log in using your Orchestrator username and password.

You see a basic Web view, with no operations or functions.

You created an empty Web view from a template, and inspected it in a browser.

### What to do next

Export the empty Web view to a working folder.

## Export the Virtual Machine Manager Web View to a Working Folder

You edit the HTML files and Web view components of the Virtual Machine Manager Web view on your local system, using Web development tools.

### Prerequisites

Verify that you imported the default Web view template to Orchestrator and used it to create a new Web view.

### Procedure

1 Create a folder on your local system in which to develop the Virtual Machine Manager Web view.

For example, `MyWebView`.

2 (Optional) If Virtual Machine Manager is running, in the **Web Views** view of the Orchestrator client, right-click **Virtual Machine Manager** and select **Unpublish**.

3 Right-click Virtual Machine Manager and select **Export to directory**.

4 Navigate to your working folder and click **Export**.

You exported the empty Web view to a working directory.

### What to do next

Provide the Web view components with unique names.

## Contents of the Default Web View Template

After you export the Web view to your working folder, you can examine the contents of the default Web view template.

### Table 8-1. Files of the `default_webview` Web view template

| File | Description |
|---|---|
| `default.html` | Home page of the default Web view template. |
| `components\layout\Access.jwc` | JWC component that provides a login function. |
| `components\layout\Access.html` | HTML template that defines how the login component appears in the browser. |
| `components\layout\Menu.jwc` | JWC component that specifies menu tabs for the default Web view. |
| `components\layout\Menu.html` | HTML template that you edit to add menu tabs to the default Web view. |
| `components\layout\MyBorder.jwc` | JWC component that defines the borders of the default Web view, sets its name, loads the logos and stylesheets, and so on. |
| `components\layout\MyBorder.html` | HTML template that defines how the borders appear in the browser. |
| `css\border.css` | CSS style sheet that renders the border component. |
| `css\custom.css` | Customizes other stylesheets, such as `border.css` or `login.css`. Every page of the default Web view imports the `custom.css` file. The `custom.css` file overrides the other style sheets in the template.<br>**Note** The `custom.css` style sheet is the only style sheet you can edit. If you edit the other style sheets directly rather than editing `custom.css`, your edits are overwritten every time you upgrade to a new version of Orchestrator. |
| `css\login.css` | CSS style sheet that renders the login component. |
| `images\` | Contains image files for the banners, logos, and icons that appear in the default Web view. |

# Provide Unique Component Names

To avoid conflicts if you run multiple Web views in the same Orchestrator server, you must make sure that all Web view components have unique names.

You can ensure that Web view components have unique names either by moving the components into a unique subfolder in the `\components` folder, or by changing the names of the Web view components. Because the Virtual Machine Manager example uses only one component from the template, the easiest solution is to rename the component.

### Prerequisites

Export the contents of the Web view template to a working folder.

### Procedure

1 Navigate to the `\components\layout` folder in the working folder.

   For example, `\MyWebView\components\layout`.

2 Rename the `MyBorder.html` file to `VMMBorder.html`.

3 Rename the `MyBorder.jwc` file to `VMMBorder.jwc`.

### What to do next

Configure the server for Web view development.

# Configure the Server for Web View Development

During the Web view development process, you can configure the Orchestrator server to publish the Web view from a working folder rather than from the Orchestrator server.

When the server runs in development mode, you can preview the Web view as you develop it, without having to import it to the Orchestrator server to view it. You set the Orchestrator server to Web view development mode in the Orchestrator configuration interface.

**Note** Because Orchestrator publishes Web views from the working folder, you cannot access Web views that you have not exported to the working folder when the server is in development mode.

### Prerequisites

To enable Web view development mode, your working folder must be on the same machine as the Orchestrator server.

### Procedure

1 Log in to the Orchestrator configuration interface by using the your Orchestrator configuration username and password.

   For example, go to https://*orchestrator_server_DNS_name_or_IP_address*:8283 or https://localhost:8283 in a Web browser.

2 On the **General** tab click **Advanced Configuration**.

3 Select the **Enable Web view development mode** check box.

4 Type the path to the root of your working folder in the text box.

   Make sure you provide the path to the root of the working folder. Do not include the name of the folder that contains the Web view in the path.

   For example, if you are working on a Web view in the folder `C:\Documents and Settings\username\Desktop\MyWebView\`, type `C:\Documents and Settings\username\Desktop\` as the path.

5 Click **Apply changes**.

6 On the **Startup Options** tab, click **Restart Service** to restart the Orchestrator server in Web view development mode.

7 After the Orchestrator server has restarted, start the Orchestrator client and log in.

8 Click **Web Views**.

9 Verify that your Web view's URL folder value matches the name of your working directory.

   For example, if you created the working folder `C:\Documents and Settings\username\Desktop\MyWebView\`, set the URL folder to `MyWebView`.

   a If the Web view is running, right-click the Web view and select **Unpublish**.

   b Right-click the Web view and select **Edit**.

   c On the **General** tab of the Web view editor, type the name of the working folder in the **URL folder** text box, and click **Save and Close** to close the Web view editor.

10 Right-click the Web view and select **Publish**.

You set the Orchestrator server to Web view development mode, in which you can preview a Web view from your working folder while you develop it.

# Edit the Virtual Machine Manager Web View Home Page

You create the home page of the Virtual Machine Manager Web view in the `default.html` file. The `default.html` file must be at the root of the working directory.

### Prerequisites

- Import the default Web view template to Orchestrator to create the Virtual Machine Manager Web view
- Export the contents of the Virtual Machine Manager Web view to a working directory
- Configure Orchestrator in Web view development mode

### Procedure

1 Go to the root of your working directory.

2. Open the `default.html` file in an HTML editor.

   The `default.html` page uses the `MyBorder` component to render itself. It contains little code, as the following code sample shows.

   ```
   <vco jwcid="@layout/MyBorder" section="literal: home" title="Home">
       <!--  Content of the homepage -->
           <h2 style="margin-left: 16px; margin-top: 0px; padding-top:18px;">
       Welcome to Default Webview Template
     </h2>
           <p style="margin-left: 16px;">
       This webview is a base for your own webview development.
     </p>
   </vco>
   ```

   The `vco` tag initializes a Tapestry component by setting the `jwcid` attribute to point to the `MyBorder` component, which you renamed to **`VMMBorder`**.

3. Change the `jwcid` attribute to refer to **`VMMBorder`** instead of `MyBorder`.

   ```
   <vco jwcid="@layout/VMMBorder" section="literal: home" title="Virtual Machine Manager">
   ```

4. Change the `title` attribute to **`Virtual Machine Manager`**.

   ```
   <vco jwcid="@layout/VMMBorder" section="literal: home" title="Virtual Machine Manager">
   ```

5. Delete the default `<h2>` heading from the `default.html` file.

   This heading is unnecessary for the simple Web view that this example demonstrates. Delete the following code line.

   ```
   <h2 style="margin-left: 16px; margin-top: 0px; padding-top:18px;">
   Welcome to Default Webview Template
   </h2>
   ```

6. Change the paragraph text from `This webview is a base for your own webview development` to the following text:

   ```
   <p style="margin-left: 16px; margin-top: 0px; padding-top:18px;">
   Click one of the virtual machines in your inventory to display its
   information.</p>
   ```

7. Go to `https://`*`orchestrator_server`*`:8280/vco/vmo/` in a browser to check the appearance of the Web view.

8 Make any necessary adjustments to the HTML code to improve the appearance of the Web view.

For example, change the spacing of the text by adjusting the margins of the <p> tag and add a hard return line under the text.

```
<p style="margin-left: 16px; margin-top: 5px; margin-bottom: 5px;">
Click one of the virtual machines in your inventory to display its information.
</p>
<hr />
```

### What to do next

Add a function to the `default.html` page by referring to a Web view component.

## Add a vco:ListPane Component to the Web View Home Page

You can add a `vco:ListPane` component to the Virtual Machine Manager Web view home page to display a list of vCenter Server objects.

### Prerequisites

- Import the default Web view template into the Orchestrator server.
- Export the contents of the default Web view template to a working directory.
- Configure the Orchestrator server for Web view development.
- Open the `default.html` file in an HTML editor.

### Procedure

- Add a `vco:ListPane Component` to the `default.html` file to add a list of virtual machines in the page.

  You add a `vco:ListPane Component` by adding the following <p> tag after the instruction to click a virtual machine.

```
<vco jwcid="@layout/VMMBorder" section="literal: home" title="Virtual Machine Manager">
   <p style="margin-left: 16px; margin-top: 5px; margin-bottom: 5px;">
   Click one of the virtual machines in your inventory to display its information.
   </p>
   <hr />

   <p jwcid="@vco:ListPane"
                    action="getVirtualMachineList"
                    actionParameters="attribute:vmFolder"
                    detailUrl="./panel.html"/>

</vco>
```

  The <p> tag instantiates the `vco:ListPane` component with the following properties:

Table 8-2. Properties of the `vco:ListPane` component

| Property | Description |
|---|---|
| `jwcid= "@vco:ListPane"` | Refers to the `vco:ListPane` component, to add a list of virtual machines from the server inventory to the Web view. |
| `action= "getVirtualMachineList"` | Links to a Web view attribute that runs an action in the Orchestrator server to retrieve a list of virtual machines. |
| `actionParameters= "attribute:vmFolder"` | Passes parameters to the `getVirtualMachineList` action. The `actionParameters` property passes the `vmFolder` Web view attribute to the action, to retrieve a list of the virtual machines from a virtual machine folder. |
| `detailUrl= "./panel.html"` | The path to an HTML page in which to display information about each virtual machine in the list. |

You added a component to the Virtual Machine Manager Web view that obtains a list of virtual machines from a given folder in vCenter Server.

**Note** You create the Web view attributes that the `vco:ListPane` component requires in the Orchestrator client.

### What to do next

Define the Web view attributes that the `vco:ListPane` component requires.

## Define the Web View Attributes for the vco:ListPane Component

The `vco:ListPane` component refers to Web view attributes that provide an action to obtain an array of virtual machines from the server and the virtual machine folder from which to obtain them.

### Prerequisites

You added the `vco:ListPane` component to the `default.html` Web view file.

### Procedure

1 From the drop-down menu in the Orchestrator client, select **Administer**.

2 In the **Web Views** view, right-click the **Virtual Machine Manager** Web view and select **Edit**.

3 Click the **Attributes** tab in the Web view editor.

4 Right-click in the **Attributes** tab and select **Add attribute**.

5 Click the attribute name and type **`getVirtualMachineList`**.

6 Click the attribute **Type** link and select **Action** from the list.

7 Click the attribute **Value** link and search for and select the **getAllVirtualMachinesByFolder** action.

The `getAllVirtualMachinesByFolder` action returns an array of `VC:VirtualMachine` objects.

8 Right-click in the **Attributes** tab and select **Add attribute** to create the following Web view attribute:

- Name: **`vmFolder`**
- Type: `VC:VmFolder`
- Value: a virtual machine folder that you select from the vCenter Server inventory

**Note** If you install the Virtual Machine Manager Web view from the Orchestrator samples bundle, you must edit the `vmFolder` attribute to point to a virtual machine folder in your vCenter Server before you publish the Web view.

9 Click **Save and Close** to exit the Web view editor.

10 Open the Virtual Machine Manager Web view in a browser at `https://`*`orchestrator_server`*`:8280/vco/vmo/`

The Virtual Machine Manager Web view displays a list of virtual machines.

The Virtual Machine Manager Web view uses the `getVirtualMachineList` Web view attribute to obtain the list of virtual machines from the virtual machine folder that the `vmFolder` attribute defines. However, clicking on a virtual machine in the list returns an error, as you did not define how to obtain and display the virtual machine information.

### What to do next

Create a Web view component to obtain and display information about a virtual machine when you click it in the list.

## Create a Web View Component to Display Virtual Machine Information

The `vco:ListPane` component lists virtual machines in the left side of the Web view. You can create a Web view component to show information about each virtual machine in a Web view panel on the right side.

The Web view panel that displays the virtual machine information requires a Web view component that obtains information from the objects that the `vco:ListPane` component lists and displays the information on the right. The Web view component that obtains object properties implements the `WebObjectComponent` class.

### Prerequisites

Make sure that you have added a `vco:ListPane` component to the `default.html` file and defined the Web view attributes that the component requires.

### Procedure

1 Create a file called `panel.html` and save it at the root of your working folder.

2 Add a title to the `panel.html` file.

```
<h3>Virtual Machine Information</h3>
```

3 Create a Web view component specification file called `DisplayVmInfo.jwc` in the `\components` folder.

4 Add references to the Tapestry DTD and the `WebObjectComponent` Java class to the `DisplayVmInfo.jwc` component specification file.

You refer to the DTD in the `DOCTYPE` metatag and use `<component-specification>` tags to refer to the `WebObjectComponent` Java class.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE component-specification PUBLIC
"-//Apache Software Foundation//Tapestry Specification 4.0//EN"
"http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">
<component-specification class="ch.dunes.web.webview.components.WebObjectComponent">
</component-specification>
```

5 Create a Web view component template called `DisplayVmInfo.html` in the `/components` folder.

The `DisplayVmInfo.html` file defines how to present the information that the component obtains.

6 Add a table to the `DisplayVmInfo.html` file to contain information about the virtual machines that are in the list.

```
<table width="200" border="1">
  <tr>
    <td>Virtual machine name</td>
    <td></td>
  </tr>
  <tr>
    <td>Virtual machine attributes</td>
    <td></td>
  </tr>
  <tr>
    <td>Object ID</td>
    <td></td>
  </tr>
</table>
```

7 Add references to the standard Tapestry `Insert` component and OGNL statements to obtain properties from the array of `VC:VirtualMachine` objects that the `vco:ListPane` component obtains.

```
<table width="200" border="1">
  <tr>
    <td>Virtual machine name</td>
    <td><vco jwcid="@Insert" value="ognl:get('name')"/></td>
  </tr>
  <tr>
```

```
    <td>Object ID</td>
    <td><vco jwcid="@Insert" value="ognl:get('id')"/></td>
  </tr>
</table>
```

The `value` attributes of the `Insert` component use the `WebObjectComponent.get()` methods in OGNL statements to obtain the following properties from the `VC:VirtualMachine` objects:

- The `name` property to display the virtual machine name
- The `id` property to display the Orchestrator ID of the virtual machine

8. Add a reference to the `DisplayVmInfo` component to the `panel.html` file.

```
<h3>Virtual Machine Information</h3>
<vco jwcid="@DisplayVmInfo" urlParameter="itemId" />
```

9. Open the Virtual Machine Manager Web view in a browser at `https://`*`orchestrator_server`*`:8280/vco/vmo/` and click one of the virtual machines in the list on the left.

   Information about the virtual machine you clicked appears on the right.

You created a Web view page that obtains a list of virtual machines from vCenter Server and displays information about each virtual machine in the list.

### What to do next

Create a tab in the Virtual Machine Manager Web view to run workflows on objects in from the vSphere inventory.

## Create a Web View Tab by Using the Menu Component

The default Web view template provides a `Menu` component that you can use to create navigation tabs in a Web view.

The `VMMBorder` component that the default Web view template uses to render its layout includes a reference to the `Menu` component. Any changes you make to the `Menu` component appear in the Web view.

### Prerequisites

- Import the default Web view template to Orchestrator to create the Virtual Machine Manager Web view
- Export the contents of the Virtual Machine Manager Web view to a working directory
- Configure Orchestrator in Web view development mode

### Procedure

1. Create an HTML file called `runWorkflows.html` and save it at the root of your working folder.

   The `runWorkflows.html` file defines the body of the new tab.

2. Add a reference to the MyBorder component, a title, and some text to the runWorkflows.html file.

   ```
   <vco jwcid="@layout/VMMBorder" section="literal: home" title="Run Workflows">
      <p style="margin-left: 16px; margin-top: 5px; margin-bottom: 5px;">Click a workflow to run
   it.</p>
   </vco>
   ```

3. Rename the /component/Menu.html Web view template file to /component/VMMMenu.html.

   If you run more than one Web view in the same Orchestrator server, all components must have a unique name.

4. Open the /component/VMMMenu.html Web view template file in an HTML editor.

   The Menu.html file contains an unordered list with one <li> entry that links to the home page of the Web view.

   ```
   <ul id="sectionMenu" class="menu">
     <li jwcid="@Any" class="ognl:page.isCurrentPage(webview.urlFolder + '/default')? 'selected':''">
     <a href="./">Home</a></li>
    </ul>
   ```

5. Copy and paste the <li> entry to create a second menu tab.

   You create menu tabs in the Web view by adding <li> entries to the unordered list.

6. Modify the new <li> entry to point to runWorkflows.html.

   a. Change the ognl:page.isCurrentPage OGNL statement to add the runWorkflows suffix to the URL.

   b. Change the <a href> link to point to ./runWorkflows.html.

   c. Change the <a href> text to Workflows.

   ```
   <li jwcid="@Any" class="ognl:page.isCurrentPage(webview.urlFolder + '/runWorkflows')?
   'selected':''">
   <a href="./runWorkflows.html">Workflows</a></li>
   ```

7. Open the Virtual Machine Manager Web view in a browser at https://*orchestrator_server*:8280/vco/vmo/.

   The **Workflows** tab appears in the Web view. If you click the tab, you see the contents of the runWorkflows.html file.

You added a tab to the Web view.

### What to do next

Use the vco:WorkflowLink component to run workflows from the **Workflows** tab.

# Add Links to Run Workflows from a Web View by Using the vco:WorkflowLink Component

You add links to run workflows from a Web view by using the `vco:WorkflowLink` component. You define the workflows to run by setting Web view attributes.

### Prerequisites

- Create a Web view tab in the Virtual Machine Manager Web view by modifying the `Menu.html` component template.
- Create the `runWorkflows.html` file to define the contents of the tab.

### Procedure

1 Open the `runWorkflows.html` file in an HTML editor.

2 Add a reference to the `vco:WorkflowLink` component to the `runWorkflows.html` file.

```
<vco jwcid="@layout/VMMBorder" section="literal: home" title="Run Workflows">
   <p style="margin-left: 16px; margin-top: 5px; margin-bottom: 5px;">
   Click a workflow to run it.</p>

   <ul>
    <li>
    <a jwcid="@vco:WorkflowLink" workflow="createVM" isDialog="true">
    Create simple virtual machine</a>
    </li>
   </ul>

</vco>
```

When you set the `isDialog` property to `true`, a dialog box appears in which users provide input parameters to run the workflow.

The `workflow` property refers to a Web view attribute called `createVM` that you create in the Orchestrator client.

3 In the **Web Views** view in the Orchestrator client, right-click the **Virtual Machine Manager** Web view and select **Edit**.

4 Right-click in the **Attributes** tab in the Web view editor and select **Add attribute**.

5 Click the attribute name and type **`createVM`**.

6 Click the attribute **Type** link and select **Workflow** from the list.

7 Click the attribute **Value** link and search for and select the **Create simple virtual machine** workflow.

8 Click **Save and Close** to exit the Web view editor.

9 Open the Virtual Machine Manager Web view in a browser at https://*orchestrator_server*:8280/vco/vmo/.

10. Click the **Create simple virtual machine** link in the **Workflows** tab.

    A Web form opens in the browser to allow users to enter parameters to create a virtual machine.

11. (Optional) Add more links to start workflows by adding more `vco:WorkflowLink` references to `runWorkflows.html`.

    For example, add the following `vco:WorkflowLink` references:

    ```
    <li><a jwcid="@vco:WorkflowLink" workflow="cloneVM" isDialog="true">
    Clone a virtual machine
    </a></li>

    <li><a jwcid="@vco:WorkflowLink" workflow="snapVM" isDialog="true">
    Take a snapshot of all virtual machines in a resource pool
    </a></li>

    <li><a jwcid="@vco:WorkflowLink" workflow="removeSnaps" isDialog="true">
    Remove virtual machine snapshots of a given size
    </a></li>

    <li><a jwcid="@vco:WorkflowLink" workflow="thickToThin" isDialog="true">
    Convert a virtual disk from thick to thin provisioning
    </a></li>

    <li><a jwcid="@vco:WorkflowLink" workflow="deleteVM" isDialog="true">
    Delete a virtual machine
    </a></li>
    ```

    **Note** Make sure that you create the Web view attribute in the Orchestrator client for each `vco:WorkflowLink` reference.

You added links to the Virtual Machine Manager Web view that run workflows on virtual machines in the vSphere inventory.

### What to do next

Customize the appearance of the Web view.

## Customize the Web View Interface

You can customize the appearance of the Web view by adjusting the `custom.css` style sheet and changing the images in the `\images` folder.

### Prerequisites

You created the Virtual Machine Manager Web view.

### Procedure

1. (Optional) Change the Web view logo by overwriting the `\images\header_logo.jpg` file, for example, with a JPEG of your company logo.

2 Open the `\css\custom.css` style sheet file in a text editor.

3 Adjust the `width` and `height` values of `siteTitle` to fit the proportions of your company logo file.

4 Refresh the page in the browser.

5 (Optional) Customize any part of the Web view by modifying the `custom.css` file.

   You can edit the `custom.css` to override the `VMMBorder.html` component template file and the `border.css` style sheet to modify the overall layout of the Web view.

   **Note** The `custom.css` style sheet is the only layout file that you should modify. If you modify other style sheets or component template files, your changes are lost if you upgrade Orchestrator.

You customized the appearance of the Web view by adjusting the `custom.css` style sheet and by changing the logo.

### What to do next

Disable Web view development mode, import the Web view files to the server, and publish the Web view.

## Publish the Virtual Machine Manager Web View

After you finish developing the Virtual Machine Manager, you must disable Web view development mode, import the Web view from your working directory to the Orchestrator server, and publish the Web view.

### Prerequisites

You finished developing the Virtual Machine Manager Web view.

### Procedure

1 Log in to the Orchestrator configuration interface by using the your Orchestrator configuration username and password.

   For example, go to https://*orchestrator_server_DNS_name_or_IP_address*:8283 or https://localhost:8283 in a Web browser.

2 On the **General** tab click **Advanced Configuration**.

3 Deselect the **Enable Web view development mode** check box.

4 Click **Apply changes**.

5 On the **Startup Options** tab click **Restart Service** to restart the Orchestrator server in normal mode.

6 After the Orchestrator server restarts, right-click **Virtual Machine Manager** in the **Web Views** view of the Orchestrator client, and select **Edit**.

7 Click **Import from directory** in the **Elements** tab in the Web view editor.

8 Select your working folder and click **Import** to import the updated files from your working directory to the server.

9 Click **Save and Close** to exit the Web view editor.

10 Right-click the **Virtual Machine Manager** Web view and select **Publish**.

You imported the Virtual Machine Manager Web view from your working directory to the server and published it.

**Note** If you install the Virtual Machine Manager Web view from the Orchestrator samples bundle, you must edit the `vmFolder` attribute to point to a virtual machine folder in your vCenter Server before you publish the Web view.

### What to do next

You can open the Virtual Machine Manager Web view in a browser and use it to examine virtual machines and run workflows.