# Table of Contents

**Preface** ............................................................................................................................ 9

**Part 1 Configuring and Customizing Web Publisher** ......................................................... 11

**Chapter 1 Quick Start to Customizing Web Publisher** .................................................. 13
How to Approach a Customization ................................................................................. 13
Documentation Resources for Customizations ......................................................... 13

**Chapter 2 Introduction to Web Publisher and Documentum** ........................................ 15
Introduction to Web Publisher ...................................................................................... 15
What Does the Documentum System Do? .................................................................... 16
What is a Docbase? ......................................................................................................... 16
What is a Checkout Directory? ..................................................................................... 17
What is a Lifecycle? ....................................................................................................... 17
What is a Workflow? ...................................................................................................... 18
What is an Inbox? ........................................................................................................... 18
What are Links, Shortcuts, and Replicas? .................................................................... 18
What is a Permission Set? ............................................................................................ 19
Lifecycle States Specific to Web Publisher .............................................................. 19
How Web Publisher Creates Web Pages ..................................................................... 20
Applications Used to Create Web Pages ................................................................. 20
How Web Publisher Publishes Content to the Web .................................................. 21
How Files are Named in Web Publisher ..................................................................... 22
File Names on the Web ............................................................................................... 23
Using Web Publisher and eWebEditPro ..................................................................... 23
Using Web Publisher and iMarkup .............................................................................. 23

**Chapter 3 Technical Overview of WDK Applications** .................................................. 25
Introduction ..................................................................................................................... 25
Before You Customize .................................................................................................... 26
WDK Application Directory Structures ...................................................................... 26
The Custom Layer ......................................................................................................... 29
How to Determine a Component from the UI ............................................................ 30
Listing the Application’s Components ....................................................................... 30
Determining Which Parts of the Component to Customize ......................................... 31

**Chapter 4 Web Publisher Application Configuration** .................................................. 33
Web Publisher Environment Settings ........................................................................... 33
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 5</td>
<td>Web Publisher Files</td>
<td>37</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Web Publisher Context Views</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Using wpview</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Configuring Actions Using wpcontext</td>
<td>40</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Role-based Configuration</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Default Roles</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Configuring Role-based Actions</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Configuring Role-based Components</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Adding Roles</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Other Types of Role Configuration</td>
<td>45</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Workflow Dynamic Performers</td>
<td>47</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Mapping CIS Properties to Docbase Attributes</td>
<td>51</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Configuring the Content and HTML Editors</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Configuring the Look and Feel of the Web Publisher Editor</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Other Web Publisher Editor Configurations</td>
<td>54</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Configuring the Web Publisher Editor</td>
<td>57</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Customizing Web View</td>
<td>59</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>Web Publisher Tracing</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Web Publisher Tracing Flags</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>WCM Tracing Flags</td>
<td>65</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>Customizing Web Publisher Editor Controls</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>Custom Control Overview</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>Custom Control Requirements</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Swing Compatibility</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Control Construction</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>EditWidget APIs</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Custom Control Wrapper</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>Using a Custom Control in a Rule</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>Writing and Packaging the Control</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>Using Supporting Classes</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>Packaging the Control</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>Custom Control Messages</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>Custom Control Example for Multi-Line Text Area</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>TestWidget Class File</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>ANT Configuration File</td>
<td>80</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>Customizing Dynamic Performers</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Performer and Alias Initialization</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>IAliasPerformerInit</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>IDynamicPerformerInit</td>
<td>84</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>Requirement for Extending a Web Publisher Service</td>
<td>91</td>
</tr>
<tr>
<td>Part 2</td>
<td>Web Publisher Reference</td>
<td>93</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>Web Publisher Controls</td>
<td>95</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>Web Publisher Actions</td>
<td>105</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>Web Publisher Components</td>
<td>155</td>
</tr>
</tbody>
</table>
List of Figures

Figure 3–1. WDK Applications Directory and File Structure................................................ 26
Figure 5–1. Web Publisher Editor Files .............................................................................. 37
# List of Tables

<table>
<thead>
<tr>
<th>Table 2–1.</th>
<th>In Web Publisher</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3–1.</td>
<td>WDK Application Directory and File Descriptions</td>
<td>27</td>
</tr>
<tr>
<td>Table 3–2.</td>
<td>WDK Application Directory and File Descriptions (cont’d)</td>
<td>28</td>
</tr>
<tr>
<td>Table 3–3.</td>
<td>Sub-Application Directory and File Descriptions</td>
<td>29</td>
</tr>
<tr>
<td>Table 3–4.</td>
<td>Structure of the Custom Layer</td>
<td>29</td>
</tr>
<tr>
<td>Table 4–1.</td>
<td>Cookies in Web Publisher</td>
<td>34</td>
</tr>
<tr>
<td>Table 10–1.</td>
<td>Rules Editor Settings</td>
<td>54</td>
</tr>
<tr>
<td>Table 11–1.</td>
<td>Web Publisher Editor Settings</td>
<td>57</td>
</tr>
<tr>
<td>Table 19–1.</td>
<td>Web Publisher Extended Components</td>
<td>156</td>
</tr>
</tbody>
</table>
The purpose of this document is to provide information about customizing Web Publisher.

**Intended Audience**

This manual is intended for two audiences:
- Java developers who are developing custom JSP-based Web applications that incorporate Documentum functionality
- Web page designers who are configuring WDK client applications or custom application designed by your enterprise

To customize WDK-based applications, you must be familiar with the following technologies:
- JSP 1.1 including tag libraries
- Cascading style sheets (CSS)
- HTML, particularly forms, tables, and framesets
- JavaScript, including client events and event handling, frame referencing, and form action methods
- XML
- Java 1.3.1 or 1.4
- Servlet 2.3

**Revision History**

The following changes have been made to this document.
## Revision History

<table>
<thead>
<tr>
<th>Revision Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 2004</td>
<td>Initial publication.</td>
</tr>
</tbody>
</table>
Part 1

Configuring and Customizing Web Publisher
Chapter 1

Quick Start to Customizing Web Publisher

Web Publisher is a WDK application, in that it contains a number of application layers, including WDK and webcomponent layers (part of Web Development Kit), a Webtop layer, a DAM (Digital Asset Manager) layer, and a Web Publisher layer, which contains components, controls, and actions used by Web Publisher only.

This guide contains the following information:

- General information on customizing WDK applications
- A nontechnical overview of Web Publisher and Documentum
- A technical overview of WDK applications
- Information specific to configuring and customizing the Web Publisher application layer

How to Approach a Customization

When you approach a customization task, there are several important pieces of information that you need to know in order to determine what type of customization you need to perform.

- Which components/controls need to be modified. See How to Determine a Component from the UI, page 30.
- Which application layer should be modified. See How to Determine a Component from the UI, page 30 and Listing the Application's Components, page 30.
- Whether you accomplish the customization by configuring the component or changing the code. See Determining Which Parts of the Component to Customize, page 31.

Documentation Resources for Customizations

If you want to acquire a basic familiarity with Web Publisher or Documentum, read Chapter 2, Introduction to Web Publisher and Documentum.
This manual contains customization information that is specific to the Web Publisher application layer, but many customizations are more generic and might be handled by extending a component in an underlying application layer. To learn basic techniques of customizing a WDK application, see the Web Development Kit Development Guide and the Webtop Development Guide.

There is also a tutorial guide for WDK and Webtop. If you have access to Webtop, you may wish to work through one or more tutorials to get a basic understanding of how WDK applications are customized.

For more code samples, examples, troubleshooting information, and peer assistance, see the Documentum Customernet website (http://customernet.documentum.com). In particular, it includes:

- Support Notes
- Knowledge base
- Support Forums
- Sample Code
- Tips
- Component Exchange
Chapter 2

Introduction to Web Publisher and Documentum

This chapter provides a nontechnical overview for readers who are not familiar with the Web Publisher application or some of the basic concepts in Documentum.

Introduction to Web Publisher

Web Publisher is a Web browser-based application that simplifies and automates the creation, review, and publication of Web sites.

Table 2-1. In Web Publisher...

<table>
<thead>
<tr>
<th>Authors...</th>
<th>Use editing applications to create and modify Web-page content — without having to know how to construct or design Web pages.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>See their content automatically published to the Web — even in multiple formats and languages.</td>
</tr>
<tr>
<td>Developers...</td>
<td>Maintain the content templates and site structures that help make such automation and streamlining possible.</td>
</tr>
<tr>
<td>Administrators...</td>
<td>Maintain the Web Publisher system settings that allow for such ease of publication.</td>
</tr>
</tbody>
</table>

Web Publisher works within Documentum 5, using Documentum Content Server to store and process content and using Documentum Site Caching Services (SCS) to publish to the Web. SCS is the publishing component of Web Publisher. All Web Publisher Web pages are published by SCS. SCS has two pieces: one sits on the Content Server and the other sits on the target Web server. SCS sends documents from a Documentum Docbase to a designated location on a Web server.

Web Publisher also works with Documentum Content Rendition Services (CRS), though this is not required to run Web Publisher. CRS is used to create PDF or HTML renditions...
from Microsoft Documents. It is only used when content must be transformed in a Web-ready format. Content originating in XML and HTML does not require CRS. XML content is rendered through our internal transformation engine to create the required web-ready formats. Any number of XSL stylesheets may be associated to a single XML file.

Web Publisher works with numerous other Documentum applications. See the Web Publisher Release Notes for details.

What Does the Documentum System Do?

The Documentum system lets you access, use, reuse, and distribute information quickly and easily. Using the Documentum system, you can store many different types of information — text, graphics, scanned images, sound, and digital movies — in one or more company repositories called Docbases.

With the Documentum system, you can access documents across different departments, sites, and computer platforms; distribute documents electronically; automate business procedures through lifecycles and workflows; restrict access to documents according to permission sets; and assign document version numbers to help you keep track of revisions. These are just a few of the ways that you can work more quickly and easily with Documentum.

What is a Docbase?

A Docbase is a secure repository for storing files. Each Docbase provides security, services, and tools for sharing content among different users. Each Docbase provides automated processes that control the lifecycles of files and the delivery of files to the right user or location at the right time. Automated processes allow users to contribute to file processing regardless of their technical skill levels.

Your organization might use several Docbases. When you log in, you choose which Docbases to log into. Once logged in, you can switch Docbases without having to exit the application.

Each Docbase stores two kinds of information for each file:
- Content, which is the text, graphics, sound, or video that makes up the document.
- Properties, which are descriptive characteristics about the file, such as creation date, author, version number, and other information. Documentum automatically enters the values for some properties when you create an object, while for other properties you enter the values yourself. Property values can only be set and edited by the object’s creator or a user with high enough security settings.

The highest level of organization in a Docbase is a node. You access different functions through the different nodes.

The highest level of file storage in a Docbase is a cabinet. Each cabinet contains a folder structure for storing that cabinet’s files. Files can also be stored directly in the cabinet. If you have the appropriate user privilege level, you can create cabinets.
In each Docbase you use, you have a home cabinet with your name on it. Only you can see or access your home cabinet. Your home cabinet is where you store personal documents, folders and other objects.

**Note:** Do not change the name of your home cabinet unless absolutely necessary. If you change the name of your home cabinet, inform your system administrator so that the administrator can specify the new name for the cabinet. This can affect customizations.

Several Docbases can be grouped into a federation. A federation is a way of configuring a group of Docbases to simplify administration.

In each Docbase, you have an Inbox. Your Inbox displays the tasks and notifications sent to you. Tasks and notifications can include attached files. In a Docbase federation, you have one Inbox for the whole federation.

When you want to modify a file, you check out the file from the Docbase. This locks the file so that only you can modify it. Other users can view the file, but they cannot make changes to it. When you complete your work, you check in the file. Your changes are saved to the Docbase and the file is no longer locked. When another user accesses the file, the file contains the changes you made.

When you check in a document, the system gives it a version number and a version label. The system increments version numbers automatically, starting with 1.0 and adding one to the number after the decimal point for each revision, unless you specify otherwise. When checking in a file, you can choose not to increment, which keeps the same version number and overwrites the existing version.

Docbases are object-oriented. Every item in a Docbase — every cabinet, folder, file, workflow, etc. — is stored as a Docbase object. Every object belongs to a defined object type. Documents, for example, typically have an object type of *dm_document*. The object type determines what types of properties are associated with the object. All object types are organized into a hierarchy — with some object types being subtypes of others. Your organization can add custom object types.

### What is a Checkout Directory?

The checkout directory is the location on your computer for storing Docbase files. When you check out a Docbase file, the file is copied to your checkout directory. Once you have the file checked out, you can open and close it directly from the checkout directory, whether or not you are connected to the Docbase. When you are ready to save your changes to the Docbase, you check the file back in.

### What is a Lifecycle?

Each document in the Docbase has a lifecycle that consists of the different states the document goes through between creation and expiration. When you create a document, the system automatically assigns the document the lifecycle designated by the document’s template and puts the document into the first state in that lifecycle.
A document advances through its different lifecycle states through either manual or automatic promotions.

When Documentum applies a lifecycle to a document, an alias set is also associated with the document and the lifecycle. The alias set defines which Docbase users can review, promote, or demote the document. Alias sets can also specify permission sets that apply to the document and the Docbases where the document can be located.

Typically, a lifecycle is incorporated into a workflow, and you are alerted to your role in a document’s lifecycle when a workflow task appears in your Inbox.

**What is a Workflow?**

A workflow is a process that electronically passes files and instructions from user to user. For example, an employee might initiate a travel expense report; another employee might review it and return it for revision; and a third employee might approve it, after which it enters an archive. A workflow automates the process, ensuring that the right documents go to the right people for the right tasks in the right order.

To start a workflow, you choose the workflow template that includes the sequence of tasks you want performed. Some workflow templates specify the users who receive the tasks; others allow you to select the users. When you start a workflow, you can attach files you want users to view or reference. A file can be attached to only one workflow at a time.

Workflow can include automatic tasks, which are performed by the system. For example, an automatic task might promote a file to a new lifecycle state once it has been approved by a reviewer.

**What is an Inbox?**

Your Inbox contains the tasks and notifications sent to you. Tasks are electronic assignments sent to you as part of a workflow. When you receive a task, you choose whether to accept and perform it, or reject it. When you complete a task, you forward it. The workflow notifies the next user in sequence. Tasks can include attached files.

Notifications are messages letting you know when a specific action has occurred on a file. You choose to be notified about events for a file by selecting the appropriate notification option in the file’s properties. For example, if you want to be notified every time a certain document is checked in, you select “checkin” as an event notification.

**What are Links, Shortcuts, and Replicas?**

You can link a Docbase file to multiple locations in a Docbase, meaning you can access the file from any of those locations. The file is considered to exist in each of those
locations. If you check out the file in one location, it is checked out in all locations. When you make a change to the file in one location, it is changed in all locations.

You can create shortcuts to Docbase items on your computer, on your network, or in emails.

Your administrator can create read-only replicas of files, which are copies in remote Docbases. Replication enables you to use remote Docbases to access information from a local Docbase. For example, if you have two sales offices, one in California and one in Germany, that share the same information, you want both offices to be able to quickly access the information they need. For each office to access information quickly the information source must be local to each. Replicas are periodically copied from the source Docbase to a target Docbase or Docbases.

What is a Permission Set?

Your access to the folders, documents and other items in a Docbase is determined by the permission sets that are assigned to those folders, documents and other items. Each item in the Docbase has an associated permission set, determining who can access the item and what actions the each user with access can perform.

A permission set lists the users and user groups who have access and lists the level of access each user and user group has.

A user or group listed in a permission set is assigned one of seven access levels. Each access level includes all the permissions of the preceding levels:

- None: No access to the item.
- Browse: Users can view the item's properties (but not its content).
- Read: Users can view the item's content.
- Relate: Users can annotate the item.
- Version: Users can modify and check in new versions of the item.
- Write: Users can modify and check in the item as the same version.
- Delete: Users can delete items.

Lifecycle States Specific to Web Publisher

When you create a document, Web Publisher automatically assigns the document the lifecycle designated by the document's template and puts the document into the first state in that lifecycle. A document advances through its lifecycle states through either manual or automatic promotions. Web Publisher's default lifecycle has the following states:

- **Start**
  When content is newly created or newly versioned, Web Publisher places it in the Start state, for initialization purposes, and then immediately promotes it to the WIP state.

- **WIP (Work In Progress)**
Content in draft or review.

- **Staging**
  Content that is complete and ready for testing on a staging Web site. By default, Web Publisher does not allow you to modify a file’s content, location or properties if the file has advanced to the Staging state or beyond.

- **Approved**
  Content that is approved for the active Web site but has not yet reached its publication date (i.e., effective date).

- **Active**
  Content that is on the active Web site.

### How Web Publisher Creates Web Pages

Web Publisher separates the design of Web pages and Web sites from the creation of their content. The separation allows your organization to share content among different sites, and share page designs among different content.

Developers design templates and other files that determine the layout of Web pages and the structure of Web sites. Developers configure Web Publisher to determine where new content is placed on a Web site and when and how files are published to it. Developers can configure Web Publisher to render files to new formats and to regenerate existing Web pages when a template or stylesheet is updated.

Authors determine the content published on the pre-designed Web pages. To create a new Web page, an author selects a template and then enters the required content. To edit an existing Web page, an author can select the page directly or select the content file used to create that page.

Web Publisher automates the delivery of content to the Web through lifecycles and workflows.

### Applications Used to Create Web Pages

Authors and developers can create and edit content and other files through a variety of internal and external applications and methods. Web Publisher offers the following:

- **Integration with external applications**
  An external editing application is a third-party application that integrates with Web Publisher. In some cases, external applications are integrated with Documentum 5’s XML functionality, which allows you to interact directly with a Docbase through the application. Such integrations require Documentum Desktop on client workstations. You can create custom integrations for other third-party XML authoring tools using the API provided by DFC.

- **In-context editing**
In-context editing functionality lets authors locate files by navigating the live, staging, or in-progress version of the site. When authors find the files they are looking for, they can edit them directly. Authors can also add new Web pages. In-context editing is available only for published Web sites and must be enabled by an administrator or developer. In-context editing is enabled per Web site and is accessed via the **In-Context Editing** node.

- **Web Publisher Editor**

  This authoring tool lets users create and edit XML- or HTML-based content without having to understand XML or HTML. Web Publisher Editor does not display the entire XML or HTML file but instead displays only the content inside certain, specified elements. The content is displayed in editable fields. An associated Web Publisher Editor rules file—called an Editor rules file—determines which elements’ contents are shown and how each element’s content is displayed.

- **eWebEditPro**

  Ektron Inc.’s eWebEditPro HTML piece is a WYSIWYG editor that lets you create and edit HTML-based content while viewing it as it appears on the Web site. Users can also view the HTML source-code. Ektron’s eWebEditPro XML piece is a WYSIWYG authoring tool that lets users create and edit XML-based content such as content templates and stylesheets without having to understand XML. If your company has installed eWebEditPro, you can set eWebEditPro as your default HTML editor or XML authoring tool, or continue to use Web Publisher Editor.

- **Rules Editor**

  This gives users a graphical interface for creating and modifying Editor rules files, which are XML files that define how Web Publisher Editor displays content. The Rules Editor lets developers create Editor rules files without having to work directly with the XML. If you prefer to use a text editor, Web Publisher provides a separate Rules File Wizard to guide you in creating rules.

**How Web Publisher Publishes Content to the Web**

Publishing copies a Web page from a Docbase to a Web server. Web Publisher uses Documentum Site Caching Services (SCS) to publish Web pages. SCS chooses which content to publish and to what location according to the parameters in a publishing configuration. Administrators and developers create publishing configurations using Documentum Administrator.

SCS chooses when to publish according to the interval designated in a publishing job. When a user creates a publishing configuration, SCS automatically creates a publishing job. A job is a Documentum object that runs a specific method at specified intervals. The SCS publishing job regularly runs a method that initiates the publishing operation. Through Documentum Administrator, users can modify the job’s interval or run the job manually.

Web Publisher can be configured to automatically initiate the SCS publishing job when a Web page is approved and reaches its effective date. The effective date is the day a file
is to be published to the live Web site. If a file is published to more than one site, you can set different effective dates for the file for each site. Users can manually initiate the publishing job for one or more files by using the Publish command in the Web Publisher Classic view.

During creation of content, Web Publisher automatically publishes the content to inhouse Web servers so users can preview how the content appears on the Web. Users preview content using Web Publisher's preview command or using the appropriate URL for an internal site.

Web Publisher automatically removes content from a Web site when the content meets its expiration date.

You can use publishing configurations to create a Web site once and publish it in multiple formats and languages to multiple Web servers. For example, you can configure Web Publisher to render a site to HTML for browser users and then render the same site to WML for wireless users. If you enable globalization functionality, you can translate the site to multiple languages. You use the publishing configuration to set parameters to publish certain renditions and languages to certain servers.

For example, you could publish the same site's content to all of the following:

- An English-language, HTML-based Web site
- An English-language, WML-based site for wireless users
- A French-language, HTML-based site for France
- A French-language, WML-based site for wireless users in France
- A French-Canadian HTML-based site
- A French-Canadian WML-based site for wireless users

### How Files are Named in Web Publisher

Files in Web Publisher have two names:

- **File name**
  
The file name identifies the file on the Web and therefore must use characters allowed on the Web. For a list of characters, see File Names on the Web, page 23. The file name is required.

- **Descriptive name**
  
The descriptive name appears only in Web Publisher and gives you a way to more fully describe a file. The descriptive name is optional. If a file has no descriptive name, Web Publisher identifies the file everywhere by using the file name.

See also File Names on the Web, page 23.
File Names on the Web

If a file or folder is to be pushed to a Web site, it must use Web-safe characters (characters that can appear in a Web URL). Any character with an ASCII code greater than 127 is not a Web-safe character.

The following are Web-safe characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
$ - _ + & @ ! (),
```

Using Web Publisher and eWebEditPro

Ektron’s eWebEditPro application contains an HTML editor, and an XML authoring tool.

Ektron Inc.’s eWebEditPro HTML piece is a WYSIWYG editor that lets you create and edit HTML-based content while viewing it as it appears on the Web site. The integration between Web Publisher and eWebEditPro facilitates creation and editing of Web Publisher content by enabling you to create text, add images, and add hyperlinks. You can also view the HTML source-code. Ektron’s eWebEditPro XML piece is a WYSIWYG authoring tool that lets users create and edit XML-based content such as content templates and stylesheets without having to understand XML. If your company has installed eWebEditPro, you can set eWebEditPro as your default HTML editor or XML authoring tool.

For information on the integration between Web Publisher and eWebEditPro refer to Web Publisher Administration Guide.

For information on using eWebEditPro refer to the eWebEditPro user guide which can be found at:

```
ektron.com/software/released/ewebeditproxml/v41/userguide.pdf
```

Using Web Publisher and iMarkup

iMarkup is a Web-based plug-in tool that lets you attach annotations and free-form drawings to a Web page in Web view. You can create comments or diagrams on Web-based content to communicate any required or recommended changes to a content author. If the iMarkup integration has been installed then the iMarkup client components, menu bar, tree pane and online Help, are available for use in Web view. The menu bar contains an Organizer that displays user nodes, the currently logged in user, and any users who have attached annotations to the opened Docbase object. Under each user node will be displayed user created annotations or drawings. The tree pane contains options for creating notes, highlights, and drawings within a Web page, and a link to the iMarkup online Help. iMarkup documentation is provided by iMarkup not by Documentum.
For information on the integration between Web Publisher and iMarkup refer to *Web Publisher Administration Guide*.

For information on using iMarkup refer to the iMarkup documentation. If you have correctly installed the iMarkup plug-in you can access the iMarkup online Help by right-clicking on the Web page and choosing the iMarkup Help option. You can also access an iMarkup annotation plug-in user guide from the iMarkup Web site at:

[imarkup.com/docs](http://imarkup.com/docs)
Chapter 3

Technical Overview of WDK Applications

This chapter provides a technical overview of the architecture of WDK applications and other basic concepts needed in order to perform customizations.

These topics are described:
• Introduction, page 25
• WDK Application Directory Structures, page 26

Introduction

Web Publisher is a J2EE web application based on WDK and Webtop, which run on an application server. The key to customizing Web Publisher effectively and efficiently is to understand its architecture and directory and file structure.

All WDK applications have the following functionality:
• A Java tag library of easily configured Web-based UI widgets
• A Java framework that supports application-server based state management, messaging, branding, history, internationalization, and content transfer
• A set of configurable components that generate HTML widgets and provide access to Docbase functionality

The WDK application architecture incorporates two models: a presentation model that uses JSP tag libraries to separate Web page design from behavior, and a component model that encapsulates Docbase functionality in configurable server-side components.

Web Publisher is a web application customized from Webtop, which itself is built on top of WDK. Consequently, Web Publisher and Webtop are both WDK-based web applications. (Hereafter, any WDK-based web application will be called a WDK application.) Any WDK application runs on an application server and consists of a set of Java classes and interfaces, application, action and component configuration files, JSPs, Javascript files, and tag library files.

For more information on WDK application architecture, see the
Before You Customize

Prior to starting a customization, you should install WDK over Web Publisher. This will create a new developer version of the application and add documentation into the files you are customizing. For more information on how to install WDK over an existing application, see the Web Development Kit and Applications Installation Guide.

WDK Application Directory Structures

Figure 3–1, page 26 shows the WDK applications directory structure on which Web Publisher is modelled.

Table 3–1, page 27 describes the directories and files in the Web application root directory (Web_application_root in Figure 3–1, page 26):
### Table 3–1. WDK Application Directory and File Descriptions

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom</td>
<td>Directory for your custom application. This directory should contain your component definition files (in a config subdirectory), JSP pages, branding directories (theme subdirectory), and externalized strings (strings subdirectory).</td>
</tr>
<tr>
<td>help</td>
<td>Contains help files for WDK applications.</td>
</tr>
<tr>
<td>META-INF</td>
<td>Contains the J2EE manifest file.</td>
</tr>
<tr>
<td>wdk</td>
<td>Contains the following directories and files:</td>
</tr>
<tr>
<td></td>
<td>• Base JSP pages for login, navigation, modal dialogs, timeout, and tracing.</td>
</tr>
<tr>
<td></td>
<td>• WDK 4.2 config.xml and config.dtd.</td>
</tr>
<tr>
<td></td>
<td>• wdk4config, action, component, env, widget: JSP pages to support WDK 4.2 components.</td>
</tr>
<tr>
<td></td>
<td>• container: JSP pages for container components.</td>
</tr>
<tr>
<td></td>
<td>• contentXfer: Content transfer resources that are downloaded to the client.</td>
</tr>
<tr>
<td></td>
<td>• control: Contains reusable JSP pages that serve as mega-controls to display attributes.</td>
</tr>
<tr>
<td></td>
<td>• images: Contains images and icons used in the UI.</td>
</tr>
<tr>
<td></td>
<td>• samples: Sample JSP pages that exercise the controls in WDK.</td>
</tr>
<tr>
<td></td>
<td>• system: Contains WDK layer component JSP pages.</td>
</tr>
<tr>
<td>webcomponent</td>
<td>Contains these directories:</td>
</tr>
<tr>
<td></td>
<td>• admin: Contains administration component JSP pages.</td>
</tr>
<tr>
<td></td>
<td>• componentTestbed: Contains sample components.</td>
</tr>
<tr>
<td></td>
<td>• environment: Clipboard and preferences services JSP pages.</td>
</tr>
<tr>
<td></td>
<td>• finder: Contains the finder component.</td>
</tr>
<tr>
<td></td>
<td>• install: DQL scripts to install subscription object types in your Docbases.</td>
</tr>
<tr>
<td></td>
<td>• library: Component JSP pages.</td>
</tr>
<tr>
<td></td>
<td>• navigation: Navigation component JSP pages.</td>
</tr>
<tr>
<td>Directory or File</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| WEB-INF          | Contains these files:  
|                  |  
|                  | • classes — Contains Java classes that are used  
|                  |  
|                  |  
|                  |   by the Web application, including classes  
|                  |  
|                  |  
|                  |   for sub-applications, wdk, webcomponent,  
|                  |  
|                  |  
|                  |   and your custom application. Also contains  
|                  |  
|                  |  
|                  |   some property files (except for NLS  
|                  |  
|                  |   property files, which are located in each  
|                  |  
|                  |   sub-application’s strings directory).  
|                  |  
|                  | • lib — Contains Java application archive  
|                  |  
|                  |  
|                  |   (JAR) files required by WDK-based  
|                  |  
|                  |  
|                  |   applications.  
|                  |  
|                  | • tlds — Contains Documentum and  
|                  |  
|                  |  
|                  |   custom JSP tag libraries.  
|                  |  
|                  | • web.xml — File required by J2EE Web applications.  
|                  |  
|                  |   Specifies application startup directory and  
|                  |  
|                  |   application-specific servlets.  
|                  |  
|                  | • web.xml.authenticate — Same as web.xml,  
|                  |  
|                  |  
|                  |   with additional J2EE security element.  
|                  |  
|                  | • weblogic.xml — Contains Weblogic-specific  
|                  |  
|                  |   settings.  
|                  |  
|                  | **application, 1st_sub_application,**  
|                  |  
|                  | **2nd_sub_application, nth_sub_application**  
|                  | Application directory and sub-application  
|                  |  
|                  |   directories on which the application is  
|                  |  
|                  |   based.  
|                  |  
|                  | config.xml, config.dtd  
|                  | WDK 4.2 application configuration files.  
|                  |  
|                  | default.html, index.html (identical)  
|                  | Entry point to a WDK application.  
|                  |  
|                  | drl.html  
|                  | Used by the DRL component to display  
|                  |  
|                  |   content referenced by a DRL  
|                  |  
|                  | unstripped.jar  
|                  | Contains the entire WDK application with  
|                  |  
|                  |   all comments intact. In the deployed  
|                  |  
|                  |   WDK application, the comments and  
|                  |  
|                  |   spaces are stripped out.  
|                  |  
|                  | version.properties  
|                  | The version of WDK that is installed.  
|                  |  
|
The top-level directory (application & sub_application in Figure 3–1, page 26) for each sub-application contains the following directories and files:

**Note:** The directories listed are the common or required directories in an application or any of its sub-applications.

### Table 3–3. Sub-Application Directory and File Descriptions

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Configuration files for the application layer components.</td>
</tr>
<tr>
<td>include</td>
<td>(Optional) JavaScript files that are used within the application layer or within applications that extend the application.</td>
</tr>
<tr>
<td>strings</td>
<td>Externalized strings.</td>
</tr>
<tr>
<td>theme</td>
<td>Theme resources, including images, icons, and cascading stylesheets (CSS).</td>
</tr>
<tr>
<td>jsp_directories</td>
<td>(Optional) Any number of directories and subdirectories that organize and contain JSP files.</td>
</tr>
<tr>
<td>app.xml</td>
<td>Application configuration file. There is an app.xml file in each application layer, which extends the app.xml file from the next lower layer.</td>
</tr>
</tbody>
</table>

### The Custom Layer

All customized XML configuration and JSP files should be placed in the custom folder within the application. This custom folder is treated as the top-level application layer and overrides the functionality in any lower layers. This also makes it easier to preserve customizations after product upgrades.

The directory structure of the custom layer is flatter than the application itself. The config subdirectory holds all XML configuration files, extended from any other application layer, with no subdirectories. The strings subdirectory holds all NLS properties files, placed within /custom/strings/com/documentum/custom.

### Table 3–4. Structure of the Custom Layer

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom</td>
<td>JSP files extended from any other application layer (no subdirectory structure)</td>
</tr>
<tr>
<td>custom/config</td>
<td>XML configuration files extended from any other application layer (no subdirectory structure)</td>
</tr>
</tbody>
</table>
### Technical Overview of WDK Applications

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom/strings/com/documentum/</td>
<td>NLS properties files extended from any other application layer (no subdirectory structure)</td>
</tr>
<tr>
<td>custom</td>
<td></td>
</tr>
<tr>
<td>custom/themes</td>
<td>Contains any custom themes (branding) that you create. Requires a subdirectory structure. See the WDK Development Guide.</td>
</tr>
</tbody>
</table>

Custom Java classes are not located in the custom layer. They are located in the WEB-INF directory of the application, with the directory structure matching the package.

## How to Determine a Component from the UI

There is no straightforward way to determine which component underlies the portion of the UI that you want to modify. One of the difficulties is that many pages are displayed by container components, which provide common layout and behavior for a number of different components. For example, the properties container displays the attributes, history, and permissions components.

However, you can identify a component or at least its container in one of the following ways:

- Mouse over a link in a particular frame, and the component and JSP for that frame will be displayed in the status bar.
- Click in the frame, then right-click and choose Properties to display the same information as for the previous method.

For example, if you navigate to the Cabinets tab and mouse over a link or view the properties on the Properties page, you will see a URL in the status bar that includes the following:

```plaintext
.../dam/library/properties/properties.jsp?....... 
```

This identifies the component as the properties component in the dam layer, and the JSP mentioned here would be located in the `app_home/wp/library/properties` directory in the application.

Once you have identified the component, you can look in the reference section of the WDK manual (for the WDK, webcomponent, or Webtop layers) or this manual (for the Web Publisher layer) to get more information about it. If it is a container, the reference documentation will have more information about which components are used in that container. You can also more information about the component’s definition by looking up the component through listing the application’s components.

## Listing the Application’s Components

Each component is defined in an XML configuration file.
You can view information the entire list of components used in all application layers by viewing the componentlist component at http://host_name:port_number/app_name/component/componentlist. This component presents a display with the following information:

- Component ID
- Scope (type and application layer)
- Whether the component is a container
- Location of the XML configuration file, with a link to the file
- Description of the component
- Path to the JSP file for the component
- Path to the file containing the NLS resource strings
- Path to the Java class for this component
- Parameters that are passed to class methods

This information is extracted from the XML configuration file for that component, and it presents a handy centralized location from which to view component definitions for the application.

Determining Which Parts of the Component to Customize

Once you identify the component you want to modify, and which application layer the component resides in, you should decide which parts of the component must be changed. If the modification involves extending a current component by modifying a JSP or a properties file, you should refer to the configuration sections of the appropriate development guide for that application layer (the WDK and webcomponent layers are included in the WDK Development Guide, the Webtop layer in the Webtop Development Guide). If the modification involves extending a component by extending or creating a new Java class, you should follow the customizing Java sections of the appropriate development guide. Also, because most Web Publisher configurations and customizations can be achieved in the same manner as any other WDK application and specifically Webtop, see the Web Development Kit Development Guide and the Webtop Development Guide for general information and examples. This manual describes configurations and customizations specific to the Web Publisher layer only.

If you are changing behavior of a component, you should consider whether the behavior encapsulates business logic and will be extended to other Documentum clients. If so, you may want to consider creating a business object rather than building the logic directly in a WDK component. For more information, see the Web Development Kit Development Guide and the Documentum Foundation Classes (DFC) Development Guide. Refer also to
the development guidelines within these manuals to ensure successful components and controls.
Chapter 4

Web Publisher Application Configuration

Web Publisher application-wide component definitions are located in APP_HOME/wp/config/app. Web Publisher definitions are configured and extended in the same way as other WDK configurations. See Web Development Kit and Client Applications Development Guide for general information on application configuration files.

Menus are configured based on context-sensitive views (Chapter 6, Web Publisher Context Views) and roles (Chapter 7, Role-based Configuration). That is, the menu that is displayed in a particular component is based on the component and on the user’s role.

The following topics describe a Web Publisher application and its configuration:

- Web Publisher Environment Settings, page 33
- Web Publisher Cookies, page 34

Web Publisher Environment Settings

WCM environment settings are configured in the file WcmApplicationConfig.properties, which is located on the DOCUMENTUM_HOME/config directory on the J2EE server host. Some of the settings should not be changed. You should not modify this file directly. You should create a copy of the file with a name such as WcmCustomConfig.properties, in the same directory, and modify this file. Configurable settings are described in the table below:
### Web Publisher Application Configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>async-task.delay</td>
<td>Delay in seconds between the user queuing a task and the task executes</td>
</tr>
<tr>
<td>concurrent-task-queue.max_num</td>
<td>Maximum number of concurrent tasks that a user can perform simultaneously</td>
</tr>
<tr>
<td>deep-promote.item_limit</td>
<td>Maximum number of items to be promoted in deep promote operation</td>
</tr>
<tr>
<td>server-method.error_log_location</td>
<td>Path to the error log for WCM server methods</td>
</tr>
<tr>
<td>autorenderable.format.#</td>
<td>Specifies the formats that can be autorendered. Each format must have a subsequent number in the list.</td>
</tr>
<tr>
<td>clipboardservice.moveallversions</td>
<td>Set to false to move only the current version of a selected object during a move operation</td>
</tr>
<tr>
<td>deleteservice.enforceacl</td>
<td>Set to false to allow deletion of expired objects by users without delete permission</td>
</tr>
<tr>
<td>wcmContent.checkoutObjectInStagingANDChangeset</td>
<td>Set to true to allow an object within a staging change set to be edited</td>
</tr>
</tbody>
</table>

### Web Publisher Cookies

Cookies in Web Publisher are not configurable. The list of Web-Publisher specific cookies is provided here for information purposes only. To find information on setting and removing cookies, see *Web Development Kit and Client Applications Development Guide*.

**Table 4-1. Cookies in Web Publisher**

<table>
<thead>
<tr>
<th>Cookie Name</th>
<th>Description</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>COOKIE_COPYCATEGORY_NAME</td>
<td>Category in copy operation</td>
<td>WpCopyNonWcm</td>
</tr>
<tr>
<td>DIFF_TEMP_LOCATION_COOKIE_ID</td>
<td>Location for temporary download of files on which to perform diff</td>
<td>WebDeveloperPreferences, ViewSourceContainer</td>
</tr>
<tr>
<td>DIFF_APP_PATHCOOKIE_ID</td>
<td>Path to differencing application</td>
<td>WebDeveloperPreferences</td>
</tr>
<tr>
<td>VIEWSOURCE_APP_PATH_COOKIE_ID</td>
<td>Path to application for source viewing</td>
<td>WebDeveloperPreferences, ViewSourceContainer</td>
</tr>
<tr>
<td>COOKIE_IMPORT_TYPE_NAME</td>
<td>Type of file to be imported</td>
<td>ImportBase, ImportContent, ImportContentTemplate</td>
</tr>
<tr>
<td>Cookie Name</td>
<td>Description</td>
<td>Used by</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>COOKIE_IMPORT_POLICY_ID</td>
<td>Lifecycle policy ID for file to be imported</td>
<td>ImportBase, ImportContent, ImportContentTemplate</td>
</tr>
<tr>
<td>COOKIE_IMPORT_TEMPLATE_ID</td>
<td>Template of file to be imported</td>
<td>ImportBase, ImportContent</td>
</tr>
<tr>
<td>COOKIE_IMPORT_EFFECTIVE_DATE</td>
<td>Effective date of content to be imported</td>
<td>ImportBase, ImportContent</td>
</tr>
<tr>
<td>COOKIE_IMPORT_EXPIRATION_DATE</td>
<td>Expiration date of content to be imported</td>
<td>ImportBase, ImportContent</td>
</tr>
<tr>
<td>COOKIE_IMPORT_LOCALE</td>
<td>Locale of content to be imported</td>
<td>ImportBase, ImportContent, ImportContentTemplate</td>
</tr>
<tr>
<td>COOKIE_IMPORT_TEMPLATE_NAME</td>
<td>Name of template for file to be imported</td>
<td>ImportContent</td>
</tr>
<tr>
<td>PREF_SHOW_LOCATIONS</td>
<td>Sets display of locations in various components</td>
<td>LocationsViewAction, StandardViewAction, ThumbnailViewAction, ListManualLink, ContainedInContentList, ContainsContentList, Search, WhereUsed, ChannelClassic, ChannelStreamline, CSMYFilesClassic, MyFilesClassic, MyFilesStreamline, CSSubscriptionsClassic, SubscriptionsClassic, SubscriptionsStreamline</td>
</tr>
<tr>
<td>PREF_SHOW_THumbnails</td>
<td>Sets display of thumbnails</td>
<td>LocationsViewAction, StandardViewAction, ThumbnailViewAction, ListManualLink, ContainedInContentList, ContainsContentList, Search, WhereUsed, ChannelClassic, ChannelStreamline, CSMYFilesClassic, MyFilesClassic, MyFilesStreamline, CSSubscriptionsClassic, SubscriptionsClassic, SubscriptionsStreamline</td>
</tr>
<tr>
<td>INHIBIT_CHANGE_LOSS_WARNING</td>
<td>Removes display of warning on cancel checkout operation</td>
<td>CancelCheckout</td>
</tr>
<tr>
<td>Cookie Name</td>
<td>Description</td>
<td>Used by</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>INHIBIT_TURN_OFF_MESSAGE</td>
<td>Sets display of confirmation prompt</td>
<td>ThumbnailPreview</td>
</tr>
<tr>
<td>SELECTED_FONT_NAME_SIZE</td>
<td>Sets the user’s selected font name and size</td>
<td>WpContentMgr</td>
</tr>
</tbody>
</table>
The WebPublisher Editor uses several kinds of files to create Web content. The following diagram shows the files and their relationships. A brief description of the purpose of each file follows.

**Figure 5–1. Web Publisher Editor Files**

The types of Web Publisher files are summarized below:

- **Template**
  
  The template file is created by a template designer that models multiple web pages. When the user wants to create new content they select the template for the type of information that they are creating. WebPublisher creates a copy of the template in the selected location. (The template file is not used by the WebPublisher editor.)

- **Content file**
The content file is a specific instance of a particular template. When the content file is first created it will be identical to the template file. This is the file that is selected, and edited by the editor. The unique content that the user created is stored in the content file.

- **Rules file**

  The rules file is an XML file that describes the user interface to the editor. A template file must be associated with a rules file before the template can be used. This association is copied to the content file. Any number of template files may share the same rules file, and multiple content files can share the same rules file.

  When the editor starts, it downloads the content file and appropriate rules file. The rules file is parsed and stored. The content file is parsed, and each element in the content file is evaluated for applicable rules. Only one rule may specify the content of the element (the data that will be between the opening and closing XML tags), but there can be multiple rules that set attributes of an element. When a rule match is found, the editor uses the information in the rule element to construct the desired user interface component. Attributes on the rule specify the characteristics of the UI component that is built.

- **Presentation file**

  The presentation file is an XSL style sheet that controls the conversion of the XML content file into HTML for publication. (This file is not relevant to the creation of a custom control.)
Web Publisher Context Views

The Web Publisher application adds a wpcontext qualifier to the application. This qualifier is used in component and action definitions to limit the actions that are valid for the wpcontext value.

The values of wpcontext and other Web Publisher specific information are stored in a Web Publisher Context object (WcmAppContext).

The following topics describe the uses of wpcontext and the Web Publisher context interface:

- Using wpview, page 39
- Configuring Actions Using wpcontext, page 40
- Chapter 6, Web Publisher Context Views

Using wpview

The most common wpcontext value is wpview. This context value indicates to the application that the component is part of the Web Publisher application and that the component does not, for example, belong to the Webtop application that is a part of the Web Publisher application.

Custom Web Publisher components should scope the configuration to the wpview value of wpcontext. For example:

```xml
<scope wpcontext="wpview"/>
```

This will ensure that the wp/nonwp view-related actions are enabled and disabled properly and will control the launching of wp/nonwp components in the appropriate view.

Custom JSP pages that have `<dmfx:actionmultiselect>` tags should have the following argument:

```xml
<dmf:argument
name='<%=IWpConstant.WP_CONTEXT%>'
value='<%=currentWpcontext%>' />
```

The Web Publisher application sets the wpcontext parameter when the user clicks on a node in the browsertree or a tab in the tabbar. The value of wpcontext is set to wpview or nonwpview based on the value of the wpcontext scope in the component for the tree node or tab. When the configuration service resolves a component or action definition, it will get the wpcontext value from the session-scoped Context object if the parameter is not defined in the request-scoped Context.
Configuring Actions Using wpcontext

Web Publisher users wpcontext values to configure actions or UI for components. A set of wpcontext views and their valid and invalid actions are defined in the contextsensitiveview_config component, defined in /wp/config/app/contextsensitive_view_config.xml. This configuration file generates a configuration file, contextsensitive_view.xml, located in the same folder.

Your component class can use a wpcontext value to perform business logic. In the following example, the component definition

In the custom component configuration file, you can use a wpcontext view to hide or display parts of the UI. To do this, use a <filter wpcontext=...> element. In the following example from the search component definition, the filter displays one column of search results for nonwpview context, one column for wpview context, and several columns for both types of views:

```xml
<filter wpcontext='nonwpview'>
  <column>
    <attribute>a_controlling_app</attribute>
    <label></label>
    <visible>true</visible>
  </column>
</filter>
<filter wpcontext='wpsearch'>
  <column>
    <attribute>webreadyicon</attribute>
    <label></label>
    <visible>true</visible>
  </column>
</filter>
<column>
  <attribute>object_name</attribute>
  <label><nlsid>MSG_NAME</nlsid></label>
  <visible>true</visible>
</column>
```

If a JSP page contains action tags that can launch an action, the action control tag should contain an argument for wpcontext. This will scope the action to the valid actions for the view. The value of the wpcontext argument can be any valid view. In the following example, the wpcontext value is inherited from the current component:

```xml
<dmf:argument name='<%=IWpConstant.WP_CONTEXT%>' value='<%=currentWpcontext%>'/>
```

In the following example, a specific view is specified for the wpcontext value, so that the actionimage will launch the properties action that is scoped to wpchannellist:

```xml
<dmfx:actionimage name="propact"
  nlsid="MSG_PROPERTIES" action="properties" src="icons/info.gif"
  showifdisabled="false"> ...
  <dmf:argument name="<%=IWpConstant.WP_CONTEXT%>"
    value="wpchannellist"/>
</dmfx:actionimage>
```
Chapter 7

Role-based Configuration

Web Publisher is configured with four default roles: content author (sometimes called author), content manager (sometimes called manager), Web developer (sometimes called developer), and administrator. These roles are installed in a Docbase by the Web Publisher doc app.

If the user is not assigned a Web Publisher role, the fallback role model is defined in the Web Publisher app.xml file as the client capability model. See Web Development Kit and Client Applications Development Guide for details on the client capability model.

You can configure Web Publisher to change the functionality available to these roles and to replace or add roles to your application:

- Configure the actions that are permitted and the actions that are disallowed for each role
- Configure components so that their functionality is available only to certain roles

Default Roles

All roles can use both the classic and streamline views in the application. Some functionality is available only through the classic view. The following table shows role names, group_class attribute value, and Docbase groups that are assigned to the role as defined by the Web Publisher application:

<table>
<thead>
<tr>
<th>Role name</th>
<th>group_class (attribute of dm_group)</th>
<th>Contains these groups or roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcm_content_author_role</td>
<td>role</td>
<td>content author, wcm_content_manager_role</td>
</tr>
<tr>
<td>wcm_content_manager_role</td>
<td>role</td>
<td>content manager, wcm_content_manager_role</td>
</tr>
<tr>
<td>wcm_web_developer_role</td>
<td>role</td>
<td>web_developer, wcm_administrator_role</td>
</tr>
<tr>
<td>wcm_administrator_role</td>
<td>role</td>
<td>administrator</td>
</tr>
<tr>
<td>wcm</td>
<td>domain</td>
<td>all roles named above</td>
</tr>
</tbody>
</table>

The functionality available to each role is described below.
• **wcm_content_author_role**: The default view is streamline, with the following functionality available:
  — Connect/disconnect from Docbases
  — Workflow: view inbox, work on tasks, stop, pause, resume, submit
  — View object lists through categories, my files, subscriptions, and search using filters for status, locale, and object groups (templates | files | folders | files and folders)
  — Object information: versions, associations, permissions, renditions, translations, where used, audit trail, properties, and Web view
  — Object transactions: edit, checkout, cancel checkout, save, view, edit ACL, export, create rendition, create and view media files (if Media Services is enabled), subscribe, delete, send locator
  — Clipboard: copy, move, link
  — Globalization (if enabled): add, remove, and view translations, turn on or off and view fallback rules
  — View workflows for which the user is initiator or supervisor
  — Create and import content using templates
  — View object lists through non-Web Publisher cabinet navigation

• **wcm_content_manager_role**: The default view is streamline. The content manager has all of the author functionality and the following additional functionality:
  — View lists and create or import new content in Web Publisher cabinets and folders
  — Create folders within Web Publisher cabinets and folders
  — Change set operations: create, add to or remove from, add or remove supporting files, view listing of user's change sets
  — Publish: expire now, synchronize
  — Lifecycle: promote, demote
  — Virtual document: convert to simple or virtual document, view, add child, remove child

• **wcm_web_developer_role**: The default view is classic. Additional member of this role: **wcm_content_manager_role**. The Web developer has all of the content manager functionality and the following additional functionality.
  — Publish: check links
  — Lifecycle: power promote
  — View source
  — Difference against local file, between versions, reinstate version
  — Templates: create, update, delete content templates, rule templates, presentation templates, and external application templates, associate and make available or unavailable
• wcm_administrator_role: The default view is classic. The site administrator has all of the Web developer functionality and the following additional functionality.
  — Reports: workflows, templates, invalid objects, lifecycles, file formats, jobs, active objects, staged objects, promoted to active, about to be promoted
  — Site manager: templates, presentations, rules, previews, configurations, change sets
  — Create, update, or delete users, groups, roles, permission sets, alias sets, object types
  — Editions: create, export, list, delete, difference
  — Locales: add, remove, view list

Configuring Role-based Actions

Role configuration definitions are located in /application_root/wp/config/app. Role configurations are configured and extended in the same way as other WDK configurations. Two configuration files govern the actions that are available to roles: contextsensitive_role.xml and contextsensitive_role_config.xml.

**Note:** Before you change role configuration, copy these files into your custom directory (for example, /application_root/custom/config) and make your changes there.

The file contextsensitive_role.xml configures the actions that are not allowed for Web Publisher roles. The role is specified as the wpreverserole attribute on the <scope> element in a configuration file, which must match a role that is defined in the Docbase. The <scope> element contains all of the actions that are specifically disallowed for the role. For example, the web developer role is not permitted to add or delete locales:

```xml
<scope wpreverserole="web developer" wpcontext="wpview"> ... <action id="rolemenu" notdefined="true"></action> </scope>
```

If you do not want a role to be permitted to perform an action, add the action to the list for that role. For example, if you don't want a web developer to be able to reinstate a version, add the following line to the list of disallowed actions for the element <scope wpreverserole='web developer' ...>:

```xml
<action id='diffversions' notdefined='true'/>
```

The file contextsensitive_role_config.xml configures the minimum role that is required for a user to perform an action. The <required_minimum_role> element names the role, and the <valid_actions> element contains all of the actions that are permitted for the role and roles that inherit that role's functionality. For example, a web developer is permitted to perform the viewsource action:

```xml
<required_minimum_role name="web developer"> <valid_actions> <action name="viewsource"></action></valid_actions> </required_minimum_role>
```
If you want a role to be permitted to perform an action, add an <action> element within the <valid_actions> element for that role. For example, to allow the web developer to see the rolemenu UI, add the following lines to the <valid_actions> element in <required_minimum_role name="web developer">:

<action name="rolemenu"></action>

Configuring Role-based Components

Components are configured through the <filter> element in the component definition.

Filter tags within configuration blocks allow components to be shown or hidden based on the filter attribute. The filter attributes are the same as the attributes you can use with scope. The scope name and scope value attribute pair defines the scope for displaying the elements named within the filter element.

Filters are generally used on container components, so that one or more contained components are displayed only to qualifying roles. For example, the preferences component has a filter that displays web developer preferences only to web developers:

<contains>
  <component>general_preferences</component>
  <component>web_author_preferences</component>
  <filter wprole="web developer">
    <component>web_developer_preferences</component>
  </filter>
</contains>

Other features of the component can be filtered so that they are enabled for certain roles. For example, the checkin component has the following settings for a web developer:

<filter wprole="wcm_web_developer_role">
  <prompt_comment>true</prompt_comment>
  <require_comment>false</require_comment>
  <show_version>true</show_version>
</filter>

These settings prompt the web developer for a comment on checkin and display the version, but checkin does not require a comment.

This use of roles enables a much finer control over features than role-enabled action configuration.

Adding Roles

Roles are configurable in Web Publisher. Define your roles as needed in Documentum Application Builder (DAB) and then use the roles in your component (see Configuring Role-based Components, page 44) or action (see Configuring Role-based Actions, page 43) definitions.
Other Types of Role Configuration

In addition to action and component role configuration, the following configurable features are available:

- Setting the default login locale: The login and authenticate components have a `<wprole>` element that configures the role that is allowed to set the default locale.
- Changing a workflow supervisor: The startwpworkflowclassic component has a `<wprolepermittedtomodify>` element that sets the minimum role the user must have to change the supervisor of a workflow. If the reverse attribute on this element is set to true, the specified role is not permitted to modify the workflow supervisor.
- Ability to use unavailable templates: The app.xml file in `/application_root/wp/config` has a `<unavailabletemplerole>` element that specifies the role that is permitted to use an unavailable template to create new content.
By default, all groups and all users are displayed as alias suggestion values for a workflow performer. You can create workflows with dynamic performers, that is, performers whose identity is not known when the workflow is designed. You can create your own alias suggestion values to be displayed for any existing default or custom alias.

A dynamic workflow performer can be selected in the following ways:

- The workflow initiator selects the actual performer.
- A participant in the workflow specifies performers of future tasks.

You can configure dynamic performers in two ways:

- Initialize the performer names or aliases with values before the user can select a performer. You would do this, for example, to assign the current user to a certain task in the workflow.
- Filter the user names or aliases that are presented to the user for performer assignment. You would do this to direct the choice of performers based on some business logic.

The following Web Publisher component definitions can include performer settings:
startwpwfperformers, forwardwpworkflowtask, finishwpworkflowtask, rejectwpworkflowtask and repeatwpworkflowtask

Components that support dynamic performers have one or both sets of the following configurable elements:

- <aliasperformerfilters> and <aliasperformerinit>
  Supports aliases for specifying the performer at design time
- <dynamicperformerfilters> and <dynamicperformerinit>
  Supports dynamic performers that are resolved at run time

The configurable elements for specifying dynamic performers are described in the following table. See the configuration descriptions for specific components in Web Development Kit and Client Applications Development Guide to find the exact set of configurable elements for each component.
<performers>
Contains the definition of custom performers in the following elements:
<aliasperformerfilters>, <aliasperformerinit>,
<dynamicperformerfilters>,
<dynamicperformerinit>

<aliasperformerfilters>
Default alias values. The content of this element must be interpreted by the class named in the
<class> child element.

<aliasperformerfilters>.<class>
Fully qualified filter class name that implements IAliasPerformerFilter. Can contain optional configuration information.

(aliasperformerinit>
Alias initializer classes

<aliasperformerinit>.<class>
Fully qualified initializer class name that implements IAliasPerformerInit. The content of this element must be interpreted by the class named in the <class> child element.

<dynamicperformerfilters>
Dynamic performer values. The content of this element must be interpreted by the class named in the <class> child element.

<dynamicperformerfilters>.<class>
Fully qualified filter class name that implements IDynamicPerformerFilter. Can contain optional configuration information.

<dynamicperformerinit>
Dynamic performers initializer classes. The content of this element must be interpreted by the class named in the <class> child element.

<dynamicperformerinit>.<class>
Fully qualified initializer class that implements IDynamicPerformerInit. Can contain optional configuration information.

Note: Any custom elements that are defined within a *.performerfilters or *.performerinit element must be interpreted by the class named in the sibling <class> element. The init or filter class is given a handle to the configuration information and can use the content in any way that you design it to do so. For example, the init class AssignCurrentUserToAlias in Web Publisher interprets the values of the <role> and <aliasname> elements in the startwpfperformers component definition.

Example 8-1. Setting up Aliases for Dynamic Performers
The startwpfperformers component configures an alias and the minimum Docbase role that should be assigned to that alias. The alias is not defined in the Docbase; it is an alias defined in the Web Publisher application. The following example changes the alias for the list of valid workflow performers to a custom role (custom_author_role) that has been defined in the Docbase:

<aliasperformerinit> <class
name="com.documentum.wp.library.workflow.performers.
AssignCurrentUserToAlias">
<!-- Defines the minimum WP role the user must have in order to be able to have his/her name assigned to the defined alias.--> <role>custom_author_role</role>
<!-- Defines the alias name to have the current user assigned to. -->
<aliasname>custom_author</aliasname> </class>

48 Web Publisher Development Guide
</aliasperformerinit>
If you have Content Intelligence Services installed, you can map CIS properties to Docbase attributes. To map CIS properties, you write mapping rules in the cis_component.xml file. The default path for cis_component.xml is:

\<app>\wp\config\library\cis\object_type_folder\cis_component.xml

You must create a custom component that extends the Web Publisher CIS component in order to protect your mapping rules from upgrade. For example, copy cis_component.xml and place it under the Web Publisher application root directory in /custom/config. For details on how to extend component definitions, see Web Development Kit and Client Applications Development Guide.

You can define within the CIS component definition a set of mapping rules for each object type. You write a rule using the following syntax:

<attribute show_all_values="true">
  <name>Docbase_property</name>
  <cisname>Docbase_property</cisname>
</attribute>

where the <name> element signifies the Docbase property, and the <cisname> element signifies the name of the XML file element from which the property value is extracted. The <cisname> value and the XML element name must match.

The <cis_name> element can contain either the CIS property map entry or an <nlsid> element. If the <nlsid> element is not used, the data dictionary label for the attribute will be used. If the <nlsid> element is used, the value specified in an NLS properties file will be used as a label for the attribute.

Example 9–1. cis_component.xml

The following is an example of the cis_component.xml file with multiple Docbase properties called “title, authors, and keywords” defined:

<attribute_list> <attribute>
  <name>title</name>
  <cis_name>title</cis_name>
</attribute>

<attribute>
  <name>authors</name>
  <cis_name>author</cis_name>
</attribute>

<attribute>
  <name>keywords</name>
  <cis_name>keywords</cis_name>
</attribute>
To apply a different set of mapping rules to custom object types, use the `<scope>` element in the CIS component definition. For details on how to use the `<scope>` element, see *Web Development Kit and Client Applications Development Guide*.

**Note:** To see changes in XML files, you must refresh the configurations in memory. Navigate to `/application_root/wdk/refresh.jsp` to refresh the configuration files.
Chapter 10
Configuring the Content and HTML Editors

The content and HTML editors can be customized through simple text editing. The file that contains Rules Editor settings is EditorUI.properties, which is located in /wp/strings/com/documentum/contenteditor/resources.

Configuring the Look and Feel of the Web Publisher Editor

The look and feel of the Web Publisher Editor is controlled by the following lines in the EditorUI.properties file:

# In current implementation, DefaultTheme and background color customization is only supported when using JGoodies
DefaultTheme=com.jgoodies.plaf.plastic.theme.ExperienceBlue

Button.disabled.background.color = C2D0EA
Button.enabled.background.color = E5E1D3
CheckBox.background.color = C2D0EA
ComboBox.background.color = C2D0EA

# other JGoodies themes:
# DefaultTheme=com.jgoodies.plaf.plastic.theme.AbstractSkyTheme
# DefaultTheme=com.jgoodies.plaf.plastic.theme.BrownSugar
# DefaultTheme=com.jgoodies.plaf.plastic.theme.DarkStar
# DefaultTheme=com.jgoodies.plaf.plastic.theme.DesertBlue
# DefaultTheme=com.jgoodies.plaf.plastic.theme.DesertBluer
# DefaultTheme=com.jgoodies.plaf.plastic.theme.DesertGreen
# DefaultTheme=com.jgoodies.plaf.plastic.theme.DesertRed
# DefaultTheme=com.jgoodies.plaf.plastic.theme.DesertYellow
# DefaultTheme=com.jgoodies.plaf.plastic.theme.ExperienceGreen
# DefaultTheme=com.jgoodies.plaf.plastic.theme.InvertedColorTheme
# DefaultTheme=com.jgoodies.plaf.plastic.theme.Silver
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyBlue
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyBluer
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyBluerTahoma
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyGreen
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyKrupp
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyPink
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyRed
# DefaultTheme=com.jgoodies.plaf.plastic.theme.SkyYellow
You can easily change to a different JGoodies theme by commenting out the current default line and removing the comment marker from the theme that you want. See the JGoodies web site at http://www.jgoodies.com to learn more about the JGoodies look and feel.

If you want to use the Windows look and feel, which is what the previous versions of Web Publisher Editor used, comment out the default line and remove the comment marker from the following line:

```
DefaultLookAndFeel=com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```

If you want to create your own custom look and feel, you can put your look and feel package and class name in the DefaultLookAndFeel key and comment out all JGoodies lines. For example, if you created a look and feel class called MyLookAndFeel.class in “com.mycompany”, the DefaultLookAndFeel entry in the EditorUI.properties would be:

```
DefaultLookAndFeel=com.mycompany.MyLookAndFeel
```

You would also comment out the following lines:

```
#DefaultTheme=com.jgoodies.plaf.plastic.theme.ExperienceBlue

#Button.disabled.background.color = C2D0EA
#Button.enabled.background.color = E5E1D3
#CheckBox.background.color = C2D0EA
#ComboBox.background.color = C2D0EA
```

### Other Web Publisher Editor Configurations

The following table describes the editor settings that you can configure in EditorUI.properties.

**Note:** Not all of the settings are configurable. Change only the ones listed in the table below.

**Table 10-1. Rules Editor Settings**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultFont</td>
<td>Specify default font name, style, and size in pixels</td>
</tr>
<tr>
<td>.showFontSelection</td>
<td>Set to true to allow user to change documentum font in the content editor</td>
</tr>
<tr>
<td>.showDebugButton</td>
<td>Set to true to display a Debug button in the content editor menu bar</td>
</tr>
<tr>
<td>.showDebugScreen</td>
<td>Set to true to split the HTML editor screen to display debugging</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>.showDebugButton</td>
<td>Set to true to display a Debug button in the HTML editor</td>
</tr>
<tr>
<td>import_binary_formats</td>
<td>Specifies formats that should be imported as binary files</td>
</tr>
</tbody>
</table>
Chapter 11

Configuring the Web Publisher Editor

The Web Publisher editor can be customized through simple text editing. The file that contains rules editor settings is RulesEditor.properties, which is located in WEB-INF/classes/com/documentum/contenteditor/resources. The following table describes some of the editor settings that you can customize.

Table 11–1. Web Publisher Editor Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIMain.defaultFont</td>
<td>Specifies the font to use in the editor. Must be Arial Unicode to display non Latin-1 characters. Users must have the specified font installed on their local system.</td>
</tr>
<tr>
<td>UIMain.defaultBoldFont</td>
<td></td>
</tr>
<tr>
<td>TextSelectorRuleEditor</td>
<td></td>
</tr>
<tr>
<td>GraphicRuleEditor.useTabs</td>
<td>Set to true to use tabs in editor. Default = false</td>
</tr>
<tr>
<td>Log.trace</td>
<td>True to turn on editor logging</td>
</tr>
<tr>
<td>Log.traceDest</td>
<td>True to print the location of the stdout trace file</td>
</tr>
<tr>
<td>UIMain.lookAndFeel</td>
<td>Specify a fully qualified class name for a Swing look and feel class</td>
</tr>
<tr>
<td>.windowTitle</td>
<td>Specify a string title for the editor window</td>
</tr>
<tr>
<td>.minimumLeftPanelSize</td>
<td>Minimum size in pixels</td>
</tr>
<tr>
<td>.minimumRightPanelSize</td>
<td></td>
</tr>
<tr>
<td>.defaultFont</td>
<td>Specify default font name, style, and size in pixels</td>
</tr>
<tr>
<td>TemplateTree.showTextView</td>
<td>Set to true to display a text tab in the tree</td>
</tr>
<tr>
<td>TemplateTextView.XXX</td>
<td>Specifies messages for parsing error, success, template update, and update failure</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.border</td>
<td>True to display a border for the editor component</td>
</tr>
<tr>
<td>RuleModel.checkboxDefaults</td>
<td>Specifies strings to use for checkboxes in the editor</td>
</tr>
</tbody>
</table>
Chapter 12

Customizing Web View

Your application may require a customization of the Web View feature that enables previewing of how content will appear on the Web. The Web View URL in Web Publisher is constructed as a static URL,, and your application may require a dynamic URL. The necessary steps and sample code for customization are provided below.

**To customize the Web View URL:**

1. Create a Java class that extends WcmWebPublishingService.
2. Overwrite either getWebViewURL() or getWebViewURLPrefix() in your extended class.
3. Open dbor.properties, which is located in the directory %DOCUMENTUM%/config,
4. Locate WcmWebPublishingService and modify the classpath of that entry to the full class path of your new extended class.
5. Restart your Web server.

The getWebViewURL() method calculates the URL for previewing an object. The method calculates a direct URL path beginning with the URL prefix stored in the appropriate dm_webc_config object and then appends the actual cabinet folder path to the URL prefix.

```java
/**
 * @param objectID The object id to be webviewed
 * @param channelID The channel from which the object is linked to
 * @param objStatus The status of the object ("WIP"|"Staging"|"Active")
 * @param objFormat The format of the object to be webviewed
 * @param pageMod The "page modifier" of the rendition to be webviewed
 * @param pageNum The "page number" of the rendition to be webviewed
 * @param formatMod The "format modifier" of the format to be webviewed
 * @return The URL to webview an object
 */
public String getWebViewURL (IDfId objectID, IDfId channelID, String objStatus, String objFormat, String pageMod, int pageNum, String formatMod, IDfId webcacheID) throws DfException {
    IDfsSession session = null;
    try {
        session = getDfsSession();
        IWcmContent doc = (IWcmContent) getWcmAppContext().getObject(objectID, IWcmContent.class.getName(), session);
        String url = "";
```

Web Publisher Development Guide 59
Customizing Web View

String urlPrefix = getWebViewURLPrefix(
    objectID, channelId, objFormat, webcacheID);

if (urlPrefix != null && urlPrefix.length() > 0)
{
    String pubName = doc.getPublishedName(objFormat, 0, pageMod);
    url = urlPrefix + doc.getURLPath(channelID) + pubName;
}
return url;
}
finally
{
    releaseDfSession(session);
}

The getWebViewURLPrefix() method gets the prefix of the Web View URL based on the information in the applicable site publishing configuration.

/**
 * @return URL prefix
 */
public String getWebViewURLPrefix (IDfId objectID, IDfId channelId,
    String objFormat, IDfId webcacheId)
    throws DfException
{
    IDfSession session = null;
    IDfCollection collection = null;
    IDfCollection collection2 = null;
    try
    {
        session = getDfSession();
        IWcmContent doc = (IWcmContent) getWcmAppContext().getObject(
            objectID, IWcmContent.class.getName(), session);
        String objStatus = doc.getVersionStatus();
        // Need to check if a specific rendition has been requested,
        // if none, then take any
        String DQLFormatSuffix = "";
        if (objFormat != null && objFormat.length() > 0)
        {
            DQLFormatSuffix = " and any (source_formats = '" + objFormat + "' or any(i_position=-1 and source_formats is null))";
        }
        String DQLSuffix = "";
        String versionSuffix = "";
        if (webcacheId != null && webcacheId.isNull() == false )
        {
            DQLSuffix = " and r_object_id = '" + webcacheId + "'";
        }else
        {
            versionSuffix = " and any version_labels = '" + objStatus + "'";
        }
        // first, get the URL prefix
        // look up the appropriate cache server object for the selected
        // delivery cabinet and lifecycle state
        String queryStr = "select r_object_id, object_name, target_id from
dm_webc_config where is_active=1" + versionSuffix + " and source_folder_id = '" + channelId.toString() + "'" + DQLFormatSuffix + " " + DQLSuffix + DQLFormatSuffix;

    }
IDfQuery query = new DfQuery();
query.setDQL(queryStr);

String URLHost = "";
try {
    collection = query.execute(session, IDfQuery.DF_READ_QUERY);
    while (collection.next()) {
        String cacheTargetID = collection.getString("target_id");
        // first one only

        // determine the host info from the associated dm_webcache_target
        queryStr = "select target_virtual_dir from dm_webc_target where r_object_id = " + cacheTargetID + "\n";
        query.setDQL(queryStr);
        collection2 = query.execute(session, IDfQuery.DF_READ_QUERY);
        if (collection2.next()) {
            URLHost = collection2.getString("target_virtual_dir");
            if (!URLHost.endsWith("/"))
                URLHost = URLHost + "/";
        }
    }
    if (Trace.WCMPUBLISH)
        Trace.println("WcmWebPublishingService.getWebViewURLPrefix: query returns nothing: " + queryStr);
    }
    catch (DfException e) {
        if (Trace.WCMPUBLISH)
            Trace.println("WcmWebPublishingService.getWebViewURLPrefix: exception: " + e.getMessage());
        throw e;
    }
    return URLHost;
} finally {
    if (collection != null)
        collection.close();
    if (collection2 != null)
        collection2.close();
    releaseDfSession(session);
}
Web Publisher Tracing

There are several ways to enable tracing in Web Publisher during development:

- Use a mouse CTRL click on Documentum icon in the Web Publisher frameset.
- Set a flag to true in the properties file (described below).
- Navigate to /wp/app/tracing.jsp.

**Caution:** High trace levels can quickly generate very large log files.

The Web Publisher tracing page allows you to turn DMCL tracing on or off and to specify the log. The tracing page tracing.jsp is located in /wp/app.

The tracing UI begins with some general tracing settings. Note that trace levels are additive. For example, messages from level 1 are added to messages from level 2 when tracing is set at level 2.

<table>
<thead>
<tr>
<th>Tracing is enabled for current session</th>
<th>Must be checked in order to use any subsequent tracing flags.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn on DMCL trace</td>
<td>Check box to turn on DMCL tracing.</td>
</tr>
<tr>
<td>Trace file path</td>
<td>File path on the Web Publisher J2EE server host</td>
</tr>
<tr>
<td>Trace file level</td>
<td>DMCL trace level (levels are additive):</td>
</tr>
<tr>
<td></td>
<td>1 = unknown</td>
</tr>
<tr>
<td>Turn on DFC trace</td>
<td>Check box to turn on DFC tracing.</td>
</tr>
<tr>
<td>Trace file path</td>
<td>File path on the Web Publisher J2EE server host</td>
</tr>
<tr>
<td>Trace file level</td>
<td>DFC trace level (levels are additive):</td>
</tr>
<tr>
<td></td>
<td>1 = exceptions</td>
</tr>
</tbody>
</table>
Web Publisher Tracing Flags

The following tracing flags are defined in the file WpTraceProp.properties, which is located in /WEB-INF/classes/com/documentum/wp/resources.

<table>
<thead>
<tr>
<th>Flag type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifecycle</td>
<td>WPLIFECYCLE traces task start, run, and finished processing. WPLIFECYCLES_SETTING traces init, commit changes, and state symbols.</td>
</tr>
<tr>
<td>Delete</td>
<td>WPDELETEOBJECT traces and reports deletion of versions and translations</td>
</tr>
<tr>
<td>General settings</td>
<td>WPGENERAL_SETTING traces initialization of Docbase controls, Web-based editor, simple search, site protection, set default ACL, set edition configuration name. WPFILE_FORMATS_SETTING traces changes to available file formats. WPEXTERNAL_RESOURCES_SETTING traces the component methods for the externalresources component.</td>
</tr>
<tr>
<td>Roles</td>
<td>WPROLE traces Web Publisher role and reverse role initialization</td>
</tr>
<tr>
<td>Search</td>
<td>WPADVANCED_SEARCH traces the pass of query to simple search. WPCS_SEARCH prints the query in search for supporting files.</td>
</tr>
<tr>
<td>Clipboard</td>
<td>WPCLIPBOARD traces calls to onNextPage(), onOk(), copy of non-WCM content, success and failure of non-WCM copy, move failure, and view clipboard query and status.</td>
</tr>
<tr>
<td>Editions</td>
<td>WPADDEDITION traces the add edition process, WPREMOVEEDITION traces the remove edition process</td>
</tr>
</tbody>
</table>

Web Publisher Tracing Flags, continued
<table>
<thead>
<tr>
<th>Flag type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content editor</td>
<td>WPCONTENTEDITOR traces folder query, get Docbase path, get WCM paths, WCM channel found or not found, cabinet is or is not channel, get content errors, displays content encoding, rules file not found, content or rules file not checked out, null content file, post processing complete or failed, create new content, policy ID assigned or failed, new object ID, import folder does not exist, document name not URL safe, already a document with duplicate name, DFC exception, preview ID with object ID and layout ID, preview failed, can't get DfSession.</td>
</tr>
<tr>
<td>In-context Editor (ICE)</td>
<td>WPICE traces ICE menu failure, new object ID, site URL, modified date changed, can't start workflow, rendition not found, publish success or failure, cancel checkout, WebCache DQL, proxy URL, site URL, rendition job process, checkin,</td>
</tr>
<tr>
<td>Checkin</td>
<td>WPCHECKIN traces jump to named component pages, onCommitChanges(), get file path, version selection, service start, error, post-checkin operation, service finished, upload</td>
</tr>
<tr>
<td>Checkout, cancel, checkout, edit, import</td>
<td>WPCHECKOUT, WPCANCELCHECKOUT, and WPEDIT trace object ID, start service, editing app, pre-checkout for 3rd party object, validate object, object links, non-current version. WPIMPORT traces new object ID, object name, and template ID.</td>
</tr>
<tr>
<td>View source and diff</td>
<td>WPVIEWSOURCE and WPDIFF traces error with preferred source viewer or differencing tool, export failure, and export path. WPVIEWSOURCE adds export object IDs, WPDIFF adds whether object is checked out.</td>
</tr>
</tbody>
</table>

WCM Tracing Flags

The following tracing flags are defined in the file WcmTraceProp.properties, which is located in the directory DOCUMENTUM_HOME/config.,
<table>
<thead>
<tr>
<th>Flag type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMCL tracing</td>
<td>DMCLENABLEDBYDEFAULT turns on DMCL tracing</td>
</tr>
<tr>
<td>WCM config</td>
<td>WCMAPPCONFIG traces WcmApplicationConfig lookup results and errors</td>
</tr>
<tr>
<td>Asynchronous tasks</td>
<td>WCMASYNCTASK traces asynchronous task status update, abort, and process from start to completed</td>
</tr>
<tr>
<td>Content transfer services</td>
<td>WCMCANCELCHECKOUTSRVC not used. WCMCHECKINSRVC traces checkin validation DQL and checkin errors. WCMCHECKOUTSRVC traces validation DQL, checkout failure due to current checkout, archive, replica, or reference. WCMEDITSRVC not used.</td>
</tr>
<tr>
<td>Change sets</td>
<td>WCMCHANGESET traces files added (ID), lifecycle state, state promoted to, document next state, promote failed, power promote, post process, batch promote errors</td>
</tr>
<tr>
<td>Clipboard</td>
<td>WCMCLIPBOARDSRVC traces move, copy, and link attempted on WCM and non-WCM folders, move to template folder</td>
</tr>
<tr>
<td>Delete</td>
<td>WCMDELETEOBJECT traces number of versions, object in an edition, delete errors, translation IDs, folder ID, delete template.</td>
</tr>
<tr>
<td>DFC</td>
<td>ENABLEDFCEXCEPTIONCALLSTACK turns on DFC stack tracing</td>
</tr>
<tr>
<td>DQL</td>
<td>WCMDQLUTIL returns DQL strings when they are read from the properties file WcmDQL.properties</td>
</tr>
<tr>
<td>Jobs</td>
<td>WCMDYNAMICCONTENTJOB traces execute of dynamic_content_job process. WCMONITORJOB traces the following jobs: expire now, promote to active, effective date, process dynamic content, promote to expired, send expire notice to author. WCMOBJECTBAGJOB traces object bag folder, validate and create prefabricated objects from templates, destroy out of date objects. WCMOBJECTBAGJOB_DETAIL traces count in object bag, add to bag, and object name added.</td>
</tr>
<tr>
<td>Lifecycles</td>
<td>WCMLIFECYCLESRVC traces add to batch expiration and batch expiration failure, add to batch promotion, promotion arguments and errors, power promote object state and power promote failure, expire object name and ID, expire error, update ACL, object state, change set and document handler error, promote, promote object ID, demote, make template available or unavailable, batch promote. WCMSESSIONLIFECYCLEEVENT traces the broadcast to listener, register and unregister listener for the session.</td>
</tr>
</tbody>
</table>

WCM Tracing Flags, Continued
### Flag Publish types

<table>
<thead>
<tr>
<th>Flag Publish type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish</td>
<td>WCMPUBLISH traces object ID added to batch, WebCache ID, force publish, asynchronous setting, and list of IDs, Web Cache server methods, object already published, bad URL publishing info, no WebCache config found, bad web view URL prefix.</td>
</tr>
<tr>
<td>SAX parser</td>
<td>WCMPARSERERROR writes Sax parser error message to standard output.</td>
</tr>
<tr>
<td>Utilities</td>
<td>WCMUTIL traces current user is superuser, current user permission on object, fail to load a properties file. WCMTRACE tracks the setting of DMCL and DFC trace on or off and trace level.</td>
</tr>
<tr>
<td>WCM server methods</td>
<td>WCMSERVERMETHOD_SESSION traces method execution timestamp, execution errors, Docbase or user names not specified, no session, workflow attachment IDs. WCMSERVERSIDEUTIL traces set ACL, object linked or unliked to folder, policy (lifecycle) attached or detached, version label, a_special_app value, execute on client.</td>
</tr>
<tr>
<td>WCM session</td>
<td>WCMSESSIONENABLEDBYDEFAULT sets session tracing on or off.</td>
</tr>
<tr>
<td>Workflows</td>
<td>WCMPROCESSWORKFLOW traces workflow ID associated or dissociated to object ID, workflow process errors, automatic change set destruction. WCMSTRATTRANSLATIONWF traces translation activity, start translation workflow, locale name, default translation workflow error, process ID, change set success.</td>
</tr>
</tbody>
</table>

Additional settings in the WcmTraceProp.properties resource file include DMCL and DFC trace file levels and file paths. These settings can be changed through the UI.

The content editor server has two tracing flags that are set in the file EditorServerTraceProp.properties in /WEB-INF/classes/com/documentum/contenteditor/resources. These trace flags must be turned on through the properties file.

<table>
<thead>
<tr>
<th>Server</th>
<th>EDITORSERVER traces lookup of the content editor application, parser errors, validation and entity resolution, double byte encoding exception, checksum error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor operations</td>
<td>CONTENTIF traces export path, export error, XML file encoding</td>
</tr>
</tbody>
</table>
Chapter 14

Customizing Web Publisher Editor Controls

Web Publisher Editor and the Rules Editor work together to build a user interface for end-users to edit content files and Editor rules files. Web Publisher Editor consists of a client-side applet and a server-side servlet. In WebPublisher 4.x, the server side code was in xmlTemplateServer.jar, and the APPLET tag was generated by AppletMgr.class at runtime. In Web Publisher 5, the _ceedit_ component generates the Web Publisher Editor applet, and the _reedit_ component generates the Rules Editor applet. The applets are the same, but they are launched from different starting points (different components) in the Web Publisher user interface.

When a user performs an action that requires an editor, such as editing or viewing an object, the appropriate editor is launched based the user’s preferences and on the type of selected document for example, Web Publisher Editor and eWebEditPro (HTML documents), or a 3rd-party editor such as Microsoft Excel.

The Web Publisher Editor applet consists of one or more editor controls. These controls are sometimes called widgets. You can include a custom control within a `<tagcontent>` or `<tagattribute>` element in an Editor rules file. If a rule is defined in a `<tagcontent>` element, the control produces content that replaces the content of the specified XML element. If the rule is defined in a `<tagattribute>` element, the control produces content that sets an attribute on the content XML element.

Custom controls are described in the following topics:

- Custom Control Overview, page 69
- Custom Control Requirements, page 70
- Using a Custom Control in a Rule, page 73
- Writing and Packaging the Control, page 74
- Custom Control Messages, page 76
- Custom Control Example for Multi-Line Text Area, page 77

**Custom Control Overview**

You can write your own controls that address specific needs that are not covered by the standard controls. For example, you might write a date selection control that is knowledgeable of accounting week or fiscal year information specific to your company.
Custom controls are wrapped by the Web Publisher Editor class CustomWidgetWrapper. This class instantiates the custom class when a custom rule is encountered, that is, `<tagcontent>.<custom>` or `<tagattribute>.<custom>` in an Editor rules file.

The wrapper class also wraps all of the methods of the EditWidget interface in order to trap any exceptions in custom control code. The wrapper class provides a boundary between the product code and custom control code. If custom control code throws an exception, the wrapper code will protect the remaining content of the data stream to prevent data loss in the document.

## Custom Control Requirements

A custom Web Publisher Editor control must conform to several requirements in order to work with the standard controls. Where possible these requirements are tested programmatically before and during the instantiation of the custom class. Some requirements cannot be tested.

Requirements are described in the following topics:

- Swing Compatibility, page 70
- Control Construction, page 71
- EditWidget APIs, page 71
- Custom Control Wrapper, page 73

### Swing Compatibility

The Web Publisher Editor uses the Java Swing class for its user interface. This allows the editor to run in the Sun Java 2 virtual machine through the plugin capabilities of the browser.

To ensure Swing compatibility, a control must have the following characteristics:

- subclasses `JComponent`
  - The wrapper code tests for this after instantiation to prevent the inclusion of heavyweight components and possible layout problems.

- lightweight
  - A heavyweight component creates the component in the native windowing system. A lightweight component draws itself into its parent’s drawing space. Avoid importing “java.awt.*” into the source file. The following java.awt.* classes can be used in a lightweight component: Font, Dimension, Color, Insets, and layout managers such as GridBagLayout.

- thread-safe
  - Most parts of Swing are not thread-safe. In your control must create new threads, make sure that your code implements thread safety.

- avoid overriding paint()
Generally you should not need to override the paint() method in your control. None of the standard editor controls override this method. If your control makes significant changes to its layout, you may have to call validate() and repaint(). You can also force a re-layout of the complete applet, by calling validate() and repaint() on the top level JApplet. A reference to the top level can be obtained from the ConfigMgr.

Control Construction

When Web Publisher Editor builds the interface and discovers a custom control, the editor constructs a CustomWidgetWrapper object. The CustomWidgetWrapper performs several checks to be sure that it can load the class specified in the rule. The wrapper class then calls the constructor of the custom element.

The following example shows the expected signature for the control class:

```java
import com.documentum.contenteditor.client.guiMgr.EditWidget;
import com.documentum.contenteditor.client.main.ConfigMgr;
import javax.swing.*;
public class Foo extends JComponent implements EditWidget {
    Rule m_rule;
    ConfigMgr m_cm;
    public Foo (Rule rule, ConfigMgr cm) {
        m_rule = rule;
        m_cm = cm;
    }
    // Rest of stuff needed for EditWidget interface omitted
}
```

EditWidget APIs

The following methods are defined in com.documentum.contenteditor.client.guiMgr.EditWidget:

- String getContent(int type)

  This method is called when the document is being saved or previewed. The type argument is EditWidget.SAVE or EditWidget.PREVIEW. Most controls do not need to distinguish between save and preview. Some controls may wish to return different content depending on whether the control is completing the content of the enclosing tag or filling in an attribute on the enclosing tag (valid characters may be different). The control does not return the enclosing tag, which is handled by code in the dataStreamMgr package. The custom control is responsible, however, for ensuring that the String that is returned does not break the XML document. In most cases this means that the control converts a “<” character to the character entity &lt;.

  If the control needs to put additional information in the output stream in the form of a XML processing instruction (PI), then you can call the Rule object associated with the custom control to reset and add this information. The PIs are written out...
with dctmEditor as the target. For example, if the Rule.addPI() method is called as 
m_rule.addPI("foo","bar"), then the resulting PI will be written as <?dctmEditor
foo=bar ?> in the document output stream. PIs will immediately follow the open tag
for the element that caused the control to be created.

- void setContent(String content)

  The setContent() method is called when the editor initializes. The string passed in
to the method is the content of the XML element. If the rule is in a <tagcontent>
element, the content of the XML element is passed to the method. If the rule is in a
<tagattribute> element, the current value of the attribute is passed to the method.

  The complexity of the processing required in setContent() depends on the nature of
the control. In the standard controls, the processing for a textline involves loading
the Java Swing control with the existing text through a setText() call. Some of the
selector controls store the selection as a Docbase path. The setContent() method
drops the cabinet, since the cabinet represents a wcm_channel object which is not
present on the published Web site.

  More complex controls use processing instructions to help re-select the previous
selection. This feature is also available for custom controls. Use the Rule method
getPIData(String key), which will retrieve the value associated with the specified key.

- String getLabel()

  The text for the label is set by the label attribute on the rule. Your control can
override this call to provide further control over the value that is displayed. For
example, if the label attribute is not set, the control might supply a default. The label
is displayed on a single line in a bold font in the user interface.

- String getInstructions()

  The instructions are set by the instruction attribute on the rule. Your control can
override this call to provide further control over the value that is displayed. The
instruction text is displayed on multiple lines with a plain font.

- boolean isRequired()

  This call allows the control to determine whether information is required based on
the value of the required attribute on the Rule. The control is consulted instead of
going directly to the rule to allow the control to override this value. For example, the
checkbox control in the editor always returns false for isRequired(), since it is not
necessary to force the user to check the box. If a control returns true for isRequired() and
then returns an empty String for getContent(), the save will not proceed.

- void setFocus()

  The editor will call setFocus() on the first control that is created in the user interface
after the complete UI has been created. This allows the control to request the focus
for the appropriate component within its Swing component hierarchy.

- org.xml.sax.Attributes getAttributeList()

  This call is almost always passed directly to the method of the same name in the rule
object, for example, m_rule.getAttributeList().

- com.documentum.contenteditor.client.dataStreamMgr.Rule getRule()
This method provides access to the rule associated with the control. The rule is passed into the control in the constructor call and should be saved in an instance variable, to be returned by this call.

- `void destroy()`

  When the editor shuts down, it will call the `destroy()` method called on each control. This allows the control to remove listeners, stop threads, or do anything else necessary to allow the JVM garbage collector to clean up memory.

## Custom Control Wrapper

The CustomWidgetWrapper class wraps the custom control and insulates the rest of the editor from errors in the custom control. This layer helps in debugging the source of an error. When there are no errors, the CustomWidgetWrapper passes the EditWidget interface calls to the custom control, and the custom control appears in the user interface. In the event of an error, the CustomWidgetWrapper displays a message in the user interface indicating the problem.

One of the first problems you might encounter is that the editor cannot find or instantiate the custom control class. To insure that it can instantiate the class through reflection the wrapper performs these steps:

1. Gets the classname attribute from the `<custom>` rule (the fully qualified class name for the class).
2. Creates the `java.lang.Class` object for the specified class. Problems with the location of the jar or class files should show up here.
3. Looks for a constructor for the class with the signature described above. Both the class and the constructor should have public scope as well as the correct signature (`rule Rule,cm ConfigMgr`).
4. Creates the instance of the class. This step can fail due to an exception in the constructor for the class or inability to load a class referenced by this class.
5. Verifies that class is an instance of `JComponent`.
6. Verifies that class is an instance of `EditWidget`.
7. Casts the class to `EditWidget` and saves in an instance variable so that the calls to the `EditWidget` interface can be passed on.
8. Casts the class to `JComponent` and adds it to the CustomWidgetWrapper.

## Using a Custom Control in a Rule

You use a custom control within the `<custom>` rule element, which itself is contained within either a `<tagcontent>` or a `<tagattribute>` rule. The following example shows a sample rule for a custom editor control that displays a multiple line text area:

```xml
<tagcontent tag_name="test">
```
The only mandatory attribute within the <custom> rule element is the classname attribute. The custom editor control may recognize other attributes.

The classname attribute defines the fully qualified Java name for the component. The example shown does not have a package name. The rule defines the label attribute, which the main editor code uses to determine the bold label which precedes the control in the main interface. The rule specifies the wrap attribute, which the code uses to determine whether to turn on line wrapping for the JTextArea, and the height attribute, which sets a minimum height in pixels for the control.

An editor control may recognize any number of attributes.

The supporting code in the Rule class gives an application direct access to the DOM Node of the <rule> element that triggered the creation of an editor control. A custom editor component might choose to add elements within the rule to pass more complex configuration information.

# Writing and Packaging the Control

The following resources can help you build and package the control:
- [Using Supporting Classes](#), page 74
- [Packaging the Control](#), page 76

## Using Supporting Classes

This topic describes some of the classes that may be required by your custom control:
- ConfigMgr

Each editor component is passed a reference to a com.documentum.contenteditor.client.main.ConfigMgr object when it is instantiated. This class serves as a clearinghouse for all of the configuration information for the application. The following methods are used frequently in the product code.

- JApplet getApplet(): Returns a reference to the top level class in the application. This can be cast to the TemplateEditor class if specific methods of that class are needed.
- DocbaseClient getDocbaseClient(): Returns a reference to a DocbaseClient object, which provides methods to interact with the Docbase, for example, to run a DQL query. Since there is no DFC available within the client virtual machine, this class forwards requests to the server side code, which executes requests with a unique session.
— Font getSelectedFont(): Returns a java.awt.Font object for the current font selection. The user can change the font at the application level. This choice is saved in the user preferences and restored whenever the application is started.

— String getValue(String key): Looks up the value of a named property by searching the parameters passed to the applet on the applet tag (or the OBJECT or EMBED tag, for the plugin). It also searches the internationalized strings contained in the message catalog, if the value is not found in a parameter. If no match for the key is found, then it returns an empty string.

- Log

The Log class is a singleton implemented by the com.documentum.contenteditor.client.utilities.Log class. A custom component should write its messages to this class through the write() method. The method takes an integer argument, which indicates the type of message (Log.Err, Log.Warn, Log.Info and Log.Debug) and a String for the message. This class buffers the messages, and will display them when a debug window is activated. A button that provides access to the log messages can be activated by locating the file “contenteditor/resources/EditorUI.properties” and editing the line: contenteditor.showDebugButton=false. On Windows, the messages screen can also be viewed by holding down Ctrl and Alt and clicking the Preview button.

- DocbaseClient

The Docbase is accessed through the DocbaseClient class. A reference to the DocbaseClient is retrieved by calling ConfigMgr getDocbaseClient(). Run DQL queries by calling DocbaseClient.runQuery(String query). If you expect non-ASCII characters to be returned from the query, call DocbaseClient.runUtf8Query(String query). The methods both return a QueryResult object. This object can then be manipulated to retrieve the results of the query.

You can retrieve an object from the Docbase by calling DocbaseClient.getRawContent(String objectId).

Note: Content Server does not buffer the result set from the query it retrieves all of the rows in the result set when the query is executed. You should use care in structuring your query. This probably is not a problem for queries returning several hundred rows.

- QueryResult

Queries return results in a QueryResult object. The following methods are available in the QueryResult object:

— boolean isValid(): States whether the query ran successfully.

— int getRowCount(): Returns the number of rows in the QueryResults object.

— String getRowCol(int row, int col): Returns the results at the indicated row and column (the column matches the positions in the original select list).

— String getRowCol(int row, String columnName): Returns the results at the indicated row and column. The code looks for the column with the specified name.

— String getRow(int index, char separator): Returns a complete row, with the column values delineated by the specified separator character.
— String dump(): Returns a String (with embedded new lines) that provides
diagnostic information about the query, and prints the complete result set.

Packaging the Control

The Java Virtual Machine (JVM) in the client browser needs access to the custom control code in order to run the Web Publisher Editor applet. You can provide that access by signing applets, by installing classes on the local Web server, or by locally installing classes on the client.

The product code for Web Publisher Editor is in a signed jar, swingTe.jar, which has been signed with the Documentum certificate. The applet is signed to support access to the system clipboard file import. Since the product jar file is signed, you cannot un-jar the product jar file, add the custom control code, and then recreate the jar. This will break the signing. You can, however, re-sign the consolidated jar file with your own certificate. The jar file is located in the directory /webcomponent/library/contentxfer.

The JVM can load custom class files if they are in the correct location on the application server, relative to the page that launches Web Publisher Editor. You may want to use this approach for development. The directory structure for your custom control needs to mirror the package structure of the Web Publisher Editor jar file swingTe.jar. One way to determine where this is on the server, is to attempt to open a document that uses the custom control. You should then be able to look at the log, and determine where the client JVM tried to load your class from. (Search on the name of your control with the class extension, for example, TestWidget.class.)

You can install classes on the client machine to either the default search locations for the Sun JVM or to a location specified in the CLASSPATH variable on the local machine.

Note: Even in the Windows IE environment, the editor runs in the Sun JVM, so default search locations for the Windows Java VM won’t work.

Custom Control Messages

Your custom control may need to display error or status messages to the user. A plugin can code a custom message of up to 255 characters using the property DM_ERROR_MSG. See the sample plugin implementation in the Content Server host $DM_HOME/install/external_apps/authplugins/sampleauth directory. When Content Server encounters errors, it checks for error messages by performing dmGetProperty on this parameter. The server then passes the message embedded in the plugin error message.

Permissible error codes are:

- 0 or false in case of failure
- 1 or true in case of success

Following is an example of an error message returned by Content Server to the client layer. The message string "Sorry, I couldn’t authenticate this user!” was returned by plugin.
The plugin ID is testauth4. [DM_SESSION_E_AUTH_PLUGIN_AUTH_FAIL]error: "Authentication failed for user testuser4 with docbase test. Plugin with ID 'testauth4' returned error: Sorry, I couldn’t authenticate this user!

Custom Control Example for Multi-Line Text Area

This topic describes a simple custom control that implements a multi-line text area. The control does not try to detect whether a user has typed in characters that will make the XML of the document invalid. This example includes a simple ant configuration file. The path names in this file will need to be edited to match your environment, for example, the location of the Web Publisher files and Java directory.

TestWidget Class File

Following is the code for the custom control:

```java
/*
 * Sample control for extending the content editor
 */
import com.documentum.contenteditor.client.utilities.Log;
import com.documentum.contenteditor.client.main.ConfigMgr;
import com.documentum.contenteditor.client.guiMgr.EditWidget;
import javax.swing.JPanel;
import javax.swing.Box;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.BorderFactory;
import org.xml.sax.Attributes;
import java.awt.BorderLayout;
import java.awt.Font;

/**
 * Control that allows user to enter multiple lines of XML
 * @version 1.0
 * @author thuet
 */
public class TestWidget extends JPanel implements EditWidget {
    private ConfigMgr m_cm;
    private Rule m_rule = null;
    private JTextArea m_textArea = null;

    /**
     * Public constructor (called by TemplateEditor)
     **/
    public TestWidget(Rule editRule, ConfigMgr cm) {
        super();
        log(Log.Debug, "TestWidget constructor");
        m_cm = cm;
        m_rule = editRule;
```
Customizing Web Publisher Editor Controls

// This call lets the ConfigMgr know about this widget; this
// allows the ConfigMgr to call the widgets destroy method,
// as well as notify the widget of font changes etc.
m_cm.addWidget(this);
builtGui();
}

/**
 * Convenience method to switch message between the TemplateEditor log, or
 * System.out (which appears in the Java console)
 **/
private void log (int level, String msg)
{
    boolean useTemplateEditorLog = false;
    if (useTemplateEditorLog)
        Log.instance().write(level,msg);
    else System.out.println(msg);
}

/**
 * Whatever is necessary for memory cleanup. The goal here is to remove
 * mutual references (e.g. the ConfigMgr has a reference to this control,
 * and this control has a reference to the CM). These sometimes will
 * prevent garbage collection.
 **/
public void destroy()
{
    removeAll();
    m_rule = null;
    m_cm = null;
    m_textArea = null;
}

/**
 * Build the TestWidget
 **/
private void buildGui()
{
    log(Log.Debug,"TestWidget.buildGui()");

    // Build the text area the first time called. Some controls needed to
    // rebuild their UI, so while this test is not necessary in this
    // control, it is left in.
    if(null == m_textArea) m_textArea = new JTextArea();

    // Check the rule for whether to turn on wrapping
    if (m_rule.getBooleanAttributeWithDefault("wrap",false))
    {
        m_textArea.setLineWrap(true);
        m_textArea.setWrapStyleWord(true);
    }

    // Add the text area to the component
    setLayout(new BorderLayout());
    this.setBorder(BorderFactory.createEtchedBorder());
    this.add(new JScrollPane(m_textArea),BorderLayout.CENTER);

    // Add a strut to keep the component from being only one line
    // high. Since the TextArea is inside a JScrollPane, it won’t
    // ask for very much space. This method is better than
    // over-riding getPreferredSize() and getMinimumSize()
    int height = m_rule.getIntegerAttributeWithDefault("height",150);
    this.add(Box.createVerticalStrut(height),BorderLayout.WEST);
}
/**
 * Sets the content of an editor component. This is called
 * after the control is constructed.
 * @param theContent The data to be stored.
 * @return true, if the content was set successfully;
 * false, if the content could not be stored.
 */
public boolean setContent(String theContent)
{
    m_textArea.setText(theContent);
    return true;
} // End of method setContent

/**
 * Gets the content from the editor component. This is called by
 * a save or preview operation
 * @param type Whether the data is being retrieved for SAVE or PREVIEW.
 * @return A string containing the data from the editor component.
 */
public String getContent(int type)
{
    return m_textArea.getText();
} // End of method getContent

/**
 * Gets the label defined for the editor component. Most controls
 * just pass on the value of the "label" attribute on the rule.
 * @return A string containing the instructions that have been defined
 * for the component.
 */
public String getLabel()
{
    return m_rule.getAttributeWithDefault("label","");
}

/**
 * Gets the instructions defined for the editor component. Most controls
 * just pass on the value of the "instruction" attribute on the rule.
 * @return A string containing the instructions defined
 * for the component.
 */
public String getInstructions()
{
    return m_rule.getAttributeWithDefault("instruction","");  
}

/**
 * Gets whether data is required for this component. This example passes
 * on the "required" attribute of the rule. The main editor code tests
 * this value and the return value of getContent() to determine whether
 * all of the controls have been filled out. If a control returns "true"
 * for isRequired() and an empty String for getContent(), then the editor
 * will display a message that all of the required fields have not
 * been completed.
 * @return True, if data must be entered for the component;
 * false, if data is not required.
 */
public boolean isRequired()
{
    return m_rule.getBooleanAttributeWithDefault("required", false);
}
* Sets the focus to this control. The editor will call this method on the 
* first component in the UI that it builds.
*/
public void setFocus()
{
    if (null != m_textArea)
        m_textArea.requestFocus();
}

/**
* Sets the font for this widget.
*/
public void setDisplayFont(java.awt.Font f)
{
    // Set the font face on some of our components
    m_textArea.setFont(f);
}

/**
* Gets the attribute List
* @return AttributeList
*/
public Attributes getAttributeList()
{
    return m_rule.getAttributeList();
}

/**
* Gets the Rule
* @return Rule
*/
public Rule getRule()
{
    return m_rule;
}
} // End of class TestWidget

ANT Configuration File

A sample ant configuration file for the sample control is reproduced below. Line breaks
have been added for readability:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all" name="buildTemplateEditor">

!-- Define properties -->
!-"------------------------------------------------------------------------"->
<property name="javaRootDir" value="D:\Apps\jdk1.3"/>
<property name="teVersion" value="5.0,0,8"/>
<property name="devTop" value="F:/TDrive/WebPublisher/WebPublisher_5.0/ Applets/Content_Editor"/>
<property name="dfcDir" value="F:/TDrive/Documentum/Shared"/>
<property name="destDir" value="$(devTop)/lib/buildTemp"/>
<property name="libDir" value="$(devTop)/lib"/>
<property name="javac" value="$(javaRootDir)/bin/javac"/>
<property name="jar" value="$(javaRootDir)/bin/jar"/>
<property name="clientClassPath" value="$(destDir);$(devTop);$(libDir)/ RulesEdit.jar;$(libDir)/xerces.jar;$(libDir)/spell.jar"/>
<property name="serverClassPath" value="$(libDir)/xerces.jar;$(dfcDir)/dfc.jar; $(libDir)/wcm.jar;$(libDir)/wcmclient.jar;$(libDir)/xalan.jar"/>
```
<!--*************************************************************-->
<!-- ******************************* -->
<!-- all - the default target -->
<!-- ******************************* -->
<target name="all">
<antcall target="buildTestWidget"/>
</target>

<!-- Compile and install test widget -->
<!-- Compile source code -->
<target name="buildTestWidget">
<javac target="1.1" includes="TestWidget.java" classpath="${clientClassPath}"
debug="on" deprecation="false" destdir="${devTop}/../Custom"
<src path="${devTop}/../Custom"/>
</javac>

<!--Build a jar file -->
<jar jarfile="F:/TDrive/app/webcomponent/library/contentxfer/customWidget.jar"
basedir="${devTop}/../Custom" includes="TestWidget.class"/>
<echo message="Compile TestWidget ...Done."/>
</target>

</project>
Chapter 15

Customizing Dynamic Performers

Workflow dynamic performer configuration is described in Chapter 8, Workflow Dynamic Performers. Examples of dynamic performer init and filter classes are described here.

Note: Any custom elements that are defined within a *.performerfilters or *.performerinit element must be interpreted by the class named in the sibling <class> element. The init or filter class is given a handle to the configuration information and can use the content in any way that you design it to do so. For example, the init class AssignCurrentUserToAlias in Web Publisher interprets the values of the <role> and <aliasname> elements in the startwpwfperformers component definition file.

Following are examples of classes that implement the dynamic performer interfaces:
- Performer and Alias Initialization, page 83
- Performer and Alias Filters, page 86

Performer and Alias Initialization

The interfaces that must be implemented by custom classes to accomplish performer or alias assignment initialization are com.documentum.wp.library.workflow.IAliasPerformerInit and com.documentum.wp.library.workflow<IDynamicPerformerInit. The implementation class must be specified in the configuration file of a component that utilizes dynamic performers. The implementation class is invoked when the component is initialized. Performer and alias initialization classes are invoked in the order they are defined. If more than one implementation class is defined, a value assigned to an alias or performer by one class cannot be changed by the classes invoked subsequently.

IAliasPerformerInit

Classes implementing this interface initialize default aliases in default alias resolution. The interface defines one method with the following signature:

public void process(IDfId processId, List aliasInfo, IDfId changeSetId, List attachments, Component component, IConfigElement config)

where:
- processId: Workflow template id of the template being instantiated
Customizing Dynamic Performers

- aliasInfo: Read-only list of IAliasAssignment objects
- changeSetId: ID of the change set in the workflow or null or null id if one is not available
- attachments: Read-only list of attachments to be sent in the workflow component - parent component invoking the method
- config: Config information supplied within the class config element list.

This method can initialize with a value of IAliasAssignment instance passed in the aliasInfo parameter.

The definition of custom classes implementing this interface in a component's configuration file would look like the following:

```xml
<performer-configuration-section>
  <aliasperformerinit>
    <class name="fully.qualified.classname">
      <!-- configuration data for custom class -->
      </class>...
    <!-- other custom classes -->
  </aliasperformerinit>
</performer-configuration-section>
```

**IDynamicPerformerInit**

Classes implementing this interface accomplish initialization of default aliases in default alias resolution. The interface defines one method with the following signature:

```java
public void process(IDfId processId, List perfInfo, Component component, IDfId changeSetId, List attachments, IConfigElement config)
```

where:

- processId: Workflow template id of the template being instantiated
- perfInfo: Read-only list of IPerformerAssignment objects
- component: Parent component invoking the method
- changeSetId: ID of the change set in the workflow or null or null id if one is not available
- attachments: Read-only list of attachments to be sent in the workflow
- config: Config information supplied within the class config element

This method can initialize with a value of IPerformerAssignment instance passed in the perfInfo list.

The definition of custom classes implementing this interface in a component's configuration file would look like the following:

```xml
<performer-configuration-section>
  <aliasperformerinit>
    <class name="fully.qualified.classname">
      <!-- configuration data for custom class -->
      </class>...
    <!-- other custom classes -->
  </aliasperformerinit>
</performer-configuration-section>
```
Customizing Dynamic Performers

Performer Assignment Implementation Class

Example 15–1. Implementation of IDynamicPerformerInit
The following example implements IDynamicPerformerInit. IAliasPerformerInit would be implemented in a similar way.

```java
package example;
import com.documentum.fc.client.IDfProcess;
import com.documentum.fc.client.IDfSession;
import com.documentum.fc.common.DfException;
import com.documentum.fc.common.DfList;
import com.documentum.fc.common.IDfId;
import com.documentum.fc.common.IDfList;
import com.documentum.services.workflow.common.IPerformerAssignment;
import com.documentum.web.formext.component.Component;
import com.documentum.web.formext.config.IConfigElement;
import com.documentum.wp.library.workflow.IDynamicPerformerInit;
import java.util.Iterator;
import java.util.List;

public class DynamicPerfInit implements IDynamicPerformerInit
{

    public void process(IDfId processId, List perfInfo, Component component,
            IDfId changeSetId, List attachments, IConfigElement config)
    {
        try
        {
            IDfSession session = component.getDfSession();

            // if wf template name is *Payroll Process*
            if (((IDfProcess) session.getObject(processId)).getObjectName().
                equals("Payroll Process")
                // the current user may not know the process
                && isCurrentUserMemberOfPayroll(session))
            {
                String currentUser = session.getLoginUserName();
                for (Iterator i = perfInfo.iterator(); i.hasNext();)
                {
                    IPerformerAssignment perf = (IPerformerAssignment)i.next();
                    // pre-assign the current user to pre-process and
```
Customizing Dynamic Performers

```java
// post-process tasks
if (perf.getTargetActivityName().equals("Pre-process")
    || perf.getTargetActivityName().equals("Post-process"))
{
    DfList list = new DfList(IDfList.DF_STRING, 1);
    list appendString(currentUser);
    perf.assignPerformers(list);
}
}
}
```

```java
private boolean isCurrentUserMemberOfPayroll(IDfSession session)
{
    boolean bMemberOfPayroll = false;
    // ..
    // find out if the current user is a member of payroll group
    return bMemberOfPayroll;
}
```

## Performer and Alias Filters

The interfaces that must be implemented by custom classes to accomplish performer or alias assignment initialization are com.documentum.wp.library.workflow.IAliasPerformerFilter and com.documentum.wp.library.workflow.IDynamicPerformerFilter. The implementation class must be specified in the configuration file of a component that uses dynamic performers. The implementation class is invoked when the component is initialized. Performer and alias filter classes are invoked in the order they are defined. If more than one implementation class is defined, a value assigned by one filter class cannot be changed by the classes invoked subsequently.

Filtering takes place when the user is presented with a list of user names to choose from for the assignment. The actual implementation classes are invoked every time, right before the list is shown, to populate the list. If none of the custom classes returns any user names, the default user list is presented. The filter class should perform any required validation. For example, in the existing component implementations, if user names are expected and group names are returned, they won’t be shown in the user interface.

### IAliasPerformerFilter

Classes implementing this interface filter user names for default alias resolution. The filter performs the logic that resolves the alias to actual user names. The interface defines one method with the following signature:

```java
public Collection getUsers(
    IDfId processId, String aliasName, int aliasType, List targetActs,
```
IDfId changeSetId, List attachments, Component component, IConfigElement config)

where:

- processId: Workflow template id of the template being instantiated
- aliasName: Name of the alias selection is taking place for
- aliasType: Type of the alias selection is taking place for
- targetActs: Read-only list with names of activities the given alias is used in
- changeSetId: Id of the change set in the workflow or null or null id if one is not available
- attachments: Read-only list of attachments to be sent in the workflow
- component: Parent component invoking the method
- config: Config information supplied within the class config element This method is expected to return a collection of user names (strings) or null.

The definition of custom classes implementing this interface in a component's configuration file would look like the following:

```xml
<performer-configuration-section>
    <aliasperformerfilters>
        <class name="fully.qualified.classname">
            <!-- configuration data for custom class -->
        </class>
        <!-- other custom classes -->
    </aliasperformerfilters>
</performer-configuration-section>
```

**IDynamicPerformerFilter**

Classes implementing this interface filter user names for dynamic performer assignment. The interface defines one method with the following signature:

```java
public Collection getUsers(
    IDfId processId, String currentAct, String targetAct, int performerType, List groupNames, IDfId changeSetId, List attachments, Component component, IConfigElement config)
```

where:

- processId: Workflow template id of the template being instantiated
- currentAct: Name of current activity or null if initializing workflow
- targetAct - Activity name assignment is for
- performerType: Type of performer to be assigned
- groupNames: Read-only list with group names the assigned performers are expected be members of
- changeSetId - ID of the change set in the workflow or null or null id if one is not available
- attachments - Read-only list of attachments to be sent in the workflow
- component: Parent component invoking the method
Customizing Dynamic Performers

- config: Config information supplied within the class config element. This method is expected to return a collection of user names (strings) or null.

The definition of custom classes implementing this interface in a component’s configuration file would look like the following:

```xml
<performer-configuration-section>
  <dynamicperformerfilters>
    <class name="fully.qualified.classname">
      <!-- configuration data for custom class -->
    </class>...
    <!-- other custom classes -->
  </dynamicperformerfilters>
</performer-configuration-section>
```

**Performer Filter Implementation Class**

**Example 15-2. Implementation of IDynamicPerformerFilter**
The following example implements IDynamicPerformerFilter. IAliasPerformerFilter would be implemented in a similar way.

```java
package example;
import com.documentum.fc.client>IDfProcess;
import com.documentum.fc.client.IDfSession;
import com.documentum.fc.common.DfException;
import com.documentum.fc.common.IDfId;
import com.documentum.web.common.LocaleService;
import com.documentum.web.formext.component.Component;
import com.documentum.web.formext.config.IConfigElement;
import com.documentum.wp.library.workflow.IDynamicPerformerFilter;
import java.util.Calendar;
import java.util.Collection;
import java.util.List;
/**
* This class returns a performer list to choose from based on
* the time of the year.
*/
public class DynamicPerfFilter implements IDynamicPerformerFilter{

  /**
   * Returns possible dynamic user assignments for the given activity
   * in the workflow template. If the returned collection is empty or null,
   * the default selection list will be presented instead.
   * Note: When the class is run during workflow initiation and
   * if the change set object exists, the list of attachments may not
   * contain the same object ids as the actual contents of the change set
   * object. The list represents the desired contents of the change set
   * and is to be committed to the change set upon starting the workflow.
   * Any manipulation of the contents of the existing change
   * set object will lead to failure during workflow initiation.
   *
   * @param processId ID of the workflow template
   * @param currentAct Name of current activity or null if
   * initializing workflow
   * @param targetAct Activity name assignment is for
   * @param performerType Type of performer to be assigned
   * @param groupNames Read-only list with group names that the assigned
   * performers are expected be members of
   * @param changeSetId ID of the change set in the workflow or null or
   * null id if one is not available
   * @param attachments Read-only list of attachments in the workflow
   */
```
Customizing Dynamic Performers

public Collection getUsers(IDfId processId, String currentAct,
String targetAct, int performerType, List groupNames, IDfId changeSetId,
List attachments, Component component, IConfigElement config)
{
    List users = null;
    try
    {
        IDfSession session = component.getDfSession();
        // if wf template name is "Payroll Process"
        if (((IDfProcess) session.getObject(processId)).getObjectName().
equals("Payroll Process")
            // and the current activity is "Process"
            && currentAct != null && currentAct.equals("Process"))
            // get current month
            int month = Calendar.getInstance(LocaleService.getLocale()).get(
                Calendar.MONTH);
        switch (month)
        {
            case Calendar.JANUARY:
            case Calendar.FEBRUARY:
            case Calendar.MARCH:
                users = getQ1PayrollProcessors(session);
                break;
            case Calendar.APRIL:
            case Calendar.MAY:
            case Calendar.JUNE:
                users = getQ2PayrollProcessors(session);
                break;
            case Calendar.JULY:
            case Calendar.AUGUST:
            case Calendar.SEPTEMBER:
                users = getQ3PayrollProcessors(session);
                break;
            case Calendar.OCTOBER:
            case Calendar.NOVEMBER:
            case Calendar.DECEMBER:
                users = getQ4PayrollProcessors(session);
        }
    }
    catch (DfException e)
    {
        e.printStackTrace();
    }
    return users;
}

private List getQ1PayrollProcessors(IDfSession session)
{
    List processors = null;
    // ...
    // fill the list with names from a Q1 processors group
    // ...
    return processors;
}

private List getQ2PayrollProcessors(IDfSession session)
{
    List processors = null;
/ ... 
// fill the list with names from a Q2 processors group 
// ... 
return processors; 
}

private List getQ3PayrollProcessors(IDfSession session) 
{
List processors = null;
// ... 
// fill the list with names from a Q3 processors group 
// ... 
return processors; 
}

private List getQ4PayrollProcessors(IDfSession session) 
{
List processors = null;
// ... 
// fill the list with names from a Q4 processors group 
// ... 
return processors; 
}
Chapter 16

Requirement for Extending a Web Publisher Service

You can extend Web Publisher services to implement your business requirements. There are two steps you must perform to extend Web Publisher services:

- Your class must implement IWcmService or extends WcmService.
- You must instantiate the service using WcmAppContext.getService() in order to bind the Web Publisher application context to the service.

WCM Context objects provide the following functionality:

- Helper functions for the implementation classes that assess persistent, system or user configuration information, such as WCM system configuration or user preferences
- A scoped caching facility for the implementation classes. The scope of the cache depends on the environment in which the application is executing.

The following interfaces and classes constitute the application context objects:

- IWcmAppContext
  WCM Extension Context main interface
- WcmAppContext
  The actual implementation class for IWcmAppContext.
- WcmService
  Abstract class that all service implementations (not interfaces) are derived from

The WcmContext implementation uses a Hashtable to provide the caching facility.

If your component or class acquires Web Publisher type extensions or services, use IWcmAppContext access methods. This will ensure that IWcmAppContext is bound to the extension or services.
Web Publisher Reference

Web Publisher controls, actions, and components are described in the following reference sections:

- Chapter 17, Web Publisher Controls
- Chapter 18, Web Publisher Actions
- Chapter 19, Web Publisher Components
Web Publisher Controls

Web Publisher controls share the same attributes that are common to all controls. See the *Web Development Kit and Client Applications Development Guide* for descriptions of common attributes.

This section describes controls in the Web Publisher tag library, which is located in `APP_HOME/WEB-INF/tlds/dmwp_1_0.tld`.

For examples of using Web Publisher controls, search on the control tag name or attribute name in the Web Publisher JSP pages.

**actionbutton**

**Purpose**  
Control that launches an action from the component toolbar, page 250. Extends the WDK actionbutton control.

**Attributes**  
This control inherits the attributes of the WDK actionbutton control.

**appicon**

**Purpose**  
Control that generates an application icon. Extends the WDK image control and adds a default path to the Web Publisher application icon.

**Attributes**  
This control inherits the attributes of the WDK image control and has the following unique attributes:
availableformatter

**Purpose**
Control that formats a template to show whether it is available.

**Attributes**
This control has the following unique attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the type of object. Use the type attribute to exclude the display of specific object types. Subtypes of dm_folder are not displayed, by default.</td>
</tr>
<tr>
<td>datafield</td>
<td>Overrides the type attribute.</td>
</tr>
</tbody>
</table>

changesetheader

**Purpose**
Generates a header for a change set.

**Attributes**
This control has only objectid and name attributes.

changesetindicon

**Purpose**
Generates a change set indicator icon that indicates that an object is part of a change set and associated with a workflow. When the icon is selected, it initiates the changesetdetailscontainer action (see changesetdetailscontainer, page 113). Extends the WDK actionimage control.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>specialapp</td>
<td>Specifies whether the object is part of a manual change set. See the WP status control (status, page 102 for a description of valid values.</td>
</tr>
</tbody>
</table>

### changesetstatusmenu

**Purpose**
Generates a menu to filter change sets based on status. Extends the Web Publisher singleselectmenu, page 101 control.

**Attributes**
This control has the same attributes as the Web Publisher singleselectmenu, page 101 control.

### channellocations

**Purpose**
Renders a link to display the locations of all channels (delivery cabinets). The scope is dm_cabinet or its subtypes.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxlen</td>
<td>Specifies the cut-off length for the string</td>
</tr>
<tr>
<td>maxlennlsid</td>
<td>Localized value of maxlen (integer)</td>
</tr>
<tr>
<td>postfix</td>
<td>Defines a postfix string appended to the truncated string, for example, '…'</td>
</tr>
<tr>
<td>postfixnlsid</td>
<td>Localized value of postfix string.</td>
</tr>
</tbody>
</table>

### contentheader

**Purpose**
Generates header HTML for a template or a file.

**Attributes**
This component has only objectId and name attributes.
**dropdownlist**

**Purpose**
Generates a databound dropdown list. Extends the WDK datadropdownlist control and adds methods to get and set option labels and values.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>Specifies the property delimiter. Must be a single character. Default = ,</td>
</tr>
</tbody>
</table>

**filtervalue**

**Purpose**
Generates

**Attributes**
This component has only objectId and name attributes.

**folderheader**

**Purpose**
Generates header HTML for a folder.

**Attributes**
This component has only objectId and name attributes.

**image**

**Purpose**
Generates an image. Extends the WDK image control.

**Attributes**
This component has the same attributes as the WDK image control with the following additional attribute:
localeformatter

Purpose
Control that converts a language code to a locale value. For example, the control changes en_US to English (US).

Attributes
This control has the following unique attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the type of object. Use the type attribute to exclude the display of specific object types. Subtypes of dm_folder are not displayed, by default.</td>
</tr>
<tr>
<td>datafield</td>
<td>Overrides the type attribute.</td>
</tr>
<tr>
<td>showdefaultlocale</td>
<td>True to display the default locale as the selected locale in the locale menu</td>
</tr>
</tbody>
</table>

localemenu

Purpose
Generates a menu of locales that are supported by the application. Extends the singleselectmenu, page 101 control.

Attributes
This component has the following unique attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowfallbacks</td>
<td>True to display more generic choices for a user’s selected locale</td>
</tr>
</tbody>
</table>

menugroup

Purpose
Control that extends the WDK menugroup control and specifies the visual appearance of nested menu tags. Generates a hidden control that specifies the selected menu control.
nameuniquevalidator

Attributes

Menugroup attributes are the same as those for the WDK menugroup control.

nameuniquevalidator

Purpose

Validator control that ensures that an object name is unique in its folder.

Attributes

The attributes are the same as those for the WDK requiredfieldvalidator control.

open

Purpose

Opens a new browser child window in a portal application with a URL constructed from the component sessionping, page 233.

portlet

Purpose

Renders an HTML table.

Attributes

The unique attributes are the following:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headercssclass</td>
<td>CSS class that sets the style for the table header row</td>
</tr>
<tr>
<td>bodycssclass</td>
<td>CSS class that sets the style for the table body rows</td>
</tr>
</tbody>
</table>

preview

Purpose

Renders a preview icon that displays a preview of the selected template.

Attributes

This control has the attributes of the control image, page 98 (except the src attribute), with the following unique attributes:
### renditionpublishedname

**Purpose**
Control that displays the published name of a rendition.

**Attributes**
The unique attributes are the following:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectiddatafield</td>
<td>Required. Datafield that supplies an object ID</td>
</tr>
<tr>
<td>formatdatafield</td>
<td>Required. Datafield that supplies the object format</td>
</tr>
<tr>
<td>pagemodifierdatafield</td>
<td>Required. Datafield that supplies the page modifier, for example, 'page_modifier'.</td>
</tr>
</tbody>
</table>

### singleselectmenu

**Purpose**
Control that generates a menu consisting of singleselectmenuitem tags. Only one menu control can be selected at a time. This control extends the WDK menu control.

**Attributes**
The attributes are the same as those for the WDK menu control with the following attribute that is overridden.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies the value for the selected item in the menu</td>
</tr>
</tbody>
</table>

### singleselectmenuitem

**Purpose**
Control that generates a menu item. Must be contained within a singleselectmenu tag. Only one menu control can be selected at a time. This control extends the WDK menuitem control.
## status

### Purpose
Control that represents the lifecycle state of a Docbase object.

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies the value for the item</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionlabeldatafield</td>
<td>Datafield that supplies the version label</td>
</tr>
<tr>
<td>specialappdatafield</td>
<td>Datafield that supplies the special app status</td>
</tr>
<tr>
<td>typedatafield</td>
<td>Datafield that supplies the type</td>
</tr>
<tr>
<td>versionlabel</td>
<td>Supplies a version label. Overridden by versionlabeldatafield.</td>
</tr>
<tr>
<td>specialapp</td>
<td>Supplies the special app status, which is a combination of HTML rendition state, workflow participation, and change set membership:</td>
</tr>
<tr>
<td></td>
<td>0 not ready, no, no</td>
</tr>
<tr>
<td></td>
<td>0 in progress, no, no</td>
</tr>
<tr>
<td></td>
<td>0 ready, no, no</td>
</tr>
<tr>
<td></td>
<td>0 not ready, yes, no</td>
</tr>
<tr>
<td></td>
<td>0 in progress, yes, no</td>
</tr>
<tr>
<td></td>
<td>0 ready, yes, no</td>
</tr>
<tr>
<td></td>
<td>0 not ready, yes, yes</td>
</tr>
<tr>
<td></td>
<td>0 in progress, yes, yes</td>
</tr>
<tr>
<td></td>
<td>0 ready, yes, yes</td>
</tr>
</tbody>
</table>

| type                 | Specifies the type. Overridden by typedatafield                             |
| workflowstyle        | Specifies a CSS rule to be applied to the workflow status. For example:     |
|                      | style="COLOR: darkmagenta; TEXT-DECORATION: underline!"                     |
| workflowcssclass     | Specifies the CSS class to be applied to the workflow status                |
**statusmenu**

**Purpose**
Control the renders a status menu. Extends the *singleselectmenu, page 101* control.

**Attributes**
The attributes are the same as those for the *singleselectmenu, page 101* control.

**unsafeurlsubstitutor**

**Purpose**
Control that replaces spaces in the specified string input control value to the underscore character (_).

**Attributes**
The attributes are the same as those for the WDK requiredfieldvalidator control.

**urlsafevalidator**

**Purpose**
Control that validates whether a URL is safe HTML.

**Attributes**
The attributes are the same as those for the WDK requiredfieldvalidator control.

**versionlabelformatter**

**Purpose**
Control that removes the ID info from the version label of an approved document in the change set. Extends the WDK abstract ValueFormatter control class.

**Attributes**
This control has the following unique attributes:
### versionnumberformatter

**Purpose**
Control that displays the version number in a version label. Extends the WDK abstract ValueFormatter control class.

**Attributes**
This control has the following unique attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the type of object. Use the type attribute to exclude the display of specific object types. Subtypes of dm_folder are not displayed, by default.</td>
</tr>
<tr>
<td>datafield</td>
<td>Overrides the type attribute.</td>
</tr>
</tbody>
</table>

### webreadyicon

**Purpose**
Control that renders an icon to indicate that an object is Web ready.

**Attributes**
The attributes are the same as those for the WP status, page 102 control except that it does not have the workflow style and CSS class attributes.
Web Publisher Actions

This section describes the Web Publisher actions and their configuration attributes. Web Publisher action definitions are located in `APP_HOME/wp/config`.

See the Web Development Kit and Client Applications Development Guide for the attributes to all actions. Some of the common attributes or elements may not be used by an individual action definition. For examples of using Web Publisher actions, search on the action name or action name or attribute name in the Web Publisher JSP pages.

Many of the Web Publisher actions are organized into type-based action definition files. For example, all of the actions that can be performed on objects of type `dm_sysobject` are defined in `dm_sysobject_actions.xml`.

An action id can be defined for different object types or other qualifier, for example, the delete action has a different definition for `dm_category` and `dmi_queue_item`. The actual action definition is resolved based on the selected object type. The remainder of this section describes each action in alphabetical order. Actions with duplicate IDs are distinguished by the object type to which the action is applied. If no object type is indicated, the action applies to `dm_sysobject` with a `wpcontext` value of `wpview`. (See `context_sensitiveview_config`, page 181 for information about views.). If no parameters are listed, the action does not have any required or optional parameters.

This section provides the following information about an action where applicable:

- **Parameters**
  Actions have required or optional parameters that are defined in the action definition. These parameters are either used by the action precondition class or passed on to a component. You can call an action via URL, JavaScript, or server-side code, passing in the parameters.

- **Extends**
  An action definition can extend a base action definition and inherit all or part of the configuration of the parent definition. If an element is defined in the extended definition, that element overrides the contents of the element in the parent definition. For example, if the parent specifies an `objectId` parameter and the extended action has an empty parameters element, then no parameters are required for the child action.

- **Elements**
  A component may define configurable elements that are specific to the component. When the action launches a container, the action definition specifies a `<container>` element. The definition can also specify the component to be displayed in the container, as the `<component>` element.
abortfailedautowpworkflwtask

**Purpose**
Aborts a failed automatic workflow task by launching the component wptaskmgrabortworkflow, page 283 in the wptaskcomponentcontainer.

**Parameters**
The parameters are the same as those for the webcomponent layer’s delegateworkflowtask action, except that isDelegable is not a parameter.

abortworkflow

**Purpose**
Aborts a workflow by launching the component abortwpworkflow, page 156 in combocontainer.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the workflow.</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflowRuntimeState</td>
<td>Required. State of the workflow. 0=dormant</td>
</tr>
</tbody>
</table>

acceptworkflwtask

**Purpose**
Accepts a workflow task for the user.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the task.</th>
</tr>
</thead>
<tbody>
<tr>
<td>canAcquireTask</td>
<td>Boolean flag that indicates that the task is in a state such that it can be acquired (not paused, already acquired, or finished)</td>
</tr>
<tr>
<td>isGroupOwned</td>
<td>Boolean flag that indicates that the task is assigned to a group of which the user is a member</td>
</tr>
</tbody>
</table>
aclbrowse

**Purpose**
Allows the administrator to browse a list of ACLs in the Docbase by launching the webcomponent container aclobjectlocator.

**Parameters**
This action has no defined parameters.

add_manual_link

**Purpose**
Launches the add_manual_link, page 159 component to allow the user to link an object to the selected object. The user selects from a predefined list of link types when they make the link. Only one link of a specific type can be made between two objects. The user can make a link of a different type between the same two objects. This action is called from the toolbar, page 250 UI.

**Parameters**
This action has no defined parameters.

add_manual_link_type

**Purpose**
Launches the add_manual_link_type, page 159 component to allow the administrator to configure manual link types in the application. The manual link type that is added must correspond to a dm_relation_type in the Docbase. This action is called from the toolbar, page 250 UI.

**Parameters**
This action has no defined parameters.

add_translation

**Purpose**
Launches the component add_translation and addtranslationcontainer, page 159 in the addtranslationcontainer to add a translation. Operates on objects of type dm_document or wcm_translation and their subtypes.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the translation.</td>
</tr>
<tr>
<td>parentId</td>
<td>ID of the translation</td>
</tr>
<tr>
<td>localeDisplay</td>
<td>Locale to display for the translation</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the translation document</td>
</tr>
<tr>
<td>folderId</td>
<td>Required. Docbase folder to which the translation will be added.</td>
</tr>
</tbody>
</table>

#### addedition

**Purpose**
Launches the component `addedition` and container, page 160 in `addeditioncontainer` to add an archived Web site edition.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the channel (delivery cabinet) to which the object will be added.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the object</td>
</tr>
</tbody>
</table>

#### addlocale

**Purpose**
Adds a locale to the Web Publisher application by launching the component `addlocale` and container, page 158 in `addlocalecontainer`.

**Parameters**

This action has no defined parameters.

#### addnewrule

**Purpose**
Adds a new rules file in a folder or cabinet by launching the component `addnewrule` and addnewruledialogcontainer, page 160 in `addnewruledialogcontainer`. The user must have `write_permit` access to the destination folder or cabinet (valid values: `DELETE_PERMIT` | `WRITE_PERMIT` | `VERSION_PERMIT` | `RELATE_PERMIT` | `READ_PERMIT` | `BROWSE_PERMIT` | `NONE_PERMIT`). This action operates on `dm_folder` objects.
**Parameters**

<table>
<thead>
<tr>
<th>object Id</th>
<th>Required. ID of the folder that will contain the new rules file.</th>
</tr>
</thead>
</table>

**addrule**

**Purpose**

Launches the component addrule and container, page 161 in the combocontainer. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected object.</th>
</tr>
</thead>
</table>

**addtochangeset**

**Purpose**

Launches the component addtochangeset, page 161 in the wpmultiargdialogcontainer to add a file to a change set. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. IDs of objects in the change set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>Required. ID of the folder that contains the change set</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Required. Name of the owner of locked documents</td>
</tr>
<tr>
<td>specialApp</td>
<td>Required. See status, page 102.</td>
</tr>
</tbody>
</table>

**addtoclipboard**

**Purpose**

Adds an object to the clipboard.

**Parameters**

This action has the same parameters as the webcomponent layer’s addtoclipboard, with the following additional parameter:
### listview

| Name of WP view. The view wpcategoryview cannot be added to the clipboard. |

#### Elements

| <precondition>.<role> | Value: Role that can perform move, copy, or link operations. Default = contributor |

#### Usage

To disable clipboard actions for a particular object type, use the notdefined attribute. For example:

```xml
<scope type="my_custom_type">
  <action id="addtoclipboard" notdefined="true">
  </action>
</scope>
```

### alerts

#### Purpose

Launches the component alerts, page 164 to display Content Server system alerts. This action is called from the component toolbar, page 250.

#### Parameters

This action has no defined parameters.

### aliassetbrowse

#### Purpose

Launches the webcomponent container aliassetobjectlocator so that the user can search for an alias set. Operates on objects of type dm_sysobject or its subtypes.

#### Parameters

This action has no defined parameters.

### applauncher

#### Purpose

Jumps to the component wp_applauncher, page 264. Operates on objects of type dm_sysobject or its subtypes.
applypresentation

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the selected object</td>
</tr>
<tr>
<td>command</td>
<td>Command that initializes and launches the external viewing application on the client</td>
</tr>
</tbody>
</table>

**Purpose**

Launches the component `applypresentationcomponent and container`, page 165 in `applypresentationcontainer`. Operates on objects of type `dm_document` or its subtypes.

associateworkflow

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Workflow ID to be associated with the selected object</td>
</tr>
</tbody>
</table>

**Purpose**

Launches the component `associateworkflow`, page 165 in the associations container. Operates on objects of type `dm_document` or its subtypes.

associations

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID for the selected object</td>
</tr>
</tbody>
</table>

**Purpose**

Launches the component `thumbnailpreview`, page 250 in the associations container.
attributes

Purpose  Displays the attributes of a selected object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID for the object to be published</td>
</tr>
<tr>
<td>enableShowAll</td>
<td>Set to true to enable the <strong>Show All</strong> button, which allows the user to see all properties for the object</td>
</tr>
<tr>
<td>folderPath</td>
<td>Full path to object in the Docbase</td>
</tr>
</tbody>
</table>

categories

Purpose  Launches the component **categories**, page 169 in the properties container to display categories for the application and allow the user to add or remove categories. This action is called from the component **menubar**, page 209.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format of the selected object</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the selected object</td>
</tr>
</tbody>
</table>

categorylocaleslist

Purpose  Launches the component **categorylocaleslist**, page 171 in the properties container. Displays titles, translations, and language codes for a category.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format of the selected object</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the selected object</td>
</tr>
</tbody>
</table>
changesetdetailscontainer

**Purpose**
Launches the component changesetdetailscontainer, page 172 to display details for a change set.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the change set</td>
</tr>
<tr>
<td>docId</td>
<td>ID of a document in a change set</td>
</tr>
</tbody>
</table>

changesetremovefiles

**Purpose**
Removes files from a change set. This action is called from the UI of changeset_files_list, page 172.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID for the object to be removed from its associated change set</td>
</tr>
</tbody>
</table>

changesetremovesupportingfiles

**Purpose**
Removes supporting files from a change set.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of file to be removed from the change set</td>
</tr>
<tr>
<td>csFolderId</td>
<td>ID of the folder that contains the change set</td>
</tr>
<tr>
<td>type</td>
<td>Not used</td>
</tr>
</tbody>
</table>

clearallalerts

**Purpose**
Clears all Server system alerts. This action is called from the component toolbar, page 250.
**completefailedautoworkflowtask**

**Purpose**
Completes a failed automatic workflow task by launching the webcomponent layer's completefailedautoworkflowtask component in the wptaskcomponentcontainer.

**Parameters**
The parameters are the same as those for the webcomponent layer’s delegateworkflowtask action, except that isDelegable is not a parameter.

**copynonwcm**

**Purpose**
Launches the component copynonwcm, page 183 in the wpclipboardcontainer. Copies non-Web Publisher objects.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. Target folder ID.</th>
</tr>
</thead>
</table>

**delegateworkflowtask**

**Purpose**
Delegates a task in a workflow by launching the webcomponent layer’s delegateworkflowtask in wptaskcomponentcontainer.

**Parameters**
The parameters are the same as finishwpworkflowtask, page 192 except that isManual is replaced by isDelegable:

<table>
<thead>
<tr>
<th>isDelegable</th>
<th>Set to true to specify that the task can be delegated</th>
</tr>
</thead>
</table>
delete

**Purpose**
Deletes the object from the Docbase by launching the component delete and container, page 186 in the combocontainer for users with permission of delete_permit (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). This action is defined for dm_document, wcm_changeset, wcm_channel, and wcm_channel_fld objects. The definition for folder and cabinets (dm_folder and dm_cabinet) is the same except that there is no precondition.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. The ID of the object to be deleted.</td>
</tr>
<tr>
<td>folderId</td>
<td>Required for all except wcm_changeset. The ID of the folder in which the object is located.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Optional for all except wcm_changeset. The name of the user who has locked the object.</td>
</tr>
<tr>
<td>ownerName</td>
<td>Optional for all except wcm_changeset. The name of the user who owns the object to be deleted.</td>
</tr>
</tbody>
</table>

**delete (dm_category)**

**Purpose**
Launches the component delete and container, page 186 in the combocontainer to delete a category.

**Parameters**
This action has the same parameters as delete, page 115.

**delete_alerts**

**Purpose**
Launches an alert to advise the user of the result of a delete operation.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stampId</td>
<td>Required. ID of the alert.</td>
</tr>
</tbody>
</table>
delete (locale)

**Purpose**
Deletes a locale (wcm_locale object) from the application by calling the component delete_locale, page 186 in the addlocalecontainer.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the locale to be deleted.</td>
</tr>
<tr>
<td>default_locale</td>
<td>Required. ID of the default locale.</td>
</tr>
<tr>
<td>objectName</td>
<td>Required. Name of the locale to be deleted.</td>
</tr>
</tbody>
</table>

delete_translation

**Purpose**
Extends delete, page 115 and does not add or override any methods. Deletes a translation of an object of type dm_document or a subtype. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be deleted.</td>
</tr>
<tr>
<td>folderId</td>
<td>Required. ID of the folder that contains the object to be deleted.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object to be deleted.</td>
</tr>
</tbody>
</table>

deleterenditions

**Purpose**
Deletes the selected rendition. The scope is dmr_content objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the change set to be deleted.</td>
</tr>
<tr>
<td>contentType</td>
<td>Required. Type of the selected object.</td>
</tr>
<tr>
<td>pageModifier</td>
<td>User-defined string that is added to the filename for a rendition. Indicates the file format.</td>
</tr>
<tr>
<td>page</td>
<td>Position of the content in the objects that contain it</td>
</tr>
</tbody>
</table>
**deleteversion**

**Purpose**  
Deletes a selected version of an object of type dm_document or a subtype. Operates on objects of type dm_document or its subtypes.

**Parameters**  
This component has the same parameters as delete, page 115.

**difflocal**

**Purpose**  
Launches the component wp_diffcontent and container, page 266 in the wp_diffcontent_container. The user must have read_permit access to the documents (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). Operates on objects of type dm_document or its subtypes.

**Parameters**  
This action has the same parameters as diffversions, page 117.

**diffversions**

**Purpose**  
Launches the component wp_diffcontent and container, page 266 in the wp_diffcontent_container. The user must have read_permit access to the documents (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
**drlview**

**Purpose**
Opens a DRL for viewing. This action is defined for dmi_queue_item scope. This action is called by the View button event handler in the WDK layer's drl component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the DRL to be viewed</td>
</tr>
</tbody>
</table>

**editfile**

**Purpose**
Launches the component wp_edit and container, page 267 in the wp_editcontainer. The user who edits the document must have write_permit access to the document (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be checked out.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Name of the user who has checked out the document</td>
</tr>
<tr>
<td>contentSize</td>
<td>Size of the document in bytes</td>
</tr>
<tr>
<td>isReference</td>
<td>Boolean flag that specifies whether a document is a reference object</td>
</tr>
</tbody>
</table>

**expirenow**

**Purpose**
Launches the component expirenow and container, page 189 in expirenowcontainer. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected document.</td>
</tr>
</tbody>
</table>
export

Purpose
Exports an object to the user’s local file system by launching the component `wp_export` and container, page 267 in the `wp_exportcontainer`. The user who edits the document must have read_permit access to the document (valid values: `DELETE_PERMIT`, `WRITE_PERMIT`, `VERSION_PERMIT`, `RELATE_PERMIT`, `READ_PERMIT`, `BROWSE_PERMIT`, `NONE_PERMIT`).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the edition to be exported.</td>
</tr>
<tr>
<td>objectName</td>
<td>Name of the edition.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the edition.</td>
</tr>
</tbody>
</table>

exportedition

Purpose
Exports an edition (archived Active Web site) for browsing by launching the component `exportedition`, page 190 in the `additioncontainer`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the edition to be exported.</td>
</tr>
<tr>
<td>objectName</td>
<td>Name of the edition.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the edition.</td>
</tr>
</tbody>
</table>

exportrendition

Purpose
Exports a rendition to the local file system and launches the viewing application for the selected rendition when the user has read_permit access to the object. The action launches the `wp_export_rendition` and container, page 268 component in `wp_export_renditioncontainer`. 
finishwpworkflowtask

Purpose
Forwards a workflow task by launching finishwpworkflowtask, page 192 in the wptaskcomponentcontainer container.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be viewed.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
</tbody>
</table>

forwardwpworkflowtask

Purpose
Finishes a workflow task by launching the WP layer’s forwardwpworkflowtask in the wptaskcomponentcontainer container.

Parameters

The parameters are the same as those for the webcomponent layer’s finishworkflowtask action, except that the parameter isFailedAutoTask is replaced by the parameter hasForwardPaths.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasForwardPaths</td>
<td>Boolean flag that specifies that the workflow has forward paths</td>
</tr>
</tbody>
</table>

history

Purpose
Launches the webcomponent layer’s history component in the container properties, page 221 to display the history of the selected object.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format of the selected object</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the selected object</td>
</tr>
</tbody>
</table>

### history (wp task)

#### Purpose

Launches the container specified as the value of the `<wptaskmanagerId>` element in the definition, to display a workflow history. This action is called by the component `menubar`, page 209. The type scope of `dm_wp_task` is defined within the action class; it is not a Docbase type.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>wptaskmanagerId</td>
<td>Required. ID of the workflow taskmanager component.</td>
</tr>
</tbody>
</table>

#### Elements

The following argument is defined for the `<execution>.<arguments>.<argument>` element.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>containertabid</td>
<td>Specifies the component tab to be displayed by default in the taskmanagercontainer.</td>
</tr>
</tbody>
</table>

### import

#### Purpose

Imports an object into a folder, cabinet, or channel. In components using the `wpcontext` “wpview”, the import action launches the component `wp_import_template`, page 271 in the `wp_importcontainer`. In components using the `wpcontext` “wpinstructions”, the import action launches the component `wp_import_instructions` and container, page 270 in the `wpinstructionsimportcontainer`. If the is importing to a supporting docs folder, the action launches the webcomponent layer’s import component in its container. The user must have `write_permit` access to the destination folder or cabinet (valid values: `DELETE_PERMIT` | `WRITE_PERMIT` | `VERSION_PERMIT` | `RELATE_PERMIT` | `READ_PERMIT` | `BROWSE_PERMIT` | `NONE_PERMIT`). The import action operates on `dm_folder`, `wcm_channel`, and `wcm_channel_fld` objects.
importrendition

### Purpose
Launches the importrenditionand container, page 200 component in the importrenditioncontainer to import a rendition into the Docbase. The user must have version_permit access to the object.

### Parameters
<table>
<thead>
<tr>
<th>object Id</th>
<th>Required. ID of the folder that will contain the import.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownerName</td>
<td>Name of the owner of the destination folder or cabinet.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Specifies the path to the folder that serves as the base for navigation.</td>
</tr>
<tr>
<td></td>
<td>This path must be a complete path from the Docbase root, for example,</td>
</tr>
<tr>
<td></td>
<td>/Doc%20Library/subfolder</td>
</tr>
</tbody>
</table>

importrenditionbutton

### Purpose
Extends the DAM importrendition action definition and overrides the parent scope of dm_sysobject.

### Parameters
| object Id         | Required. ID of the selected object                                      |

import

### Purpose
Imports content into a category by launching the component wp_import_content_template, page 269 in the wp_importcontainer. Operates on wcm_category objects.
lifecyclebrowse

(category folders). This action requires the user to have write_permit access to the target category folder (valid values: DELETE_PERMIT \ WRITE_PERMIT \ VERSION_PERMIT \ RELATE_PERMIT \ READ_PERMIT \ BROWSE_PERMIT \ NONE_PERMIT).

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the target category folder.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownerName</td>
<td>Name of the owner of the target category folder.</td>
</tr>
</tbody>
</table>

lifecyclebrowse

Purpose

Launches the container wplifecyclelocator and container , page 276 so that the user can select a lifecycle.

Parameters

This action has no defined parameters.

list_manual_link

Purpose

Launches the list_manual_link_component, page 205 component using jump navigation to display a list of objects that the user has manually linked to the selected object. For each object listed, the system will display the name of the linked object, the link type, whether notifications are enabled, and the date the link was established.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderPath</td>
<td>Folder path of the selected object. This path must be a complete path from the Docbase root, for example, \Documentation\Library\subfolder</td>
</tr>
</tbody>
</table>
list_related_content

Purpose
Launches the list_related_content, page 206 component using jump navigation to display a list of objects that are related to the selected objects. This action is called from the menubar, page 209 component UI.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Folder path of the selected object. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
<tr>
<td>relationType</td>
<td>Value of the object relation_name attribute</td>
</tr>
</tbody>
</table>

locationsview

Purpose
Displays a list of locations for selected objects. This action is called by the UI of menubar, page 209.

Parameters
This action has no defined parameters.

makeavailable

Purpose
Makes a template available if the user is the owner of the template. Operates on objects of type dm_document or its subtypes.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected template.</td>
</tr>
<tr>
<td>versionLabel</td>
<td>Required. Label for the object version.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Required. Name of owner who has locked the template.</td>
</tr>
</tbody>
</table>
**makeunavailable**

**Purpose**
Makes a template unavailable if the user is the owner of the template. Operates on objects of type dm_document or its subtypes.

**Parameters**
The parameters are the same as those for `makeavailable`, page 124.

**makewftemplate_available**

**Purpose**
Makes a workflow template available if the user is the owner of the template. Operates on objects of type dm_process or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected template.</td>
</tr>
<tr>
<td>objectName</td>
<td>Required. Name of the template.</td>
</tr>
<tr>
<td>make_available</td>
<td>Required. Boolean flag that specifies that the template should be made available.</td>
</tr>
</tbody>
</table>

**makewftemplate_unavailable**

**Purpose**
Makes a workflow template unavailable if the user is the owner of the template. Operates on objects of type dm_process or its subtypes.

**Parameters**
This action has the same parameters as `makewftemplate_available`, page 125.

**menuitemtransform**

**Purpose**
Launches the DAM component chooseprofiletransform component in transformcontainer. Transforms objects of type dm_document or its subtypes if the user has write permit access.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of object to be transformed</td>
</tr>
<tr>
<td>pageComp</td>
<td>Component page on which the menu item is displayed. If a value is supplied, the menu item will not be displayed.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder that contains the object</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
</tbody>
</table>

**menuitemtransform_button**

**Purpose**

Extends menuitemtransform, page 125 and does not add any functionality. Called by a transform button.

**Parameters**

The parameters are inherited from .

**newcategory**

**Purpose**

Creates a new category by launching the component newcategory and container, page 211 in the newcategorycontainer. This action is called from the UI of menubar, page 209 and category_streamline, page 170.

**Parameters**

This action has no parameters.

**newcategoryfolder**

**Purpose**

Creates a new category and its corresponding folder by launching the component newcategoryfolder, page 212 in the newcategoryfoldercontainer. The user must have write_permit access to the parent folder or cabinet (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). The action creates a wcm_category object.
**newchangeset**

**Purpose**  
Creates a new change set by launching the component `newchangeset, page 213` in the WDK layer’s `dialogcontainer`.

**Parameters**  
This action has no defined parameters.

**newchannel**

**Purpose**  
Launches the component `newchannel, page 213` to create a new channel (delivery cabinet) in `newchannelcontainer`. The scope is `dm_docbase`.

**Parameters**  
This action has no parameters.

**newchannelfolder**

**Purpose**  
Launches the component `newchannelfolder, page 214` in the `newchannelfoldercontainer`, which creates a new folder in the Site. The user must have `write_permit` on the delivery cabinet (valid values: `DELETE_PERMIT` | `WRITE_PERMIT` | `VERSION_PERMIT` | `RELATE_PERMIT` | `READ_PERMIT` | `BROWSE_PERMIT` | `NONE_PERMIT`).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objectId</code></td>
<td>Required. The ID of the folder that will contain the new folder.</td>
</tr>
<tr>
<td><code>ownerName</code></td>
<td>Name of the owner of the folder in which the folder will be created</td>
</tr>
</tbody>
</table>
newcontent

**Purpose**

Creates new content by launching the component *newcontent and container, page 215* in *newcontentcontainer*.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID for the object to be created</td>
</tr>
</tbody>
</table>

newcontenttemplate

**Purpose**

Extends the component *newcontent and container, page 215* to create content from a template.

**Parameters**

Parameters are inherited from *newcontent and container, page 215*.

newtaxonomy

**Purpose**

Creates a new taxonomy within a category by launching the component *newtaxonomy, page 217* in *newtaxonomycontainer*. This action is called from the component UI of *menubar, page 209* and *category_streamline, page 170*.

**Parameters**

This action has no defined parameters.

permissions

**Purpose**

Launches the webcomponent layer’s permissions component in the *properties, page 221* container. For objects of type *wcm_channel_fld* and *wcm_channel*, the component *readonlypermissions, page 226* is launched.
### powerpromote

**Purpose**  
Launches the component `powerpromote` and `container`, page 218 in `powerpromotecontainer` to promote a document or change set directly to the Approved state.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the selected object</td>
</tr>
</tbody>
</table>

### presentations

**Purpose**  
Launches the component `presentations`, page 219 in `associations container`. Operates on objects of type `dm_document` or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>associateVersion</td>
<td>Set to true to associate a presentation with the current version. If false (set by the versions component for a non-current version), the presentation cannot be associated.</td>
</tr>
</tbody>
</table>

### promote

**Purpose**  
Launches the promote component in `promotecontainer` to promote a document or change set. Operates on objects of type `dm_document` or `wcm_change_set` and their subtypes.
properties

Purpose
Launches the properties, page 221 component in the appropriate container based on the content type. For example, the supportingtemplateproperties component is launched if the content type is rule, layout, or publishing template.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected document.</td>
</tr>
<tr>
<td>folderID</td>
<td>Specifies the ID of the folder that contains the object</td>
</tr>
<tr>
<td>enableShowAll</td>
<td>Set to true to enable the Show All button, which allows the user to see all properties for the object</td>
</tr>
</tbody>
</table>

properties (dmr_content)

Purpose
Disables properties if media server is not available. The scope is dmr_content objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object for which properties will be displayed</td>
</tr>
<tr>
<td>enableShowAll</td>
<td>Set to false to disable the Show All button, so the user will not see all properties for the object</td>
</tr>
<tr>
<td>contentType</td>
<td>Set to required=&quot;false&quot;</td>
</tr>
<tr>
<td>pageModifier</td>
<td>Set to required=&quot;false&quot;</td>
</tr>
<tr>
<td>pageNumber</td>
<td>Set to required=&quot;false&quot;</td>
</tr>
</tbody>
</table>
protected_assets

**Purpose**
Launches the component protectedassets, page 224 in protectedassetscontainer to display the Web cabinet’s protected assets. This action is scoped to wcm_channel. The action is called in menubar, page 209.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the Web cabinet</td>
</tr>
<tr>
<td>folderID</td>
<td>Specifies the ID of the protected assets folder</td>
</tr>
<tr>
<td>enableShowAll</td>
<td>Set to true to enable the Show All button, which allows the user to see all protected assets</td>
</tr>
</tbody>
</table>

publish

**Purpose**
Launches the component publish and container, page 225 in publishcontainer to publish a document, channel, or channel folder. This action is defined for dm_document, wcm_channel, and wcm_channel fld objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID for the object to be published</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the object</td>
</tr>
<tr>
<td>folderPath</td>
<td>Full path to folder that contains the object.</td>
</tr>
</tbody>
</table>

publishtaxonomy

**Purpose**
Publishes a taxonomy. This action is called from the components menubar, page 209 and category_streamline, page 170.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID for the taxonomy to be published</td>
</tr>
<tr>
<td>a_status</td>
<td>Online status. Valid values: online</td>
</tr>
</tbody>
</table>
publish

**Purpose**
Launches the component publishing, page 225 in the container properties, page 221 to save or update publishing properties for the object. This action is defined for dm_document and wcm_channel objects. The action is called from the component menubar, page 209.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of object to be published.</td>
</tr>
</tbody>
</table>

rejectwpworkflowtask

**Purpose**
Rejects a workflow task by launching the component rejectwpworkflowtask, page 227 in the wptaskcomponentcontainer container.

**Parameters**
The parameters are the same as those for the webcomponent layer’s finishworkflowtask action, except that the parameter isFailedAutoTask is replaced by the parameter hasRejectPaths.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasRejectPaths</td>
<td>Boolean flag that specifies that the workflow has reject paths</td>
</tr>
</tbody>
</table>

reinstate

**Purpose**
Launches the component reinstate, page 227 to make a previous version of a file the current version. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object version to reinstate (each object has a unique ID for each of its versions).</td>
</tr>
<tr>
<td>versionLabel</td>
<td>Label describing the source object version. Must not be CURRENT.</td>
</tr>
</tbody>
</table>
**removeedition**

**Purpose**  
Removes an edition by launching the component `removeedition`, page 227.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the edition to be removed.</td>
</tr>
<tr>
<td>objectName</td>
<td>Required. Name of the edition.</td>
</tr>
</tbody>
</table>

**removefilefromchangeset_in_startwf**

**Purpose**  
Removes a file from a change set. Operates on objects of type `startwpwfattachment_scope` or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the file to be removed.</td>
</tr>
</tbody>
</table>

**remove_manual_link**

**Purpose**  
Remove all manual links to the current object. This action is called from the `toolbar`, page 250 UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Array of IDs of objects to unlink from the current object.</td>
</tr>
<tr>
<td>issystem</td>
<td>Required. If true (system link), the link will not be removed.</td>
</tr>
</tbody>
</table>

**remove_manual_link_type**

**Purpose**  
Removes manual link types in the application. The corresponding `dm_relation_type` object is not removed from the Docbase. This action is called from the `toolbar`, page 250 UI.
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Array of IDs of link types to remove from the application.</td>
</tr>
<tr>
<td>isSystem</td>
<td>Required. If true (system link), the link type will not be removed.</td>
</tr>
</tbody>
</table>

---

**removeprotected lifecycles, workflows, categories, ctemplates, and stemplates**

**Purpose**
Removes protected assets from the Web cabinet. These actions are called from the UI of the protectedassets, page 224 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the file to be removed.</td>
</tr>
<tr>
<td>type</td>
<td>Required. Type of the object to be removed.</td>
</tr>
<tr>
<td>objectName</td>
<td>Name of the object to be removed</td>
</tr>
</tbody>
</table>

---

**removesupportingfilefromchangeset_in_starwf**

**Purpose**
Removes supporting files from a change set. Operates on objects of type startwpwfsupportingfiles_scope or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the files to be removed.</td>
</tr>
</tbody>
</table>

---

**renditions**

**Purpose**
Launches the component renditions, page 228 in wpnavigationcontainer.
### repeatwpworkflowtask

**Purpose**
Repeats a task in a workflow by launching the component `repeatwpworkflowtask`, page 228 in the wptaskcomponentcontainer.

**Parameters**
The parameters are the same as those for the webcomponent layer’s `delegateworkflowtask` action, except that `isDelegable` is replaced by `isRepeatable`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system).</td>
</tr>
<tr>
<td>isRepeatable</td>
<td>Boolean flag that specifies whether a task can be repeated.</td>
</tr>
</tbody>
</table>

### request_translation

**Purpose**
Launches the `request_translation`, page 229 component in the WDK layer’s `dialogcontainer`. Operates on objects of type `dm_document` or `wcm_translations` and their subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the document to which a translation will be attached.</td>
</tr>
<tr>
<td>parentId</td>
<td>ID of the translation</td>
</tr>
<tr>
<td>localeDisplay</td>
<td>Locale to display for the translation</td>
</tr>
<tr>
<td>localeId</td>
<td>ID of the translation locale</td>
</tr>
</tbody>
</table>

### rerunfailedautoworkflowtask

**Purpose**
Reruns a failed automatic workflow task by launching the webcomponent layer’s `rerunfailedautoworkflowtask` component in the wptaskcomponentcontainer.
Parameters

The parameters are the same as those for the webcomponent layer’s delegateworkflowtask action, except that isDelegable is not a parameter.

restartchangeset

Purpose

Restarts a change set. This action acts only on wcm_change_set objects.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the change set to be restarted.</th>
</tr>
</thead>
</table>

rmpresentation

Purpose

This action is handled by the Presentations class, which removes the presentation from the Docboase.

rmrule

Purpose

Removes a rule from the application. The action is performed by the Rules class. The action is called by the UI of the component rules, page 230.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected rule.</th>
</tr>
</thead>
</table>

rmtaxonomywebcabinet

Purpose

Removes a taxonomy from a Web cabinet. The action is performed by the TaxonomyWebCabinetsList class. The action is called by the component UIs of taxonomies, page 248 and taxonomywebcabinetslist, page 248.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected taxonomy.</th>
</tr>
</thead>
</table>
rules

Purpose
Displays a rules file by launching the component rules, page 230 in the associations container. Operates on objects of type dm_document or its subtypes.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected rules file.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
</tbody>
</table>

sendlocator

Purpose
Sends a document locator through email by calling the webcomponent layer’s sendlocator component in its container.

Parameters
This action inherits the parameters of the webcomponent layer’s sendlocator component.

setdefault_locale

Purpose
Sets the default locale to be used when a locale has not been selected by the user, for example, at first access to the application.

Parameters
This action has the same parameters as the action delete (locale), page 116.

showpreferences

Purpose
Launches the component general_preferences, page 193 in the container preferences, page 219. The action is called by the UI of inboxclassic, page 201 and inboxstreamline, page 201.
showtaskstatus

Purpose
Launches the component taskstatus, page 248, which allows the user to view the status of tasks. The action is called by the UI of statusbar, page 242.

Parameters
This action has no defined parameters.

code:showtaskstatus

siteoverview

Purpose
Launches the component siteoverview, page 234, which allows the user to view the publishing configurations, protected assets, and archived editions, and turn off or on the in-context editor.

Parameters
This action has no defined parameters.

code:siteoverview

standardview

Purpose
Displays a file list in the standard view. The action is called by the UI of menubar, page 209.

Parameters
This action has no defined parameters.

code:standardview

startworkflownotemplate

Purpose
Launches the container startwpwftemplateloctorcontainergenericstreamline, page 240 to start a workflow for a new document that does not have a template. Displays a streamline view UI.
Parameters

This action has no defined parameters.

---

**startwpworkflownotemplate**

**Purpose**
Launches the container startwpwftemplatelocatorcontainerclassic, page 239 to start a workflow for a new document or change set that does not have a template. Operates on objects of type dm_document or wcm_change_set and their subtypes. Displays a classic view UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>IDs of documents to be attached in the workflow, or a single change set ID</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Name of the user who has checked out the document. (Not a defined parameter for change set objects.)</td>
</tr>
<tr>
<td>specialApp</td>
<td>See status, page 102.</td>
</tr>
</tbody>
</table>

---

**startwpworkflownotemplatecs**

**Purpose**
Launches the container startwpwftemplatelocatorcontainerclassic, page 239 to start a workflow for a new change set that does not have a template. Displays a classic view UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>IDs of documents to be attached in the workflow, or a single change set ID</td>
</tr>
<tr>
<td>specialApp</td>
<td>See status, page 102.</td>
</tr>
</tbody>
</table>

---

**Elements**

The following attribute is defined for the `<argument>` element within `<execution>`.<arguments>:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachmentMode</td>
<td>Specifies whether the attachment is an existing change set. Valid values = existingchangeset</td>
</tr>
</tbody>
</table>
**startwpworkflownotemplategeneric**

**Purpose**
Launches the container `startwpwftemplatelocatorcontainerclassic`, page 239 to start a workflow with no template selected. Displays a classic view UI.

**Parameters**
This action has no defined parameters.

**startwpworkflownotemplategenericstreamline**

**Purpose**
Launches the container `startwpwftemplatelocatorcontainerstreamline`, page 240 to start a workflow with no template selected. Displays a streamline view UI.

**Parameters**
This action has no defined parameters.

**startwpworkflowclassic**

**Purpose**
Starts a workflow (classic view) by launching the `startwpwfcontainerclassic`, page 235 container. This action is launched from the OK button in the `startwpwftemplatelocatorcontainerclassic`, page 239 UI after the user has selected a workflow template to launch. This action operates on `dm_process` objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the workflow process.</td>
</tr>
<tr>
<td>attachmentId</td>
<td>ID of the workflow attachment</td>
</tr>
<tr>
<td>changesetId</td>
<td>ID of the change set to which the workflow belongs, if any</td>
</tr>
<tr>
<td>skipWorkflowReadyCheck</td>
<td>Boolean parameter. Set to true to launch a workflow without checking readiness.</td>
</tr>
</tbody>
</table>
**startwpworkflownotemplatestreamline**

**Purpose**
Launches the container `startwpwftemplateloctorcontainerstreamline`, page 240 for a new document or change set. Operates on objects of type `dm_document` or `wcm_change_set` or their subtypes.

**Parameters**
This action has the same parameters as `startwpworkflownotemplatecs`, page 139.

**Elements**
This action has the same configurable element as `startwpworkflownotemplatecs`, page 139.

| attachmentMode | Specifies whether the attachment is an existing change set. Valid values = existingchangeset | null |

**startwpworkflowstreamline**

**Purpose**
Starts a workflow (streamline view) by launching the container `startwpwfcontainerstreamline`, page 236.

**Parameters**
The parameters are the same as those for `startwpworkflowclassic`, page 140.

**subscriptions**

**Purpose**
Launches the component `subscriptions`, page 242 in the properties, page 221 container.

**Parameters**
This action has the same parameters as `permissions`, page 128.
taxonomies

**Purpose**
Launches the component taxonomies, page 248 in the properties container to display taxonomies for a selected object. This action operates on wcm_channel objects. The action is called by menubar, page 209.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format of the selected object.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the selected object.</td>
</tr>
</tbody>
</table>

templatebrowse

**Purpose**
Launches the container contenttemplatelocator and container, page 180 so that the user can select a content template.

**Parameters**

This action has no defined parameters.

thumbnailpreview

**Purpose**
Launches the component thumbnailpreview, page 250 in the associations container. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID for the selected object.</td>
</tr>
</tbody>
</table>

thumbnail

**Purpose**
Displays a list of thumbnail images (representing files) for the selected objects. This action is called from the UI of menubar, page 209.
**translationslist**

**Purpose**  Launched the component translationslist, page 251 in wpnavigationcontainer using jump navigation. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected document.</td>
</tr>
<tr>
<td>folderId</td>
<td>Folder containing the selected object.</td>
</tr>
<tr>
<td>type</td>
<td>Required. Type of the selected object.</td>
</tr>
</tbody>
</table>

**updatecontent**

**Purpose**  Updates all XML documents that are based on a selected template by using an instructions file. The action launches the component updatecontent, page 252 in the container wpmultiargdialogcontainer, page 278.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>Required. Template ID.</td>
</tr>
<tr>
<td>objectId</td>
<td>Required. List of selected documents to be updated.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Required. Name of the user who has checked out the template.</td>
</tr>
<tr>
<td>specialApp</td>
<td>Required. See status, page 102.</td>
</tr>
<tr>
<td>versionLabel</td>
<td>Required. Version of the template.</td>
</tr>
</tbody>
</table>

**uploadcontent**

**Purpose**  Uploads content from the client machine to the Web server by launching the component uploadcontent and container, page 252 in the uploadcontentcontainer.
**validatecontenttemplate**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be uploaded.</td>
</tr>
<tr>
<td>filePath</td>
<td>Required. Path to the object to be uploaded.</td>
</tr>
</tbody>
</table>

**Purpose**

Validates for well-formed XML a content template having the extension *.html or *.xml if the template is located in the templates folder. Operates on objects of type dm_document or its subtypes.

**versions**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder that contains the selected object.</td>
</tr>
<tr>
<td>type</td>
<td>Object type</td>
</tr>
</tbody>
</table>

**Purpose**

Launches the versions component in the wpnavigationcontainer using jump navigation. Operates on objects of type dm_document or its subtypes.

**view**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
</tbody>
</table>

**Purpose**

Launches the viewing application for viewing the selected object on the user’s local system.component. The <permit> element passes to the execution class the basic permission of read_permit (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). If the content can be streamed to the browser or if globalization is off and the content is html, the view action shows the content in a new browser window. For all other cases, the view action shows the content by using the content transfer applet. If the content is an XML file, the view action will try to show the HTML rendition if one exists.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the selected object</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
<tr>
<td>contentSize</td>
<td>Size of the object in bytes</td>
</tr>
<tr>
<td>isReference</td>
<td>Boolean flag that indicates whether the object is a reference object</td>
</tr>
<tr>
<td>isCurrentVDMNode</td>
<td>Boolean flag that specifies whether the node is the root node of the current virtual document.</td>
</tr>
<tr>
<td>isVirtualDoc</td>
<td>Boolean flag that specifies whether the object is a virtual document (required for pre-4.1 Docbase virtual documents)</td>
</tr>
<tr>
<td>linkCount</td>
<td>Number of links (child nodes) in a document. If greater than 0, the document is a virtual document.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Name of the lock owner</td>
</tr>
<tr>
<td>pageModifier</td>
<td>Specifies the string that is added to the end of the file name to denote the file format</td>
</tr>
<tr>
<td>pageNumber</td>
<td>Used by Digital Asset Manager client to stream images</td>
</tr>
<tr>
<td>folderPath</td>
<td>Path to the object in the Docbase</td>
</tr>
<tr>
<td>navigation</td>
<td>Type of navigation upon execution. Valid values: jump</td>
</tr>
</tbody>
</table>

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;display_xml_transformation&gt;</td>
<td>Set to true to display XML files as transformed HTML files</td>
</tr>
</tbody>
</table>

view_all_of_type

Purpose
Displays all manual link types of the selected type in the application. This action is called from the toolbar, page 250 UI.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Array of IDs of link types to view</td>
</tr>
</tbody>
</table>
view (wp task)

**Purpose**
Launches the webcomponent layer's taskmanager container component to display the workflow info component. The type scope of dm_wp_task is defined within the action and not in the Docbase.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object Id</td>
<td>Required. ID of the selected object</td>
</tr>
<tr>
<td>wptaskmanagerId</td>
<td>Required. ID of the WP task manager component that will display workflow info.</td>
</tr>
</tbody>
</table>

**Elements**
The following argument is defined for the `<execution><arguments><argument>` element.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>containertabid</td>
<td>Specifies the component tab to be displayed by default in the taskmanagercontainer.</td>
</tr>
</tbody>
</table>

viewwplocalelist

**Purpose**
Launches the component wplocalelist, page 277 to display supported locales.

**Parameters**
This action has no defined parameters.

viewsourc e

**Purpose**
Launches the component wp_viewsource and container, page 272 in the wp_viewsource_container when the user selects the action in the toolbar UI. The user who views the document source must have read_permit access to the document (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). The source is exported from the Docbase to the user’s temporary download directory. Operates on objects of type dm_document or its subtypes.
**webcabinets**

**Purpose**
Launches taxonomywebcabinetslist, page 248 in the properties container to display category cabinets in the Docbase. This action is called from the UI of menubar, page 209.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be exported.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
<tr>
<td>contentSize</td>
<td>Size of the object in bytes</td>
</tr>
</tbody>
</table>

**webview**

**Purpose**
Launches webview_component, page 260 to preview a file in Web view.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object Id</td>
<td>Required. ID of the selected object</td>
</tr>
<tr>
<td>pageModifier</td>
<td>Specifies the string that is added to the end of the file name to denote the file format</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
<tr>
<td>pageNumber</td>
<td>Parameter used by Digital Asset Manager</td>
</tr>
<tr>
<td>renditionId</td>
<td>ID of the rendition. If null, the object is considered to be the primary object and its format is used as the source format.</td>
</tr>
</tbody>
</table>
**wftemplatebrowse**

**Purpose**
Launches the container *wpwftemplatelocator* and container, page 287 to display the status of a workflow.

**Parameters**
This action has no defined parameters.

**whereused**

**Purpose**
Launches the component *whereused*, page 261. Operates on objects of type *dm_document* or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Folder path that is displayed in the breadcrumb. This path must be a complete path from the Docbase root, for example, /Doc%20Library/subfolder</td>
</tr>
<tr>
<td>wpcontext</td>
<td>Relation name. Valid values are defined in the component <em>contextsensitiveview_config</em>, page 181.</td>
</tr>
<tr>
<td>versionlabel</td>
<td>Version to be located</td>
</tr>
</tbody>
</table>

**workflowhistory**

**Purpose**
Launches *wpworkflowhistory*, page 287 to display the history of a selected workflow. This action is called from the UI of *menubar*, page 209.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected workflow.</td>
</tr>
</tbody>
</table>
**workflowstatusclassic**

**Purpose**
Launches workflowstatusclassic, page 262 so that the user can view the status of workflows.

**Parameters**
This action has no defined parameters.

**wp_editwithce**

**Purpose**
Jumps to the component wp_ceedit, page 265 to edit a document using the content editor. Operates on objects of type dm_document or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be checked out for editing.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Name of the user who has checked out the document</td>
</tr>
<tr>
<td>contentSize</td>
<td>Size of the document in bytes</td>
</tr>
</tbody>
</table>

**wp_editwithre**

**Purpose**
Jumps to the wp_reedit, page 271 component to edit a rules file using the Rules Editor after a new file is created. Operates on objects of type dm_document or its subtypes.

**Parameters**
This action has the same parameters as wp_editwithce, page 149.

**wp_editwithrenested**

**Purpose**
Nests to the wp_reedit component to edit a rules file using the Rules Editor. Operates on objects of type dm_document or its subtypes.
Parameters

This action has the same parameters as `wp_editwithce`, page 149.

**wp_editwithwhe**

**Purpose**
Launches the component `wp_wheedit`, page 272 to edit a file using eWebEditPro if the application is installed and the user has configured the default HTML editor for Web Publisher to be eWebEditPro. Operates on objects of type `dm_document` or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objectId</code></td>
<td>Required. ID of the object to be checked out for editing.</td>
</tr>
<tr>
<td><code>lockOwner</code></td>
<td>Name of the user who has checked out the document</td>
</tr>
<tr>
<td><code>contentSize</code></td>
<td>Size of the document in bytes</td>
</tr>
<tr>
<td><code>folderPath</code></td>
<td>Path to file.</td>
</tr>
</tbody>
</table>

**wpcancelcheckout**

**Purpose**
Launches the component `wpcancelcheckout`, page 150 in the `wp_cancelcheckoutcontainer`. The user who cancels checkout must have version_permit level to the document (valid values: `DELETE_PERMIT` | `WRITE_PERMIT` | `VERSION_PERMIT` | `RELATE_PERMIT` | `READ_PERMIT` | `BROWSE_PERMIT` | `NONE_PERMIT`). Operates on objects of type `dm_document` or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objectId</code></td>
<td>Required. ID of the object that is checked out.</td>
</tr>
<tr>
<td><code>lockOwner</code></td>
<td>Name of the user who has checked out the document</td>
</tr>
<tr>
<td><code>isReference</code></td>
<td>Boolean flag that specifies whether a document is a reference object</td>
</tr>
</tbody>
</table>

**wpcheckin**

**Purpose**
Launches the component `wp_checkin and container`, page 265 in `wp_checkincontainer`. The user who checks in the document must have version_permit access to the document (valid values: `DELETE_PERMIT` | `WRITE_PERMIT` | `VERSION_PERMIT` | `RELATE_PERMIT` | `READ_PERMIT` | `BROWSE_PERMIT` | `NONE_PERMIT`).
WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). Operates on objects of type dm_document or its subtypes.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be checked in.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Name of the user who has checked out the document</td>
</tr>
<tr>
<td>ownerName</td>
<td>Name of the document owner</td>
</tr>
<tr>
<td>isReference</td>
<td>Boolean flag that specifies whether a document is a reference object</td>
</tr>
</tbody>
</table>

wpcheckout

Purpose

Launches the wp_checkout component in the wp_checkout container. The user who checks out the document must have version_permit access to the document (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). Operates on objects of type dm_document or its subtypes.

Parameters

This component has the same parameters as wpcheckin, page 150.

wpcopyaction

Purpose

Launches the component wpcopy, page 275 to perform a clipboard copy operation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the target folder.</td>
</tr>
</tbody>
</table>

wpdeeppromote

Purpose

Launches the component deeppromote and container, page 185 in its container to promote the contents of a folder to the next state. This action is scoped to channels (wcm_channel) and channel folders (wcm_channel fld).
wpdemote

Parameters

| objectId | Required. ID of the folder whose contents should be promoted. |

wpdemote

Purpose

Launches the component demote and container, page 186 in demotecontainer to demote an object to the previous state. This action is scoped to dm_document and wcm_change_set.

Parameters

| objectId | Required. ID of the object to be demoted. |

wplinkaction

Purpose

Launches the component wplink, page 277 to perform a clipboard link operation.

Parameters

| objectId | Required. ID of the target folder. |

wpmoveaction

Purpose

Launches the component wpmove, page 278 to perform a clipboard move operation.

Parameters

| objectId | Required. ID of the target folder. |

wppromote

Purpose

Launches the component promote and container, page 221 in promotecontainer to promote an object to the next state. This action is called by menubar, page 209. The action operates on dm_document and wcm_change_set.
**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the object to be promoted.</th>
</tr>
</thead>
</table>

**wptaskmanagerchangesetremovefiles**

**Purpose**
Removes files from a change set. Operates on objects of type dm_sysobject or its subtypes.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the file to be removed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>isCsStateOK</td>
<td>Boolean flag that specifies whether the change set state allows removing attachments.</td>
</tr>
</tbody>
</table>

**wpviewclipboardaction**

**Purpose**
Launches the component wpclipboard and container, page 274 to display the contents of the clipboard.

**Parameters**

This action has no defined parameters.
Web Publisher components are located in `APP_HOME/wp/config`. Web Publisher components are configured and extended in the same way as other WDK components. See *Web Development Kit and Client Applications Development Guide* for general information about component configuration.

This part of the documentation describes components that are unique to the Web Publisher application.

This section provides the following information about a component where applicable:

- **Parameters**
  Components have required or optional parameters that are defined in the component definition. You can call a component via URL, JavaScript, or server-side code, passing in the parameters. All parameters are passed as strings, but some components may cast the parameter into another type. These expected parameter types are noted.

- **Extends**
  A component can extend a base component definition and inherit all or part of the definition of the parent component. If an element is defined in the extended definition, that element overrides the contents of the element in the parent definition. For example, if the parent specifies an objectld parameter and the child definition has an empty parameters element, then no parameters are required for the child component.

- **Called by**
  A component is generally by an action or by another component class. The calling agent is described for each component.

- **Elements**
  A component may define configurable elements that are specific to the component.

- **Usage**
  This section describes notes for configuring or customizing the component.

The following Web Publisher components extend WDK or Webtop components, adding functionality or a different layout page (JSP):
### Table 19–1. Web Publisher Extended Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Extends</th>
<th>Added functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>applylifecycleall</td>
<td>webcomponent layer’s applylifecycle component</td>
<td>Checks whether the lifecycle is a valid Web Publisher lifecycle</td>
</tr>
<tr>
<td>editionlist</td>
<td>Webtop layer’s objectlist component</td>
<td>Adds an onClose() event handler and lists all objects in an edition</td>
</tr>
<tr>
<td>general_preferences</td>
<td>webcomponent layer’s general_preferences component</td>
<td>Adds startup view preference (streamline or classic)</td>
</tr>
<tr>
<td>preferences</td>
<td>webcomponent layer’s preferences component</td>
<td>Adds Web author and Web developer preferences</td>
</tr>
<tr>
<td>savedsearches</td>
<td>webcomponent layer’s savedsearches component</td>
<td>Adds WP context arguments</td>
</tr>
<tr>
<td>startwpwfcomments</td>
<td>webcomponent layer’s startwpwfcomments component</td>
<td>Sets value of <code>&lt;showheaderforsinglepackage&gt;</code> to false</td>
</tr>
<tr>
<td>streamlineview</td>
<td>Webtop layer’s streamlineview component</td>
<td>Filters out WP cabinets and folders from the folder path</td>
</tr>
<tr>
<td>wpmultiargdialog</td>
<td>webcomponent layer’s multiargdialog component</td>
<td>This component is called by the <code>addtochangeset</code>, page 161 component and adds a folder ID to the container</td>
</tr>
</tbody>
</table>

---

**abortwpworkflow**

**Purpose**

Extends the webcomponent layer’s abortworkflow component. Terminates Documentum workflows prematurely, deletes them, and updates the change set. This component is scoped to `dm_workflow` objects and is generally used to abort obsolete workflows. This component is launched by the action `abortworkflow`, page 106 in the combocontainer.
Parameters

| objectId | Required. The ID of the workflow object to be terminated. |

about

Purpose
Extends the Webtop about component and displays information about the Web Publisher application. This component is called from an action menu item in the menubar, page 209 component and is nested within the icetitlebar, page 200 component and the main, page 209 component titlebar page.

Parameters
This component inherits the parameters of Webtop’s about component.

Elements

| <filter> | The role attribute value specifies the wprole that is permitted to modify configurations in the nested hiddentool, page 194 component. |
| <enableconfig> | True to allow the specified role to modify configurations. |

abouttoactivecriteria

Purpose
Displays a UI to set criteria for a report on files that are ready to be promoted to active.

Parameters
The parameters are the same as those for alerts, page 164.

abouttoexpirecriteria

Purpose
Displays a UI to set criteria for a report on files that are ready to expire.
Parameters

The parameters are the same as those for alerts, page 164.

activeobjectscriteria

Purpose

Displays a UI to set criteria for a report on files that are active.

Parameters

The parameters are the same as those for alerts, page 164.

add_locale and container

Purpose

Adds a supported locale to the application. The addlocalecontainer extends the WDK layer’s dialogcontainer.

Parameters

This component has no defined parameters.

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;languagelist&gt;</td>
<td>Contains <code>&lt;language&gt;</code> elements that correspond to all supported languages</td>
</tr>
<tr>
<td><code>&lt;language&gt;</code></td>
<td>Attributes: name specifies the language name</td>
</tr>
<tr>
<td><code>&lt;countrylist&gt;</code></td>
<td>Contains <code>&lt;country&gt;</code> elements that correspond to all of the countries for which there is a set of translated documents.</td>
</tr>
<tr>
<td><code>&lt;country&gt;</code></td>
<td>Attributes: name specifies the country name</td>
</tr>
</tbody>
</table>
add_manual_link

Purpose
Extends the webcomponent layer’s documentlocatorcontainer and displays a UI to add a manual link between two objects. This component contains the following components: webcabinetslocator, wpsubscriptionlocator, and wpmyfileslocator. This component is called by the add_manual_link, page 107 action.

Parameters

| objectId | Required. Selected object to which link will be added. |

Elements
This component no configurable element.

add_manual_link_type

Purpose
Adds a supported type of manual link to the application. The manual link type that is entered in the UI by the administrator must correspond to a dm_relation_type in the Docbase. This component is called by the add_manual_link, page 107 action.

Parameters
This component has no defined parameters.

add_translation and addtranslationcontainer

Purpose
Displays a UI to add a translation. The container extends the webcomponent layer’s contentxfercontainer component.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the translation.</td>
</tr>
<tr>
<td>parentId</td>
<td>Required. ID of the current document to which a translation will be added.</td>
</tr>
<tr>
<td>localeDisplay</td>
<td>Required. Locale to display for the translation</td>
</tr>
<tr>
<td>type</td>
<td>Object type of the translation document</td>
</tr>
<tr>
<td>folderId</td>
<td>Required. Docbase folder to which the translation will be added.</td>
</tr>
</tbody>
</table>

**addedition and container**

**Purpose**  Adds an edition to the archive. This component is called by the action `addedition`, page 108. The container extends the WDK layer’s dialogcontainer.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the channel (delivery cabinet) to which the object will be added.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the object</td>
</tr>
</tbody>
</table>

**addnewrule and addnewruledialogcontainer**

**Purpose**  Adds a new rules file in a folder or cabinet. The user must have write_permit access to the destination folder or cabinet (valid values: DELETE_PERMIT | WRITE_PERMIT | VERSION_PERMIT | RELATE_PERMIT | READ_PERMIT | BROWSE_PERMIT | NONE_PERMIT). The container extends the WDK layer’s dialogcontainer.

**Parameters**

The container requires the objectId as well as the component ID (required by all containers). The component requires the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the folder that will contain the new rules file.</td>
</tr>
</tbody>
</table>
addpresentation and container

Purpose Displays a UI to add a selected XML style sheet or HTML presentation file and to set the format, locale, and content application for the presentation. The container addpresentationcontainer extends the container wpsysobjectlocator and container, page 281. The component is called from a link in the UI for the component presentations, page 219.

Parameters The component defines the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected presentation.</td>
</tr>
<tr>
<td>selectedTemplateId</td>
<td>ID of a template for the presentation</td>
</tr>
</tbody>
</table>

The container has the same parameters as categoryassetlocator and container, page 169 with the additional parameters described above for the addpresentation component.

addrule and container

Purpose Adds a selected rules file to the application, presenting a UI to set the locale for the rule. The component is called by the action addrule, page 109 in the component UI of rules, page 230. The container addrulecontainer extends the container wpsysobjectlocator and container, page 281.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected rule file.</td>
</tr>
<tr>
<td>selectedTemplateId</td>
<td>ID of a rules template</td>
</tr>
</tbody>
</table>

The container has the same parameters as categoryassetlocator and container, page 169 with the additional parameters described above for the addrule component.

addtochangeset

Purpose Adds a file to a change set. This component is called by the action addtochangeset, page 109 in the component UI of menubar, page 209.
**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>Required. ID of the folder that contains the change set.</td>
</tr>
</tbody>
</table>

**adminconsole**

**Purpose**

This component is called as a node in the administration, page 162 component.

**Parameters**

This component can be initialized in four ways:

- No parameters: Displays Docbase root cabinets
- folderId: Displays the primary location of the specified folder ID
- folderPath: Displays the specified folder
- nodeIds: Uses a list of ‘.’ separated object IDs that are used to build an absolute location to be displayed

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>The folder of the adminconsole in the browser tree</td>
</tr>
<tr>
<td>folderPath</td>
<td>The folder path to be displayed in the adminconsole. This path must be full, from the Docbase root, for example, /Docbase%20Admin</td>
</tr>
<tr>
<td>nodeIds</td>
<td>IDs that constitute the path to the adminconsole component in the browsertree</td>
</tr>
</tbody>
</table>

**Elements**

This component has configurable nodes (see the webcomponent layer’s usermanagement component).

**administration**

**Purpose**

Extends sitemanager, page 234 and displays a list of links to install administration tools.

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeIds</td>
<td>ID and parameters for a target component</td>
</tr>
</tbody>
</table>
Elements

This component has configurable nodes (see the webcomponent layer's usermanagement component) with an additional element:

| `<wpcontext>`          | Specifies the set of actions permissible on objects in the view (see contextsensitiveview_config, page 181 for details). |

**advsearch**

**Purpose**

This component configures the advanced search options.

**Parameters**

| folderpath | The path to the folder to be searched. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder |

**Elements**

The configurable elements are the same as those for the webcomponent layer's advsearch component.

**Usage**

This component must be extended to search on object types or attributes that are not `dm_sysobject`, `dm_folder`, or `dm_document`.

**advsearchcs and container**

**Purpose**

This component configures the advanced search options to search change sets. The container extends the WP advsearchcontainer.

**Parameters**

The parameters are inherited from WP advsearch.
Elements

The configurable elements are the same as those for the webcomponent layer’s advsearch component.

advsearchsup and container

Purpose

This component configures the advanced search options to search supplemental files for change sets. The container extends the WP advsearchcontainer.

Parameters

This component has no defined parameters.

Elements

The configurable elements are the same as those for the webcomponent layer’s advsearch component with the following additional elements:

<table>
<thead>
<tr>
<th>&lt;columns_list&gt;</th>
<th>Contains a list of &lt;column&gt; elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;filter&gt;</td>
<td>Can be placed around an element that will be shown only in the specified context. For example, in the default configuration the column for the language_code attribute is displayed only when the context is wpsearch.</td>
</tr>
</tbody>
</table>

alerts

Purpose

Displays Content Server system alerts. This component is called by the action alerts, page 110 in the UI of the components toolbar, page 250 and publishingconfig, page 226.

Parameters

<table>
<thead>
<tr>
<th>parentNodeTitle</th>
<th>Parent node in the breadcrumb path to the current node</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentld</td>
<td>ID of the parent node</td>
</tr>
<tr>
<td>nodeIds</td>
<td>Target nodes in the tree</td>
</tr>
</tbody>
</table>
**applylifecycleall**

**Purpose**
Applies a lifecycle to selected objects and validates that the lifecycle is a WP lifecycle.

**Parameters**

| objectId | Required. ID of the object to which the lifecycle shall be applied |

**applypresentationcomponent and container**

**Purpose**
Displays a list of files that can be selected for application of the current presentation.

**Parameters**

| objectId | Object ID of the presentation |
| type     | Required. Object type of the selected presentation. |

The container applypresentationcontainer has the same parameters as the component, plus the <component> parameter, which overrides any <component> element within the container <contains> element.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component).

**associateworkflow**

**Purpose**
Displays a UI for selecting a workflow to be associated with the selected object. This component is called by the action associateworkflow, page 111 in the associations container from the component UI of menubar, page 209.

**Parameters**

| objectId | Required. The selected object. |
associations

Purpose
Extends the WDK layer’s propertysheetcontainer component and contains several association components that associate a selected object with another object: thumbnailpreview, presentations, rules, and associateworkflow.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Required. One or more components to be contained.</td>
</tr>
<tr>
<td>objectId</td>
<td>Required. The selected object that will be associated with a thumbnail, presentation, rule, or workflow.</td>
</tr>
<tr>
<td>associateVersion</td>
<td>Set to true to associate a the target object with the current version of the selected object. If false (set by the versions component for a non-current version), the association cannot be created.</td>
</tr>
</tbody>
</table>

associationslocatorcontainer_component

Purpose
Extends the webcomponent layer’s locatorcontainer component and contains one or more sysobjectlocator components. This container is not currently used in Web Publisher.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objecttype</td>
<td>Type of the selected object</td>
</tr>
<tr>
<td>flatlist</td>
<td>Boolean: true to list all objects in a flat list, false to display a drop down list.</td>
</tr>
</tbody>
</table>

attributes (dm_document, dm_category)

Purpose
Presents a UI for viewing and setting attributes on objects of type dm_document or dm_category. For wcm attributes, see attributes (wcm types), page 167. The attributes component is designed to run within the properties container component.
**attributes (dm_taxonomy)**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required The ID of the object for which attributes are to be shown.</td>
</tr>
<tr>
<td>readOnly</td>
<td>Boolean: Displays the attributes as read-only (true) or in edit mode (false). Applies to dm_document.</td>
</tr>
<tr>
<td>contentType</td>
<td>Type of the object. This parameter is generally supplied by the actionmultiselect control. Applies to dm_document.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Specifies the folder path to the category. Applies to dm_category only. This path must be a complete path from the Docbase root, for example, /Doc%20Library/subfolder</td>
</tr>
</tbody>
</table>

**attributes (dm_taxonomy)**

**Purpose**

Presents a UI for viewing and setting attributes on a taxonomy in a category (dm_taxonomy). The attributes component is designed to run within the properties container component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required The ID of the taxonomy for which attributes are to be shown.</td>
</tr>
</tbody>
</table>

**Elements**

For objects of type dm_taxonomy, a <filter> element filters for WP role. Non wcm_administrator role is limited to a read-only view of the attributes by the <filter>.<readonly> element.

**attributes (wcm_types)**

**Purpose**

Presents a UI for viewing and setting attributes on objects of type dm_taxonomy and wcm_*: wcm_category, wcm_change_set, wcm_channel, wcm_channel_fld, wcm_edition, wcm_edition_fld, and wcm_locale. Each scope has its own display JSP page, behavior class, NLS bundle, and help ID. The attributes component is designed to run within the properties container component.
Parameters

| objectId | Required The ID of the object for which attributes are to be shown. |

**authenticate**

**Purpose**
Extends the Webtop layer’s authenticate component and adds the Web Publisher alert JSP page to the UI and specifies the role that establishes the default locale for the Docbase.

**Parameters**
This component has no defined parameters.

**Elements**

| <wprole> | Specifies the role that sets the default locale for the Docbase. Default role: administrator |

**browsertree**

**Purpose**
Extends the Webtop browsertree component and changes the nodes in the browser tree.

**Parameters**

| folderPath | Specifies the folder path to the node that is initially selected in the browser tree. This path must be a complete path from the Docbase root, for example, /Doc%20Library/subfolder |
| entryNode | Specifies the node that is initially selected the browser tree |

**Elements**
This component has the same configurable elements as the Webtop browsertree component, with the following additions.
categories

Purpose Displays categories to which the selected object is assigned in the application. This component is called by the action categories, page 112 in the component UI of menubar, page 209.

Parameters

| objectId | Required. ID of the selected object. |
| readOnly | Set to true to display the categories without allowing the user to remove the object from a category or add it to another category |

Elements

| <wpcontext> | Specifies the set of actions permissible on objects in the view (see contextsensitiveview Config, page 181 for details). |

categoryassetlocator and container

Purpose Extends the webcomponent layer's allfolderlocator component and locates all Docbase category folders of type wcm_category. The component lets a user navigate from the Content Template folder to locate a category. The categoryassetlocatorcontainer extends the webcomponent layer's folderlocatorcontainer.
category_classic

Parameters

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objecttype</td>
<td>Type of the selected object</td>
</tr>
<tr>
<td>flatlist</td>
<td>Boolean: true to list all objects in a flat list, false to display a drop down list.</td>
</tr>
<tr>
<td>multiselect</td>
<td>Boolean: true to support multiple object selection</td>
</tr>
</tbody>
</table>

Elements

The configurable elements are the same as those for the webcomponent layer’s objectlocator component.

category_classic

Purpose

The category list view component extends channel_classic, page 174.

Parameters

This component inherits the parameters of channel_classic, page 174

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext&gt;</td>
<td>Specifies the set of actions permissible on objects in the view (see contextsensitiveview_config, page 181 for details).</td>
</tr>
</tbody>
</table>

category_streamline

Purpose

The category streamline view component extends the WP channel_streamline component.

Parameters

This component has no defined parameters.
categorylocaleslist

**Purpose**
Lists translations for a selected category. This component is called by the action `categorylocaleslist`, page 112 in the UI of `menubar`, page 209.

**Parameters**
This component has no defined parameters.

**Elements**
A `<filter>` element filters for WP role. Non `wcm_administrator` role is limited to a read-only view of the attributes by the `<filter>`.<readOnly> element.

changesetaddfilescontainer

**Purpose**
Extends the WDK layer’s `propertysheetcontainer` component and contains several components to add files to a change set: `csmyfiles_classic` (user’s change sets), `cssubscriptions_classic` (subscribed change sets), `cscategory_classic` (change set folders), and `searchcs` (search change sets).

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Specifies the start component</td>
</tr>
<tr>
<td>csfolderId</td>
<td>ID of the change set folder</td>
</tr>
</tbody>
</table>

changesetaddsupportingfilescontainer

**Purpose**
Extends the WDK layer’s `propertysheetcontainer` component and contains several components that add supporting files to a change set: `supmyfiles_classic` (user’s change set supporting files), `supsubscriptions_classic` (subscribed change set supporting files), `supcategory_classic` (change set supporting file folders), `searchcsup` (search change set supporting files), `supchannellist` (list of supporting files delivery
channels), publicsupportingfiles (public supporting files), and usersupportingfiles (supporting files that belong to the user).

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Specifies the start component</td>
</tr>
<tr>
<td>csfolderId</td>
<td>ID of the change set folder</td>
</tr>
</tbody>
</table>

**changesetdetailscontainer**

**Purpose**

Extends the WDK layer’s propertysheetcontainer component and contains components that display details about a change set: changeset_files_list and changeset_supportingfiles_list.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Specifies the start component</td>
</tr>
<tr>
<td>objectId</td>
<td>ID of the change set</td>
</tr>
</tbody>
</table>

**changeset_files_list**

**Purpose**

Extends the Webtop layer’s objectlist component and displays the files in a change set

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the change set</td>
</tr>
</tbody>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component) with the following additional element:

<table>
<thead>
<tr>
<th>element</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;objecttype&gt;</td>
<td>Specifies the object type to be displayed by default</td>
</tr>
</tbody>
</table>
changesetfilter

**Purpose**
Displays a filter for change sets within the container `filtercontainer`, page 192. The filter menu is defined in the UI (`/wp/library/changeset/filter.jsp`) with the following options: change set status and reset change set list to default. This component is called from a link in the UI of the component `changesets_list`, page 173.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Set to true to display the change set status menu item</td>
</tr>
</tbody>
</table>

changesets_list

**Purpose**
Displays a list of change sets. This component is called as a node in the `sitemanager`, page 234 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeId</td>
<td>IDs of the nodes to be displayed in the browser tree</td>
</tr>
<tr>
<td>virtualPath</td>
<td>Specifies the virtual breadcrumb path to child components, for example, a/b/c.</td>
</tr>
<tr>
<td>components</td>
<td>Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c</td>
</tr>
<tr>
<td>creatorName</td>
<td>Change set owner</td>
</tr>
</tbody>
</table>

**Elements**
The configurable node elements are the same as those for the webcomponent layer’s `usermanagement` component. The configurable column elements are described in the webcomponent layer's `doclist` component.

`changeset_supportingfiles_list`

**Purpose**
Displays the supporting files in a change set.
### Parameters

| objectId   | Required. ID of the change set. |

### Elements

This component has configurable columns (see the webcomponent layer’s doclist component) with the following additional elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;objecttype&gt;</td>
<td>Specifies the object type to be displayed by default</td>
</tr>
<tr>
<td>&lt;wpcontext&gt;</td>
<td>Sets the WP context view for this component</td>
</tr>
<tr>
<td>&lt;wpcontext_nonwp&gt;</td>
<td>Sets the context view for non-WCM objects</td>
</tr>
</tbody>
</table>

### channel_classic

**Purpose**

Displays the contents of a channel in a classic view.

**Parameters**

The parameters are the same as those for adminconsole, page 162.

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component). Additional configurable elements include:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext&gt;</td>
<td>Specifies the set of permissible actions on WP objects. (See contextsensitiveview_config, page 181 for details). Default = wpchannellist.</td>
</tr>
<tr>
<td>&lt;wpcontext_root&gt;</td>
<td>Root folder path or base WP context</td>
</tr>
<tr>
<td>&lt;showtitle&gt;</td>
<td>Boolean flag to display the title for the channel list display</td>
</tr>
<tr>
<td>&lt;excludedocinchangeset&gt;</td>
<td>Boolean flag to exclude change set docs from the channel list</td>
</tr>
</tbody>
</table>
channelclassicfilter

Purpose
Extends channelfilter, page 175 and displays a filter for channels in the classic view within the container filtercontainer, page 192. The filter menu is defined in the UI (/wp/classic/channel/filter.jsp) with the following options: files and folders, files only, folders only, locale, status, and reset to default. This component is called from a link in the UI of the component channel_classic, page 174.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specify a valid Docbase type for the filter criteria</td>
</tr>
<tr>
<td>locale</td>
<td>Set to true to display the locale menu item</td>
</tr>
<tr>
<td>showfallbacks</td>
<td>Set to true to display the showfallbacks option in the menu</td>
</tr>
<tr>
<td>status</td>
<td>Set to true to display the change set status menu item</td>
</tr>
</tbody>
</table>

channelfilter

Purpose
Extends contentfilter, page 178 and displays a filter for channels within the container filtercontainer, page 192. The filter menu is defined in the UI (/wp/streamline/channel/filter.jsp) with the following options: locale, status, and reset menu to default. This component is called from a link in the UI of the component channel_streamline, page 175.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>Set to true to display the locale menu item</td>
</tr>
<tr>
<td>showfallbacks</td>
<td>Set to true to display the showfallbacks option in the menu</td>
</tr>
<tr>
<td>status</td>
<td>Set to true to display the change set status menu item</td>
</tr>
</tbody>
</table>

channel_streamline

Purpose
Displays the contents of a channel in a streamline view.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>The initial folder that is displayed</td>
</tr>
<tr>
<td>folderPath</td>
<td>The initial folder path that is displayed. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
</tbody>
</table>

Elements

This component has configurable columns (see the webcomponent layer’s doclist component). Additional configurable elements are the same as those for channel_classic, page 174.

cis and container

Purpose

Displays suggested values for attributes from Content Intelligence Services for a selected object. This component must run in the ciscontainer component. You must configure the Web Publisher global settings to use CIS integration. See Web Publisher Administration Guide for more information on CIS integration. See Chapter 9, Mapping CIS Properties to Docbase Attributes for instructions on setting up CIS mapping to Docbase attributes in the CIS component.

Parameters

The following parameter is required by the cis component and ciscontainer:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
</tbody>
</table>

Elements

The elements in this component constitute the map of CIS values to Docbase attributes.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;attribute_list&gt;</td>
<td>Contains &lt;attribute&gt; elements that correspond to CIS property names</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>Contains an attribute mapping rule that matches a CIS name to an attribute</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>Attribute name</td>
</tr>
<tr>
<td>&lt;cis_name&gt;</td>
<td>String that matches a property map entry that is defined in the CIS server</td>
</tr>
<tr>
<td>&lt;nlsid&gt;</td>
<td>Specifies a label to be used for the attribute. If none is supplied, the data dictionary label will be used.</td>
</tr>
</tbody>
</table>
configurations

Purpose

Extends sitemanager, page 234 and displays a configurations node under the Site Manager branch of the browsertree. Child nodes of the configurations component consist of foldermaps and administrator. In the content frame, the component displays foldermaps and administrator as virtual folders.

Parameters

The parameters are the same as those for channel_classic, page 174 with the following additional elements:

| virtualpath | Specifies the virtual breadcrumb path to child components |
| components  | Specifies the components to be displayed as child nodes |

Elements

This component has configurable nodes (see the webcomponent layer’s usermanagement component).

contained_in_content_list

Purpose

Displays the objects that contain the selected object. This component is called in the list_related_content, page 206.

Parameters

| objectId  | Required. Selected object. |

Elements

This component has configurable columns (see the webcomponent layer’s doclist component). The following additional configurable element is available:

| objecttype | Object type to display by default |
contains_content_list

**Purpose**
Extends the Webtop layer’s objectlist component and displays the content that is contained by the selected object. This component is called in the list_related_content, page 206.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. Selected object.</th>
</tr>
</thead>
</table>

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component). The following additional configurable element is available:

<table>
<thead>
<tr>
<th>objecttype</th>
<th>Object type to display by default</th>
</tr>
</thead>
</table>

contentfilter

**Purpose**
Displays a filter for categories within the container filtercontainer, page 192. The filter menu is defined in the UI /wp/streamline/category/filter.jsp) with the following options: locale, status, and reset menu to default. This component is called from a link in the UI of the component category_streamline, page 170.

**Parameters**

<table>
<thead>
<tr>
<th>locale</th>
<th>Set to true to display the locale menu item</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Set to true to display the change set status menu item</td>
</tr>
</tbody>
</table>

contentlist

**Purpose**
Extends the component channel_streamline, page 175 and displays a list of content that was created from a selected template. This component is launched by the MySubmitContent and MyPublishedContent portlets.
Parameters

If no parameter value is supplied, all published content with no category will be displayed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>templateId</td>
<td>ID of template for which to display a list of content files using the template. If supplied with state, all content using the template in the specified state are displayed.</td>
</tr>
<tr>
<td>state</td>
<td>State of the content (WIP, staging, approved, active, expired)</td>
</tr>
<tr>
<td>categoryId</td>
<td>All content that belongs to the specified category is displayed. If categoryId and folderId are specified, all content in the specified category that is also linked to the specified folder (including sub-folders) is displayed.</td>
</tr>
<tr>
<td>folderId</td>
<td>Folder for which to show content</td>
</tr>
</tbody>
</table>

contenttemplateassetlocator and container

Purpose

Extends wpcontenttemplatelocator and container, page 275 and displays a UI for locating content template protected assets. The container and component are called by the protectedassets, page 224 component.

Parameters

The parameters are the same as those for categoryassetlocator and container, page 169.

Elements

The configurable elements are the same as those for the webcomponent layer’s objectlocator component.

contenttemplatecriteria

Purpose

Displays a UI for requesting a report on content templates. The component is called when the user selects the contenttemplatecriteria option in the reportselection component. The component nests to the contenttemplatereport component.
contenttemplate locator and container

**Purpose**
Extends wpcontenttemplate locator and container, page 275 and displays a UI for locating content templates. The container is nested within the wp_reedit, page 271 component.

**Parameters**
The parameters are the same as those for categoryasset locator and container, page 169 with the following additional parameter:

| rulesId       | ID of the new rule |

**Elements**
The configurable elements are the same as those for the webcomponent layer’s objectlocator component.

contenttemplatereport

**Purpose**
Extends the Webtop objectlist component and displays a report on content templates. The component is called by the contenttemplatecriteria component.

**Parameters**
The parameters are the same as those for alerts, page 164.

Configurable Roles

**Purpose**
The contextsensitivityrole_config component configures the minimum set of valid actions for each Web Publisher role. The component is accessed by a combined control key and mouse click on the About icon in the Web Publisher frameset.

**Parameters**
There are no defined parameters for this component.
The configurable elements for the context_sensitive_role_config component are described below.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;role_list&gt;</code></td>
<td>Contains a list of roles (<code>&lt;required_minimum_role&gt;</code>)</td>
</tr>
<tr>
<td><code>&lt;required_minimum_role&gt;</code></td>
<td>The name attribute must correspond to a role that is defined in the Docbase. Contains actions permitted to the role (<code>&lt;valid_actions&gt;</code>).</td>
</tr>
<tr>
<td><code>&lt;valid_actions&gt;</code></td>
<td>Contains a list of actions (<code>&lt;action&gt;</code>) that are permitted for the role</td>
</tr>
<tr>
<td><code>&lt;action&gt;</code></td>
<td>The name attribute must correspond to an action ID that is defined in the application or in a parent application layer</td>
</tr>
</tbody>
</table>

The file `context_sensitive_role.xml` is generated by the `context_sensitive_config` component. Do not modify this file.

**Usage**

For an overview of Web Publisher roles, see Chapter 7, Role-based Configuration.

To define a set of minimum actions permitted to a role, list those actions within the `<required_minimum_role>`, `<valid_actions>` element that corresponds to the role. To limit the valid actions to none, use an empty `<valid_actions>` element. Your custom component can extend the `context_sensitive_role_config` component and add or remove valid actions for a role or add support for other roles defined in your Docbase.

If your custom definition changes the set of actions in this definition, you must duplicate any actions within the original set, or else your new set of actions for the role will overwrite the permitted set as originally defined. For example, if you are adding two actions for the manager role, you must copy all of the existing actions into your new definition file as well as adding your two actions. If your definition contains only the two new actions, none of the previously defined actions will be available to that role.

**context_sensitiveview_config**

**Purpose**

The `context_sensitiveview_config` component presents a UI for the administrator to configure valid and invalid actions for each Web Publisher view context. This configuration determines the actions that will be available through menus in the UI. The `context_sensitiveview_config` component generates the file `context_sensitive_view.xml`. Do not edit the latter file, as it is for internal use only.

**Parameters**

This component has no defined parameters.
Elements

The configurable elements in this component are described in the following table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;view_list&gt;</code></td>
<td>Contains a list of views (<code>&lt;view&gt;</code>) and the minimum set of actions defined for the view.</td>
</tr>
<tr>
<td><code>&lt;view&gt;</code></td>
<td>Contains actions that are exclusive to, permitted in, or invalid in the view. If the view extends another view, it inherits the actions permitted to the parent view.</td>
</tr>
<tr>
<td><code>&lt;exclusively_valid_actions&gt;</code></td>
<td>Contains a list of all of the valid actions (<code>&lt;action&gt;</code>) that are permitted only for the specific view. If the action is not in this list, it is not valid for the view. If an <code>&lt;exclusively_valid_actions&gt;</code> element is present, you cannot use an <code>&lt;invalid_actions&gt;</code> or a <code>&lt;valid_actions&gt;</code> element.</td>
</tr>
<tr>
<td><code>&lt;invalid_actions&gt;</code></td>
<td>Contains a list of actions that are not permitted for the view.</td>
</tr>
<tr>
<td><code>&lt;valid_actions&gt;</code></td>
<td>Contains a list of actions (<code>&lt;action&gt;</code>) that are permitted for the view. If empty, no actions are valid for the view.</td>
</tr>
<tr>
<td><code>&lt;action&gt;</code></td>
<td>The name attribute on this element must correspond to an action ID that is defined in the application or in a parent application layer. If the action has an instruction attribute value set to “remove”, the action is not defined for that view. This setting generates the action attribute value <code>notdefined=&quot;true&quot;</code> in <code>contextSensitive_view.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;actions_list&gt;</code></td>
<td>Contains a list of actions in the application</td>
</tr>
<tr>
<td><code>&lt;an_action&gt;</code></td>
<td>The name attribute on this element must correspond to an action ID that is defined in the application or in a parent application layer. If the <code>valid_by_default</code> attribute is set to <code>true</code>, the action is valid for all views by default.</td>
</tr>
<tr>
<td><code>&lt;an_action&gt;.&lt;invalid_views&gt;</code></td>
<td>Specifies any views (using an <code>&lt;view&gt;</code>) element) that are not valid for the action.</td>
</tr>
</tbody>
</table>

The file `contextSensitive_view.xml` is generated by the component `contextSensitiveView_config` and should not be edited.

Usage

For an overview of Web Publisher context, see Chapter 6, Web Publisher Context Views.

A component uses a view context to specify its own allowed and disallowed actions. The actions that are allowed and disallowed for a view are defined in the component definition file `contextSensitiveView_config_component.xml`. This component generates the file `contextSensitive_view.xml`. You should not edit `contextSensitive_view.xml`.

The menu actions that are available to any given component are restricted to the actions that are defined for the component’s view. To do this, the component definition specifies the view as the value of the `wpcontext` scope qualifier. For example, if a component’s `<scope>` element specifies a `wpcontext` value of `wppreviews`, the menu items available
for that component will be restricted to the list of actions in `<exclusively_valid_actions>` in contextsensitive_view_config.xml.

To limit the valid actions to none, use an empty `<exclusively_valid_actions>` element.

The disallowed actions for a wpview context are defined by the attribute instruction="remove" on an action element in the contextsensitiveview_config definition.

Use `<exclusively_valid_actions>` to define menu actions that are available only in a specific view. All actions in this element cannot be available in any other view. All actions not in this element are not available in the view.

A view can extend another view and inherit the actions specified for that view. You can add or remove actions defined in the parent view by naming the action in `<exclusively_valid_actions>`. For example, the wpchannellist view that is used by the channel_classic and channel_streamline components extends a base list of actions defined for the wpbaselist view and adds eight exclusive actions. The allrelationslist view extends the same wpbaselist view. It adds two exclusive actions and removes three that were defined in the parent view:

```
<view name="allrelationslist" extends="wpbaselist">
  <exclusively_valid_actions>
    <action name="add_manual_link"></action>
    <action name="remove_manual_link"></action>
    <action name="wpmoveaction" instruction="remove"></action>
    <action name="wpcopyaction" instruction="remove"></action>
    <action name="wplinkaction" instruction="remove"></action>
  </exclusively_valid_actions>
</view>
```

### copynonwcm

**Purpose**

Allows the user to select non Web Publisher items from the clipboard to copy. The clipboard of non-WCM items is displayed when the user selects a filter in the UI. If one or more selections cannot be copied, the **Copy** button is disabled.

### Parameters

<table>
<thead>
<tr>
<th>objectID</th>
<th>Required. ID of the target folder into which the items will be copied.</th>
</tr>
</thead>
</table>

### Elements

This component has the same configurable elements as **wpcopy**, page 275.
cscategory_classic

**Purpose**
Displays a list of category change sets. This component extends category_classic, page 170.

**Parameters**
This component inherits the parameters of channel_classic, page 174.

**Elements**
This component inherits the configurable elements of channel_classic, page 174.

cschannellist_classic

**Purpose**
Displays a list of channels that are in change sets. This component extends channel_classic, page 174 and is contained within changesetaddfilescontainer, page 171.

**Parameters**
This component inherits the parameters of channel_classic, page 174.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component). The only difference is that the <columns> element is renamed to <wpcolumns>.

csymyfiles_classic

**Purpose**
Displays a list of the user’s files that are in change sets. This component extends myfiles_classic, page 210.

**Parameters**
This component inherits the parameters of myfiles_classic, page 210.
Elements

This component has configurable columns (see the webcomponent layer’s doclist component). The only difference is that the <columns> element is renamed to <wpcolumns>.

cssubscriptions_classic

Purpose

Extends subscriptions_classic, page 243 and displays a list of the user’s subscribed files that are in change sets. This component is contained within changesetaddfilescontainer, page 171.

Parameters

This component inherits the parameters of subscriptions_classic, page 243, if any

elements

This component has configurable columns (see the webcomponent layer’s doclist component). The only difference is that the <columns> element is renamed to <wpcolumns>.

deeppromote and container

Purpose

Displays a UI to promote the contents of a channel or channel folder. This component and its container is launched by the action wpdeeppromote, page 151.
Parameters

The component has no defined parameters.

**delete and container**

**Purpose**

Displays a UI to delete an object. The delete component is called by the action `delete`, page 115 for `dm_document`, `dm_category`, `wcm_channel`, and `wcm_channel_fld` objects. The container `deletecontainer` extends the WDK layer's `wizardcontainer` component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the object to be deleted</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder that contains the object</td>
</tr>
</tbody>
</table>

**delete_locale**

**Purpose**

Displays a UI to delete a locale. This component is called by the action `delete` (locale), page 116 in the component toolbar, page 250.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the locale to be deleted</td>
</tr>
</tbody>
</table>

**demote and container**

**Purpose**

Displays a UI to demote an object. The demote component is called by the WP demote action for `dm_sysobjects` and `wcm_change_set` objects and their subtypes. The container `demotecontainer` extends the WDK layer's `container` component.

**Parameters**

The component has no defined parameters.
editionlist

**Purpose**
Extends the Webtop layer’s objectlist component and displays the folders and files in an edition.

**Parameters**
Inherits the parameters of the Webtop layer’s objectlist component.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component).

editionslist

**Purpose**
Displays a list of editions for the Web site. This component is included within the siteoverview, page 234 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Object ID of the site. This parameter is supplied by the siteoverview component.</td>
</tr>
<tr>
<td>numDisplay</td>
<td>Number of editions to display in the list. The remaining editions will be displayed by the More link. This value overrides the value in &lt;num_editions_to_display&gt;.</td>
</tr>
</tbody>
</table>

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;num_editions_to_display&gt;</td>
<td>Sets the base number of editions to display. This setting can be overridden by calling the component with the numDisplay parameter.</td>
</tr>
</tbody>
</table>
edit_manual_link_type

Purpose
Allows the user to edit a supported type of manual link to the application. This component is jumped to from the component list_manual_link_component, page 205.

Parameters

| relation_name | Required. Specifies the relation to be edited. |

editoptions

Purpose
Displays a UI that allows the user to select the action to perform the second or Nth time a user edits a document. Options displayed: Submit for Approval (Start workflow), Power Promote, Work on Later (cancel), View More Actions.

Parameters

| objectId | ID of the selected object. Either objectId or chronicleId must be supplied. |
| chronicleId | Object ID of the original version of the object. Either objectId or chronicleId must be supplied. |

Elements

| <options> | Contains one or more <option> elements. Each element is displayed as a radio button. |
| <option> | Each option can define one or two actions to be performed after new content is created. The value of the action1 and action2 attribute must be the name of a valid action for the application. One of the options should have the value of the default attribute set to true to make it the default action. |
| <arguments> | Contains one or more argument elements. |
| <argument> | Specifies an argument name and attribute. The component class looks up the attribute value for the object and passes it to the selected action. |
**editorpresentations**

**Purpose**
Displays a databound list of presentations that are managed by the Web Publisher editor. This component extends the webcomponent layer’s doclist component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>The initial folder that is displayed</td>
</tr>
<tr>
<td>folderPath</td>
<td>The initial folder path that is displayed. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
<tr>
<td>nodeIds</td>
<td>A dot-separated list of object IDs used with virtualpath and components to display a breadcrumb to the current component</td>
</tr>
<tr>
<td>virtualpath</td>
<td>Specifies the virtual breadcrumb path to child components, for example, a/b/c</td>
</tr>
<tr>
<td>components</td>
<td>Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c</td>
</tr>
</tbody>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component).

**expirenow and container**

**Purpose**
Expires an object immediately. The container extends the basic WDK layer’s container component and is launched by the action `expirenow`, page 118.

**Parameters**
The component has no defined parameters. The container performs the expiration on the object whose ID is passed into the expirenow action.
exportedition

**Purpose**
Exports an edition to the Site Caching Services target machine for browsing. This component is launched within the addeditioncontainer by the action `exportedition`, page 119.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the edition to be exported.</td>
</tr>
<tr>
<td>objectName</td>
<td>Name of the edition.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder containing the edition.</td>
</tr>
</tbody>
</table>

**Usage**
The site publishing configuration must have All Versions as the version label in order to be exported.

externalpresentations

**Purpose**
Extends the webcomponent layer's doclist component and displays a databound list of presentations that are managed by an external editor.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>The initial folder that is displayed</td>
</tr>
<tr>
<td>folderPath</td>
<td>The initial folder path that is displayed. This path must be a complete path from the Docbase root, for example, <code>/Documentation%20Library/subfolder</code></td>
</tr>
<tr>
<td>nodeIds</td>
<td>A dot-separated list of object IDs used with virtualpath and components to display a breadcrumb to the current component</td>
</tr>
<tr>
<td>virtualpath</td>
<td>Specifies the virtual breadcrumb path to child components, for example, a/b/c</td>
</tr>
<tr>
<td>components</td>
<td>Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c</td>
</tr>
</tbody>
</table>
Elements

This component has configurable columns (see the webcomponent layer’s doclist component).

externalresources_settings

Purpose
Displays a page for configuring external resources: Documentum Administrator URL (attribute="dctm_admin_location"), CIS Host address (attribute="cps_host_addr"), CIS default node (attribute="cps_node_name"), CIS Formats (attribute="cps_formats"), and Engagement Server host address (attribute="engagement_server_host_addr"). The attributes that are displayed are attributes for the wcm_config object. This component is contained within settings, page 233.

Parameters
This component has no defined parameters.

fileformatcriteria

Purpose
Creates a report of file formats. This component includes the reportselection component. When the user selects the formats option in the reportselection UI, the fileformatcriteria component nests to fileformatreport, page 191.

Parameters
This component has the same parameters as alerts, page 164.

fileformatreport

Purpose
Displays a report of the file formats. This component is nested within the fileformatcriteria, page 191 component.

Parameters
This component has the same parameters as alerts, page 164.
fileformats_settings

**Purpose**
Displays a UI to allow an administrator to change the text extensions list, Web-based editable extensions list, Web-safe format extensions, and other format extensions. This component is contained within the settings, page 233 component.

**Parameters**
This component has no defined parameters.

filtercontainer

**Purpose**
Extends the WDK layer’s dialogcontainer and serves as a container for specialized filters. Filter menus are defined in the UI as filter.jsp files with menu items appropriate to the component in which they are displayed. This container is nested to from within several component classes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>Set to true to display the locale menu item</td>
</tr>
<tr>
<td>status</td>
<td>Set to true to display the change set status menu item</td>
</tr>
</tbody>
</table>

finishwpworkflowtask

**Purpose**
Submits a Web Publisher workflow. This component is launched by the finishwpworkflowtask action.

**Parameters**
This component has no defined parameters.

**Elements**
This component has the same configurable elements as startwpwperformers, page 236.
foldermaps

Purpose
Extends the webcomponent layer’s doclist component and displays a list of all secondary foldermap files. This component is a node in the configurations component.

Parameters

| virtualpath | Sets the breadcrumb virtual path to child components, for example, a/b/c |
| components  | Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c |

Elements
This component has configurable columns (see the webcomponent layer’s doclist component).

forwardwpworkflowtask

Purpose
Extends the webcomponent layer’s forwardworkflowtask and forwards a Web Publisher workflow task to the next performer. This component is launched by the forwardwpworkflowtask, page 120 action.

Parameters
This component has no defined parameters.

Elements
See finishwpworkflowtask, page 192 for information on using dynamic performers.

general_preferences

Purpose
Extends the webcomponent layer’s general_preferences component and displays a UI in which the user can select a classic or streamline presentation and theme
to apply to the UI. This component is contained with the preferences, page 219 component.

**Parameters**

There are no parameters defined for this component.

### general_settings

**Purpose**

Displays a UI in which the user can set the following global application settings: Web cabinet protection, globalization, simple search, multisite effectivity, edition cache, default ACL, update creator, new object count, log level, enable Web-based editor, expiration increment, and renew notification. This component is contained within the settings, page 233 component.

**Parameters**

There are no parameters defined for this component.

### help-index

**Purpose**

Specifies context-sensitive map entries for help topics. For information about adding custom or localized help files to the application, see Web Development Kit and Client Applications Development Guide.

**Parameters**

This component has no defined parameters.

**Elements**

This component has the same configurable elements as the Webtop layer’s help-index component.

### hiddentool

**Purpose**

Provides an interface to set tracing parameters and configure the context sensitivity of actions based on views and role. This component is nested within the about, page 157 component. The hiddentool component is visible only to the user role specified in the about, page 157 component configuration.
### Parameters

| **tool** | Required. Specifies the tool start page to be displayed. This value is supplied by the button selection on the about UI. |

### Elements

| **<traceform>** | Specifies the JSP page to use to set trace flags |
| **<refreshform>** | Specifies the JSP page to use to refresh configuration files in memory |

### history

#### Purpose

Extends the webcomponent layer’s history component and displays an audit trail of all audited events (system and custom) for an object. This component is launched by the action `history, page 120` within the container `properties, page 221`. Provides link to workflow history if the object is in a workflow. Auditing can be turned on by executing the API audit method.

#### Parameters

| **objectId** | Required. Specifies the ID of the object for which to display history. |

#### Elements

| **<traceform>** | Specifies the JSP page to use to set trace flags |
| **<refreshform>** | Specifies the JSP page to use to refresh configuration files in memory |

### ice_custom_config

#### Purpose

This component has no UI. The component definition is used by several ICE components to provide various configuration settings.
### Parameters

This component has no defined parameters.

### Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;proxy_start_port&gt;</code></td>
<td>Defines the local proxy port to which the remote host is mapped. The port must be free on the user’s local machine.</td>
</tr>
<tr>
<td><code>&lt;editable_tags&gt;.&lt;tag&gt;</code></td>
<td>Specifies an HTML tag that surrounds editable content in the selected Web page. Default tags are: <code>&lt;a href&gt;</code>, <code>&lt;img src&gt;</code>, <code>&lt;span...&gt;</code>. Use <code>&lt;span...&gt;</code> tags to surround editable content.</td>
</tr>
<tr>
<td><code>&lt;span dctm-id='xyz'&gt;</code></td>
<td>Docbase object ID of the editable content. Overrides <code>&lt;span dctm-src&gt;</code></td>
</tr>
<tr>
<td><code>&lt;span dctm-src='xyz'&gt;</code></td>
<td>Relative file path of the editable file on the Web site.</td>
</tr>
<tr>
<td><code>&lt;span dctm-oper='operation_name'&gt;</code></td>
<td>Operation to be launched, such as a JavaScript method, for example, <code>add_content</code></td>
</tr>
<tr>
<td><code>&lt;span dctm-args= &gt;</code></td>
<td>Specifies a comma-delimited list of arguments to be passed to the operation named in <code>&lt;span dctm-oper&gt;</code>. For example, the <code>add_content</code> operation requires the template and category arguments.</td>
</tr>
<tr>
<td><code>&lt;span dctm-label=&gt;</code></td>
<td>Specifies the label to be displayed for a custom operation named in <code>&lt;span dctm-oper&gt;</code></td>
</tr>
<tr>
<td><code>&lt;convert_fqp_tags&gt;.&lt;tag&gt;</code></td>
<td>Specifies tags that should be converted to fully qualified paths, for example, <code>/img/logo.gif</code> is converted to <code>http://mysite.com/img/logo.gif</code></td>
</tr>
</tbody>
</table>
<job_interval>
Specifies the interval in seconds to run a job that checks autorender renditions and publishes out to the Web site.

<object_menu>
Contains a context-sensitive menu of actions (specified within the <actions> element). The menu for a custom tag is displayed if the tag attribute inline=false.

<actions>.<action>
Contains an action that is displayed in the ICE context-sensitive menu. Set the inline attribute to true to display the action. Set to false if there is no action defined for the custom operation (the operation will not be displayed).

<action>.<name>
Specifies the name of the action.

<action>.<label>
Specifies a label for the action, to be displayed in the menu.

<action>.<class>
Specifies the action class.

Usage

To add an editable HTML tag, add the tag to the <editable_tags> element. For example, to add the <body background=...> tag, you enter the following elements under <editable_tags>:

<tag>
  <name>body</name>
  <attribute>background</attribute>
</tag>

To add an editable custom tag, add the tag as a <span> tag. For example,

<tag>
  <name>
  </name>
</tag>

To launch an operation such as add_content, which generates a link to add content to the page, use the span tag similar to the following:

<span dcm-operation="add_content" dcm-args="template_id={DOCUMENT('/Web Publisher Configuration/Content Templates/path_to_template/template_filename')} &category_id={FOLDER('/Web Publisher Configuration/Content Templates/path_to_category')}" dcm-label="hyperlink_text">
</span>

See Web Publisher Administration Guide for more details on operations.
**iceconsole**

**Purpose**
Displays a toolbar with buttons that launch the following in-context editor actions or components: `iceeditpage`, page 198, `iceeditcomponent`, page 198, `iceproperties`, page 199.

**Parameters**
This component has no defined parameters.

**icedummy**

**Purpose**
Nests to the ICE component named as the value of the component parameter and then returns to the ICE frameset. This component is nested to from `icestatusbar`, page 199.

**Parameters**

| container | Specifies a container that is launched by the icedummy component |
| component | Specifies a container that is launched by the icedummy component |

**iceeditcomponent**

**Purpose**
This component is called from the ICE toolbar. Its UI page `iceoperationstart.jsp` posts the server event `onActionExecute`, which opens a Web page for editing.

**Parameters**

| objectId | ID of the object to be edited |

**iceeditpage**

**Purpose**
This component is called from the `iceconsole`, page 198 component to edit a Web page in context.
icelist

**Purpose**
This component is called from the siteoverview, page 234 component to turn in-context editing on or off.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the object to be edited</td>
</tr>
</tbody>
</table>

iceproperties

**Purpose**
This component is called from the iceconsole, page 198 component. The component nests to the properties component to display the properties of the Web page object in the Docbase.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the delivery channel</td>
</tr>
<tr>
<td>url</td>
<td>Not used</td>
</tr>
<tr>
<td>webcacheId</td>
<td>ID of the site</td>
</tr>
</tbody>
</table>

icestatusbar

**Purpose**
Extends statusbar, page 242 and displays messages and task status within the ICE editor.. This component is called from the statusbar, page 242 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the delivery channel</td>
</tr>
<tr>
<td>url</td>
<td>Not used</td>
</tr>
<tr>
<td>webcacheId</td>
<td>ID of the site</td>
</tr>
</tbody>
</table>
icetitlebar

Purpose
Extends the Webtop layer’s titlebar component and nests to the properties component to display a title for the ICE editor. This component is called from the iceeditpage, page 198 component.

Parameters
Inherits the parameters of the Webtop layer’s titlebar component, if any.

importrenditionand container

Purpose
Extends the DAM importrendition component and imports supporting a rendition into the Docbase. This component is called with its container by the action importrendition, page 122. The container extends the DAM importrenditioncontainer.

Parameters
This component inherits the parameters of importrenditionand container, page 200.

importsupportfiles

Purpose
Imports supporting files into a change set.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csFolderId</td>
<td>ID of the change set folder</td>
</tr>
</tbody>
</table>

Elements
This component has configurable columns (see the webcomponent layer’s doclist component) with the following additional element:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;objecttype&gt;</td>
<td>Specifies the object type to be displayed by default</td>
</tr>
</tbody>
</table>
inboxclassic

**Purpose**  Extends the Webtop layer’s inboxclassic component and displays the inbox in classic view. This component is a node in the browsertree, page 168 component.

**Parameters**

Inherits the parameters of the Webtop layer’s inboxclassic.

**Elements**

This component has the same configurable parameters as the webcomponent layer’s inboxdrilldown.

inboxstreamline

**Purpose**  Extends the Webtop layer’s inboxstreamline component and displays the inbox in streamline view. This component is a tab in the tabbar, page 247 component.

**Parameters**

Inherits the parameters of the Webtop layer’s inboxstreamline component.

**Elements**

This component has the same configurable parameters as the webcomponent layer’s inboxdrilldown, with the .

indicate_wpview

**Purpose**  This component indicates the target application for online help.

**Parameters**

This component has no defined parameters.
instructionslist

Purpose
Extends the webcomponent layer’s doclist component and displays a list of instructions files that are used to update XML content. This component is launched when a user navigates to an instructions folder.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>Specifies the ID of the folder containing instructions files.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Specifies the path to the folder containing instructions files. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
<tr>
<td>nodeIds</td>
<td>A dot-separated list of object IDs used with virtualpath and components to display a breadcrumb to the current component</td>
</tr>
<tr>
<td>virtualpath</td>
<td>Specifies the virtual breadcrumb path to child components, for example, a/b/c</td>
</tr>
<tr>
<td>components</td>
<td>Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c</td>
</tr>
</tbody>
</table>

Elements

This component has configurable columns. See the webcomponent layer’s doclist component for details.

invalidobjectscriteria

Purpose
Displays a UI for setting criteria to be used for a report of invalid objects in the delivery cabinet. The user can select a cabinet or a lifecycle. This component is called from an option in the reportselection, page 229 UI.

Parameters

This component has the same parameters as alerts, page 164.
invalidobjectsreport

**Purpose**
Extends Webtop objectlist, page 218 and displays a report of invalid objects in the selected delivery cabinet or lifecycle. This component is nested within the invalidobjectscriteria, page 202 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cabinetId</td>
<td>ID of the selected cabinet</td>
</tr>
<tr>
<td>lifecycleId</td>
<td>ID of the selected lifecycle</td>
</tr>
</tbody>
</table>

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component).

**jobcriteria**

**Purpose**
Displays a UI for setting criteria to be used for a report of jobs in the delivery cabinet. The user can select a cabinet or a lifecycle. This component is called from an option in the reportselection, page 229 UI.

**Parameters**
This component has the same parameters as alerts, page 164.

**jobreport**

**Purpose**
Displays a report of jobs in the selected delivery cabinet or lifecycle.

**Parameters**
This component has the same parameters as alerts, page 164.
latestalerts

Purpose Displays the more recent Content Server system alerts and displays a button to display all alerts.

Parameters

The parameters are the same as those for alerts, page 164.

lifecycleassetlocator and container

Purpose Extends wplifecyclelocator and container, page 276 to display protected assets. The container extends the wplifecyclelocatorcontainer and is nested within the protectedassets, page 224.

Parameters

The parameters are the same as those for wplifecyclelocator and container, page 276.

Elements

The configurable element is the same as wplifecyclelocator and container, page 276.

lifecyclereport

Purpose Extends the Webtop layer’s objectlist component and displays a report of the lifecycles in the selected delivery cabinet or lifecycle, including state, start date, task, current user, and date received. This component is nested within the lifecyclereportcriteria, page 205 component.

Parameters

This component has the same parameters as alerts, page 164.
Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;objecttype&gt;</td>
<td>Object type to display by default (default = dm_document)</td>
</tr>
<tr>
<td>&lt;showfilters&gt;</td>
<td>Boolean flag to display object type dropdown list</td>
</tr>
</tbody>
</table>

lifecyclereportcriteria

**Purpose**  
Displays a UI for setting criteria to be used for a report of lifecycles in the delivery cabinet, by template or initiator. This component is called from an option in the reportselection, page 229 UI.

**Parameters**  
This component has the same parameters as alerts, page 164.

lifecycles_settings

**Purpose**  
Displays a UI for creating a list of lifecycles in the Docbase, editing lifecycle state aliases, and setting a delay for publishing that is handled by a Monitor Lifecycles job configured in Developer Studio.

**Parameters**  
This component has no defined parameters.

list_manual_link_component

**Purpose**  
Extends the Webtop layer’s objectlist component and displays a UI that lists the manual links to an object. This component is called by the list_manual_link, page 123 action.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Selected object for which links will be displayed.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Folder path that is used in the breadcrumb. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
<tr>
<td>relationType</td>
<td>Type of relation for the link</td>
</tr>
</tbody>
</table>

Elements

This component has configurable columns (see the webcomponent layer’s doclist component). The following additional configurable element is available:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objecttype</td>
<td>Object type to display by default</td>
</tr>
</tbody>
</table>

list_related_content

Purpose

Extends the WDK layer’s propertiesheetcontainer component and contains the contains_content_list, page 178 and contained_in_content_list, page 177 components. This component is called by the list_related_content, page 124 action.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Selected object.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Folder path that is displayed in the breadcrumb. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
</tbody>
</table>

Elements

This component has configurable columns (see the webcomponent layer’s doclist component). The following additional configurable element is available:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objecttype</td>
<td>Object type to display by default</td>
</tr>
</tbody>
</table>
listobjectsreport

**Purpose**
Extends the Webtop objectlist, page 218 component and displays a list of objects in the delivery cabinet. The objects that are listed are determined by the set of criteria passed in from the calling component: This component is nested within the stagedobjectscriteria, page 235 and activeobjectscriteria, page 158 components.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cabinetId</td>
<td>ID of the delivery cabinet</td>
</tr>
<tr>
<td>user_name</td>
<td>Name of the user to whom the active objects belong</td>
</tr>
<tr>
<td>startdate</td>
<td>Used by the abouttoactivecriteria, page 157, abouttoexpirecriteria, page 157, and promotetoactivecriteria, page 221 components to set a start date for the query</td>
</tr>
<tr>
<td>enddate</td>
<td>Used by the abouttoactivecriteria, page 157, abouttoexpirecriteria, page 157, and promotetoactivecriteria, page 221 components to set an end date for the query</td>
</tr>
<tr>
<td>report_component</td>
<td>Specifies the type of report to generate. Valid values: activeobjectscriteria, stagedobjectscriteria, promotetoactivecriteria, abouttoactivecriteria, abouttoexpirecriteria</td>
</tr>
<tr>
<td>report_title</td>
<td>Overrides the properties string for the report title and sets the report window title</td>
</tr>
</tbody>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component).

localelist

**Purpose**
This component displays locale settings for a Web site.
**Parameters**

This component has no defined parameters.

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component) with the following additional element:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;objecttype&gt;</code></td>
<td>Specifies the object type to be displayed by default</td>
</tr>
</tbody>
</table>

**login**

**Purpose**

Extends the Webtop layer’s login component and displays a login page. The login component is called by the component dispatcher if a component is called without a valid session.

**Parameters**

This component inherits the parameters from the WDK layer’s login component.

**Elements**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;inspector_wprole&gt;</code></td>
<td>Login by a user having the specified role will trigger the system sanity check</td>
</tr>
<tr>
<td><code>&lt;admin_group&gt;</code></td>
<td>System sanity check determines whether the specified role is installed correctly in the Docbase</td>
</tr>
<tr>
<td><code>&lt;web_developer_group&gt;</code></td>
<td>System sanity check as above</td>
</tr>
<tr>
<td><code>&lt;content_manager_group&gt;</code></td>
<td>System sanity check as above</td>
</tr>
<tr>
<td><code>&lt;content_author_group&gt;</code></td>
<td>System sanity check as above</td>
</tr>
</tbody>
</table>

**logoff**

**Purpose**

Extends the Webtop layer’s logoff component and displays a Login Again link. The logoff component invalidates the user’s session.
**Parameters**

This component has no defined parameters.

**main**

**Purpose**

Extends the Webtop layer’s main component and includes the WP JavaScript file `main.js` to open content in a new window.

**Parameters**

This component inherits the parameters of the Webtop layer’s main component.

**Elements**

This component inherits the configurable elements of the Webtop layer’s main component.

**manuallinktypelist**

**Purpose**

Lists the supported types of manual link to the application. Manual link types correspond to a `dm_relation_type` in the Docbase. This component is called as a node in the `adminconsole, page 162` component.

**Parameters**

This component has no defined parameters.

**Elements**

This component has configurable columns (see the webcomponent layer’s `doclist` component). The following additional configurable element is available:

| objecttype          | Object type to display by default |

**menubar**

**Purpose**

Extends the Webtop menubar component and adds Web Publisher menu items.
**message**

**Purpose**
For a portlet, displays an error message in a dialog and closes the window.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>If true, the script to close the window automatically is disabled</td>
</tr>
</tbody>
</table>

**multisite_effectivity and container**

**Purpose**
Displays effectivity and expiration dates for multiple sites. This component is nested to from `publishing, page 225`. The container multisite_effectivity_container extends the WDK layer’s dialogcontainer.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected site.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the site folder to display in the breadcrumb</td>
</tr>
</tbody>
</table>

**myfiles_classic**

**Purpose**
Displays the user’s files. This component is called as a node in the `browsertree, page 168`.

**Parameters**

This component has no defined parameters.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component). The only difference is that there are two sets of columns defined: the `<columns>` element defines the columns that are displayed for Web Publisher objects,
and `<wpcolumns>` defines the columns that are displayed for non-WP objects. Additional configurable elements are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;modifiedwithindays&gt;</code></td>
<td>Criterion for display: Maximum number of days since the user modified a file</td>
</tr>
<tr>
<td><code>&lt;createdwithindays?&gt;</code></td>
<td>Criterion for display: Maximum number of days since a file was created by the user</td>
</tr>
<tr>
<td><code>&lt;wpcontext_content&gt;</code></td>
<td>Sets the permitted actions on WP objects. See <code>contextsensitiveview_config</code>, page 181 for details.</td>
</tr>
<tr>
<td><code>&lt;default_wpcontext&gt;</code></td>
<td>Sets the permitted actions on objects other than WP content. See <code>contextsensitiveview_config</code>, page 181 for details.</td>
</tr>
<tr>
<td><code>&lt;filter&gt;</code></td>
<td>Filters actions permissible on WP objects by user role (see Configurable Roles, page 180 for details).</td>
</tr>
<tr>
<td><code>&lt;wpcontext&gt;</code></td>
<td>Sets the action context for the role specified in the filter</td>
</tr>
</tbody>
</table>

### myfiles_streamline

**Purpose**

Displays the user’s files. This component is called as a tab in the `tabbar`, page 247.

**Parameters**

This component has no defined parameters.

**Elements**

This component has the same configurable elements as `myfiles_streamline`, page 211.

### newcategory and container

**Purpose**

Displays a UI for creating a new category channel. This component is contained within `newcategorycontainer`, page 212. The container and component are called by the `newcategory`, page 126 action in the UI of `menubar`, page 209 and `category_streamline`, page 170.
newcategorycontainer

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. Selected object for which to create a new category.</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderPath</td>
<td>Path to the selected object.</td>
</tr>
</tbody>
</table>

Elements

| basetype               | Valid default type for the new channel.                  |

newcategorycontainer

Purpose

Extends the webcomponent layer’s newcabinetcontainer component and contains components that create a new category channel.

Parameters

In addition to the parameters for newcategory and container, page 211, the following parameter is defined:

Elements

| component | Specifies the default component to be displayed within the container |

newcategoryfolder

Purpose

Extends the webcomponent layer’s newfolder component and displays a UI for creating a new category folder, either protected or non-protected. This component is contained within newcategoryfoldercontainer, page 213. The container and its component are called by the newcategoryfolder, page 126 action.

Parameters

| objectId | Required. ID for the selected channel that will contain the category |

212 Web Publisher Development Guide
newcategoryfoldercontainer

**Purpose**
Extends the webcomponent layer’s newfoldercontainer component and contains the following components: newcategoryfolder, page 212, attributes (dm_document, dm_category), page 166, and the webcomponent layer’s permissions component.

**Parameters**
This component inherits its parameter from the webcomponent layer’s newfoldercontainer component.

newchangeset

**Purpose**
Creates a new change set. This component is called by the action newchangeset, page 127 in the component UIs of menubar, page 209 and toolbar, page 250.

**Parameters**
This component has no defined parameters.

newchannel

**Purpose**
Creates a new channel. This component is called by the action newchangeset, page 127 in the component UIs of menubar, page 209 and toolbar, page 250.

**Parameters**
This component has no defined parameters.

**Elements**

| basetype | Valid default type for the new channel. |

newchannelcontainer

**Purpose**
Extends the webcomponent layer’s newcabinetcontainer component and contains several components that create a new channel. This component is called by the
newchannelfolder

Parameters

This component has no defined parameters.

Elements

<table>
<thead>
<tr>
<th>basetype</th>
<th>Valid default type for the new channel.</th>
</tr>
</thead>
</table>

newchannelfolder

Purpose

Extends the webcomponent layer’s newfolder component and displays a UI for creating a new channel. This component is contained within newchannelfoldercontainer, page 214. The container and its component are called by the newchannelfolder, page 127 action in the component UIs of menubar, page 209 and channel_streamline, page 175.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID for the selected channel that will contain the channel folder</th>
</tr>
</thead>
</table>

Elements

<table>
<thead>
<tr>
<th>basetype</th>
<th>Valid default type for the new channel folder.</th>
</tr>
</thead>
</table>

newchannelfoldercontainer

Purpose

Extends the webcomponent layer’s newfoldercontainer component and contains components for creating a new channel folder. The container is called by the newchannelfolder, page 127 action in the component UIs of menubar, page 209 and channel_streamline, page 175.

Parameters

This component inherits the parameters of the WDK layer’s combocontainer.
newchannelpublish

**Purpose**
Publishes a new channel. This component is called by the action `newchannel`, page 127 in the newchannelcontainer.

**Parameters**
This component has no defined parameters.

newcontent and container

**Purpose**
Displays a UI for creating a new content file. The container extends the WDK layer's dialogcontainer. The component and container are called by the `newcontent`, page 128 action.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>ID of the new object</th>
</tr>
</thead>
</table>

**Elements**

| `<allow_import>` | Set to true to allow the user to import the new content from the file system |
| `<enable_full_text_index>` | Boolean flag that specifies whether to enable full-text indexing on the document |

newcontentmanager

**Purpose**
Displays a UI for creating a new content file in a portal. This component is launched by the DocumentumMySubmitContent portlet.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>ID of the new object</th>
</tr>
</thead>
</table>
newcontentoptions

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;allow_import&gt;</td>
<td>Set to true to allow the user to import the new content from the file system</td>
</tr>
<tr>
<td>&lt;enable_full_text_index&gt;</td>
<td>Boolean flag that specifies whether to enable full-text indexing on the document</td>
</tr>
</tbody>
</table>

newcontentoptions

Purpose

Displays a UI in a portal application that allows the user to select the action to perform after creating new content.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the new object.</td>
</tr>
<tr>
<td>templateId</td>
<td>Required. Object ID of the template for the new content.</td>
</tr>
</tbody>
</table>

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;options&gt;</td>
<td>Contains one or more &lt;option&gt; elements. Each element is displayed as a radio button.</td>
</tr>
<tr>
<td>&lt;option&gt;</td>
<td>Each option can define one or two actions to be performed after new content is created.</td>
</tr>
<tr>
<td>&lt;arguments&gt;</td>
<td>Contains the argument and action attributes and the names of the options. If an option is selected, it will have the value of the default attribute set to true to make it the default action.</td>
</tr>
<tr>
<td>&lt;argument&gt;</td>
<td>Specifies an argument name and attribute. The component class looks up the attribute value for the object and passes it along with the argument name to the selected action. The object ID is added by default.</td>
</tr>
</tbody>
</table>

newfolder

Purpose

Creates a new folder. This component is called by the newfolder action in the UIs of menubar, page 209 and channel_streamline, page 175.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the folder or cabinet that will contain the new folder</td>
</tr>
</tbody>
</table>

**newtaxonomy**

**Purpose**

Creates a new taxonomy within a category. This component and the newtaxonomycontainer are called by the action newtaxonomy_, page 128 in the component UIs of menubar, page 209 and category_streamline, page 170.

**Parameters**

This component has no defined parameters.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basetype</td>
<td>Valid default type for the new taxonomy.</td>
</tr>
</tbody>
</table>

**newtaxonomycontainer**

**Purpose**

Extends the webcomponent layer's newcabinetcontainer component and contains components that create a new taxonomy within a category. This container is called by the action newtaxonomy_, page 128 in the component UIs of menubar, page 209 and category_streamline, page 170.

**Parameters**

This container inherits the parameters of the WDK layer's combocontainer.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basetype</td>
<td>Valid default type for the new taxonomy.</td>
</tr>
</tbody>
</table>
objectlist

Purpose
Extends the DAM objectlist component (which extends the Webtop layer’s objectlist component). The objectlist component is extended by several other components that display specific types of objects.

Parameters
This component inherits the parameters of the DAM component objectlist.

Elements
This component has configurable columns inherited from DAM (see the webcomponent layer’s doclist component).

portletnewcontentcontainer

Purpose
Extends the container newcontent and container, page 215. This container is nested to from newcontentmanager, page 215 in a portal environment.

Parameters
This container inherits parameters from the container newcontent and container, page 215.

powerpromote and container

Purpose
Advances the selected items to the approved state. The powerpromotecontainer extends the WDK layer’s container component. The component and container are called by the powerpromote, page 129 action.

Parameters
The component has no defined parameters. The container inherits parameters from the WDK layer’s container component.
preferences

Purpose
Extends the WDK layer’s propertiesheetcontainer component and contains preferences components general_preferences, web_author_preferences, and web_developer_preferences. This component is called as a node in the the browsertree, page 168 component.

Parameters
This component inherits the parameters of the WDK layer’s container component.

Elements
This component is configured by default to filter the preferences components that are displayed in the container. The filter is specified by the <filter> element, and the wprole attribute on the filter specifies that the component within the filter element will be displayed only when the user has the role. For example, the container is configured by default such that the wprole value is web developer. The component within the filter, web_developer_preferences, will be displayed only to users who have the role web developer.

presentations

Purpose
Displays a UI that allows the user to add or remove presentations to the list of available presentations for a selected document. This component is called by presentations, page 129.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected object.</th>
</tr>
</thead>
</table>

Elements

<table>
<thead>
<tr>
<th>&lt;wpcontext_editor&gt;</th>
<th>Sets the context for files that are edited using the WP editor (used by the whereused component)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext_external&gt;</td>
<td>Sets the context for files that are edited using an external editor (used by the whereused component)</td>
</tr>
</tbody>
</table>
presentationsfolder

**Purpose**
Extends sitemanager, page 234 and displays the contents of the presentations folder. This component is called from a node in the sitemanager, page 234 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>The initial folder that is displayed</td>
</tr>
<tr>
<td>folderPath</td>
<td>The initial folder path that is displayed. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
<tr>
<td>nodeIds</td>
<td>A dot-separated list of object IDs used with virtualpath and components to display a breadcrumb to the current component</td>
</tr>
<tr>
<td>virtualpath</td>
<td>Specifies the virtual breadcrumb path to child components, for example, a/b/c</td>
</tr>
<tr>
<td>components</td>
<td>Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c</td>
</tr>
</tbody>
</table>

**Elements**
This component has configurable nodes (see the webcomponent layer’s usermanagement component).

previews

**Purpose**
Extends the webcomponent layer’s doclist component and displays a list of previews available for a selected document. This component is called from a node in the sitemanager, page 234 component.

**Parameters**
The parameters are the same as those for presentationsfolder, page 220.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component).
promote and container

**Purpose**
Promotes the selected object to the next phase in the attached lifecycle. This component and container are called by the promote, page 129 action. The component has no UI.

**Parameters**
The component has no defined parameters.

promotetoactivecriteria

**Purpose**
Displays a UI in which the user can select start and end dates for promotion of a selected document. This component is called as an option in the reportselection, page 229 component UI.

**Parameters**
The component has the same parameters as alerts, page 164.

properties

**Purpose**
Extends the WDK layer’s propertysheetcontainer component and displays the properties of an object. This component is a container and is called by the properties, page 130 action.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object for which properties will be displayed</td>
</tr>
<tr>
<td>folderPath</td>
<td>Specifies the path that should be displayed in the breadcrumb. Must be a complete path from the Docbase root, for example, /Doc%20Library/subfolder</td>
</tr>
<tr>
<td>component</td>
<td>Specifies the component to be displayed by default</td>
</tr>
</tbody>
</table>
Elements

The contained components differ depending on the scoped properties component definition. The unscoped definition (scope = *) contains the attributes, publishing, permissions, subscriptions, and history components. The definitions that are scoped to specific object types extend the base WP properties component and contain various components.

properties (dm_document)

Purpose

Extends properties, page 221 and displays the properties of a document. This component is a container and is called by the properties, page 130 action.

Parameters

The parameters are inherited from properties, page 221

Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;componentfilterlist&gt;</td>
<td>Contains a set of filters (&lt;componentfilter&gt;) that hide components when they are not appropriate for a document type</td>
</tr>
<tr>
<td>&lt;componentfilter&gt;</td>
<td>Defines the filter that hides components for a specified document type</td>
</tr>
<tr>
<td>&lt;objectkind&gt;</td>
<td>Type of document. Valid values are defined as members of WcmObjectKind.</td>
</tr>
<tr>
<td>&lt;hidecomponents&gt;</td>
<td>Contains &lt;component&gt; elements that specify components to be hidden for the object kind</td>
</tr>
<tr>
<td>&lt;component&gt;</td>
<td>Must correspond to a component name in the application</td>
</tr>
</tbody>
</table>

properties (wcm_edition, wcm_locale)

Purpose

Displays the properties of an edition or locale. This component is a container and is called by the properties, page 130 action.
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the edition for which properties will be displayed</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder that contains the edition. This parameter is used to update the breadcrumb path. (wcm_edition only)</td>
</tr>
<tr>
<td>folderPath</td>
<td>Specifies the path that should be displayed in the breadcrumb. Must be a complete path from the Docbase root, for example, /Doc%20Library/subfolder</td>
</tr>
</tbody>
</table>

**properties (dm_document)**

**Purpose**

Extends properties, page 221 and displays the properties of a dm_document object. This component is a container and is called by the properties, page 130 action.

**Parameters**

This component inherits the parameters of properties, page 221.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;componentfilterlist&gt;</td>
<td>Contains &lt;componentfilter&gt; elements that define filters for hiding or displaying components based on the template type</td>
</tr>
<tr>
<td>&lt;componentfilter&gt;</td>
<td>Defines a filter that hides components for a selected template type. For example, the properties component for a rules template hides the publishing and categories tabs.</td>
</tr>
<tr>
<td>&lt;objectkind&gt;</td>
<td>Type of template. Valid values are static members of WcmObjectKind: CONTENT_TEMPLATE</td>
</tr>
<tr>
<td>&lt;hidecomponents&gt;</td>
<td>Specifies the components to hide in the properties container.</td>
</tr>
<tr>
<td>&lt;hidecomponents&gt;,&lt;component&gt;</td>
<td>Specifies the component to be hidden. Must be a component named in the &lt;contains&gt; element.</td>
</tr>
</tbody>
</table>
**propertiestransform**

**Purpose**
Displays a UI in which the user can change the following properties: title, subject, lifecycle, object type, template, locale, effective date, and location. This component is not used in the Web Publisher UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object for which properties will be displayed</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of a folder, which corresponds to the new folder location for the object</td>
</tr>
</tbody>
</table>

**protectedassetlist**

**Purpose**
Displays settings for protected assets in a selected Web cabinet: status, permissions, and properties. This component is included by the UI page of the siteoverview, page 234 component.

**Parameters**

This component has no defined parameters.

**protectedassets**

**Purpose**
Displays the properties of protected assets within the protectedassetsproperties, page 224 container for Web cabinets.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the Web cabinet</td>
</tr>
</tbody>
</table>

**protectedassetsproperties**

**Purpose**
Extends the WP component properties, page 221 and displays the properties of protected assets within a selected Web cabinet if site protection is turned on. This
component is nested within the properties action for objects of type wcm_channel (Web cabinets).

**Parameters**

This component inherits the parameters of properties, page 221.

**publicsupportingfiles**

**Purpose**

Displays a list of public supporting files. This component is contained with the changesetaddsupportingfilescontainer, page 171 component.

**Parameters**

This component has no defined parameters.

**Elements**

This component has configurable columns (see the webcomponent layer's doclist component).

**publish and container**

**Purpose**

Displays a UI to support the publishing of a Web file, folder, or cabinet. This component is called by the publish, page 131 action. The publishcontainer extends the the WDK layer's combocontainer component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder that contains the object to be published.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Path to the folder to be published</td>
</tr>
</tbody>
</table>

**publishing**

**Purpose**

Displays a UI in which the user can view and edit publishing settings for a selected object. This component is contained within the properties, page 221 container and is called from the component menubar, page 209.
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>contentType</td>
<td>Type of the selected object.</td>
</tr>
<tr>
<td>folderId</td>
<td>Folder that contains the selected object</td>
</tr>
</tbody>
</table>

For objects of type `wcm_channel`, only the `objectId` parameter is defined.

**publishingconfig**

**Purpose**
Displays a UI in which the user can view and edit the publishing configurations for a selected Web cabinet. This component is included within the `siteoverview`, page 234 UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the selected Web cabinet</td>
</tr>
</tbody>
</table>

**quicklinks**

**Purpose**
Displays links that perform the following management functions: globalization settings (*localelist*, page 207 component), turning globalization on and off, workflow templates (*wpworkflowtemplatelist*, page 288 component), and change sets (*changesets_list*, page 173 component). The `quicklinks` component is included within the `adminconsole`, page 162 UI.

**Parameters**

This component has the same parameters as the *alerts*, page 164 with the following additional parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentName</td>
<td>Specifies the component that will handle the globalization link</td>
</tr>
</tbody>
</table>

**readonlypermissions**

**Purpose**
Extends the webcomponent layer’s permissions component and displays a read-only view of permissions on objects.
**reinstate**

**Purpose**
Reinstates a non-current version of an object to the current version. This component is called by the reinstate, page 132 action.

**Parameters**
The parameters are the same as those for the reinstate, page 132 action.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;attributelist&gt;</td>
<td>Contains the set of attributes whose values will be copied to the new CURRENT version</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>Specifies an attribute name. Must match an attribute name for the object type in the Docbase.</td>
</tr>
</tbody>
</table>

**rejectwpworkflowtask**

**Purpose**
Extends the webcomponent layer’s rejectworkflowtask component. This component is called by the rejectwpworkflowtask, page 132 action.

**Parameters**
The parameter is the the same as that for the webcomponent layer’s rejectworkflowtask component.

**Elements**
The configurable elements are the same as those for finishwpworkflowtask, page 192.

**removedition**

**Purpose**
This component is called by the removeedition, page 133 action in the editionslist, page 187 component UI to remove an edition (archived Web site).
renditions

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected edition.</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectName</td>
<td>Required. Name of the selected edition, used to set the label in the UI.</td>
</tr>
</tbody>
</table>

**Purpose**

Displays the renditions for a selected object. This component is called by the renditions, page 134 action from the menubar, page 209 component.

**Parameters**

The parameters are the same as those for the renditions, page 228 component.

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component).

repeatwpworkflowtask

**Purpose**

Extends the webcomponent layer’s repeatworkflowtask component. This component is called by the repeatwpworkflowtask, page 135 action.

**Parameters**

The parameters are inherited from the webcomponent layer’s repeatworkflowtask component.

**Elements**

The configurable elements are the same as those for finishwpworkflowtask, page 192.

reports

**Purpose**

Displays a UI that allows the user to select a report on the Site. This component is called as a node in the adminconsole, page 162 component. The UI JSP page includes the reportselection, page 229 component.
reportselection

Purpose
Displays a selector for Site reports. This component is included within the UI JSP page of the reports, page 228 component.

Parameters
This component has the same parameters as alerts, page 164.

request_translation

Purpose
Displays a UI to request translation of a document. This component is called by the request_translation, page 135 action from the menubar, page 209 and toolbar, page 250 components.

Parameters
This component has the same parameters as request_translation, page 135.

rolemenu

Purpose
Regenerates the context-sensitive role list and reloads the configuration service files. Context-sensitive roles are described in Configurable Roles, page 180. This component is launched by user selection in the hiddentool, page 194 UI.

Parameters
This component has no defined parameters.

rulelist

Purpose
Extends the webcomponent layer’s doclist component and displays a list of rules in the Site rules folder and a subset of the rules attributes. This component is called from a node in the sitemanager, page 234 component.
Parameters

The parameters are the same as those for presentationsfolder, page 220.

Elements

This component has configurable columns (see the webcomponent layer’s doclist component).

rules

Purpose

Displays the rules associated with a selected object and allows the user to associate another rule. This component is called by the rules, page 137 in the menubar, page 209 component and is also included in the associations, page 166 container.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>ID of the selected object</th>
</tr>
</thead>
</table>

Elements

<table>
<thead>
<tr>
<th>&lt;rulesformats&gt;</th>
<th>Contains &lt;format&gt; elements for all supported rules file formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;format&gt;</td>
<td>Value must be a format in the Docbase. The default supported rules file formats are XML and HTML.</td>
</tr>
</tbody>
</table>

search

Purpose

Displays a UI to perform search queries and displays the search results. This component jumped to from the component UI of the Webtop layer’s titlebar component and icetitlebar, page 200.

Parameters

The parameters are the same as those for the webcomponent layer’s search component, with the following additional parameters:

| wpcontext       | Specifies the set of actions permissible on objects in the search results (see contextsensitiveview_config, page 181 for details). |
Elements

The configurable elements are the same as those for the webcomponent layer’s search component, with the following additional elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext_nonwp&gt;</td>
<td>Specifies the view context for non-WP objects</td>
</tr>
<tr>
<td>&lt;columns_list&gt;</td>
<td>Contains &lt;column&gt; elements that correspond to columns of results that will be displayed</td>
</tr>
<tr>
<td>&lt;filter&gt;</td>
<td>Can be placed around an element that will be shown only in the specified context. For example, in the default configuration the column for the language_code attribute is displayed only when the context is wpsearch.</td>
</tr>
<tr>
<td>&lt;where_attribute_list&gt;</td>
<td>Contains &lt;attribute&gt; elements that will be used in the search when full-text search is not specified</td>
</tr>
<tr>
<td>&lt;attribute&gt;.&lt;name&gt;</td>
<td>Corresponds to the name of an attribute in the Docbase</td>
</tr>
</tbody>
</table>

Usage

See Web Development Kit and Client Applications Development Guide for information and examples on configuring search.

searchcs

Purpose

Displays a UI to build a search query on change sets and displays the search results. This component is contained within the changesetaddfilescontainer, page 171 container.

Parameters

This component has no defined parameters.

Elements

This component has configurable columns (see the webcomponent layer’s doclist component) and the <attribute_where_list> elements from search, page 230.
searchsup

**Purpose**
Displays a UI to build a search query on change set supporting files and displays the search results. This component is contained within the `changesetaddfilescontainer`, page 171 container.

**Parameters**
This component has no defined parameters.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component) and the `<attribute_where_list>` elements from search, page 230.

selectactions

**Purpose**
Displays a UI in a portal application that allows the user to select an action to be performed.

**Parameters**

<table>
<thead>
<tr>
<th>objectld</th>
<th>Required. ID of the selected object.</th>
</tr>
</thead>
</table>

### Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;options&gt;</code></td>
<td>Contains one or more <code>&lt;option&gt;</code> elements. Each element is displayed as a radio button.</td>
</tr>
<tr>
<td><code>&lt;option&gt;</code></td>
<td>Each option can define one or two actions to be performed after new content is created. The value of the action attributes must be valid actions for the application. One of the options should have the value of the default attribute set to true to make it the default action.</td>
</tr>
<tr>
<td><code>&lt;arguments&gt;</code></td>
<td>Contains one or more <code>&lt;argument&gt;</code> elements. Argument names and values are passed to the action precondition and execution classes. The component class looks up the attribute value for the object and passes it along with the argument name to the selected action. The object ID is added by default.</td>
</tr>
<tr>
<td><code>&lt;argument&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

### sessionping

**Purpose**

Redirects to a Web Publisher application session. This component is called by the WP portlets open control [open, page 100](#).

**Parameters**

- **ticketurl**
  - Required. URL containing a Docbase login ticket.
- **desturl**
  - Required. URL containing the component to open.
- **docbase**
  - Target Docbase
- **username**
  - Name of current user

### settings

**Purpose**

This container component is a node in the adminconsole, page 162 component. The component extends the WDK layer’s propertySheetContainer component and contains four settings components: general, lifecycles, file formats, and external resources.
Parameters

This component has the same parameters as the alerts, page 164 component.

sitemanager

Purpose

This component is a node in the browsertree, page 168 component. The component displays links to the following components: templates, presentationfolder, rulelist, previews, configurations, instructionslist, and changesets_list.

Parameters

This component can be initialized in four ways:

- No parameters: Displays Docbase root cabinets
- folderId: Displays the primary location of the specified folder ID
- folderPath: Displays the specified folder
- nodeIds: Uses a list of ‘.’ separated object IDs that are used to build an absolute location to be displayed

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderId</td>
<td>The folder of the adminconsole in the browser tree</td>
</tr>
<tr>
<td>folderPath</td>
<td>The folder path to be displayed in the adminconsole. This path must be full, from the Docbase root, for example, /Docbase%20Admin</td>
</tr>
<tr>
<td>nodeIds</td>
<td>IDs that constitute the path to the adminconsole component in the browsertree</td>
</tr>
</tbody>
</table>

Elements

This component has configurable nodes (see the webcomponent layer’s usermanagement component) and columns (see the webcomponent layer’s doclist component).

siteoverview

Purpose

This component is called by the siteoverview, page 138 action in the menubar, page 209 component UI to display an overview of a selected Web cabinet.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the selected Web cabinet (wcm_channel object)</td>
</tr>
</tbody>
</table>
stagedobjectscriteria

**Purpose**  
Displays a UI for setting criteria to be used for a report of staged objects in the delivery cabinet. This component is called from an option in the reportselection, page 229 UI.

**Parameters**
This component has the same parameters as the alerts, page 164 component.

startwpwfattachment

**Purpose**  
Displays a list of attachments in a workflow process. Attachments can be added and removed through the UI. This component is included in the startwpworkflowclassic, page 240 component UI and the startwpworkflowstreamline, page 241 component UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the workflow process.</td>
</tr>
<tr>
<td>attachmentId</td>
<td>ID of attachment</td>
</tr>
</tbody>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component). Additionally, you can configure attributes of the attachments to be added to the query. These attributes will be displayed in the attachments list.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;attributes&gt;</td>
<td>Contains &lt;attribute&gt; elements used to build the query for attachment attributes to be displayed in the list</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>The value corresponds to a Docbase attribute of dm_sysobject</td>
</tr>
</tbody>
</table>

startwpwfccontainerclassic

**Purpose**  
Extends the webcomponent layer’s startwfcontainer component and contains the components that start a WP workflow: startwpworkflowclassic,
startwpwfcontainerstreamline

Parameters

| objectID | Required. ID of the workflow process |
| componentArgs | Required. This parameter is supplied by the LaunchComponent class. |
| component | Start component |
| attachmentId | ID of the attachment. Either an attachmentId or changeld parameter must be supplied. |
| changesetId | ID of the change set to which the workflow belongs, if any. Either an attachmentId or changeld parameter must be supplied. |
| type | Required. Object type of the attachment, passed to the contained components |
| skipWorkflowReadyCheck | Boolean parameter. Set to true to launch a workflow. |

startwpwfcontainerstreamline

Purpose

Extends startwpwfcontainerclassic, page 235 and contains the components that start a WP workflow: startwpworkflowstreamline, startwpwfperformers, startwpwfsupportingfilesstreamline, and startwpwfcomments. This container is called by the startwpworkflowstreamline, page 141 action.

Parameters

The parameters are inherited from startwpwfcontainerclassic, page 235.

startwpwfperformers

Purpose

Extends the webcomponent layer’s startwpwfperformers component and allows the user to select custom performer aliases for workflow performers. This component is contained within startwpwfcontainerclassic, page 235 and startwpwfcontainerstreamline, page 236 components.

Parameters

| objectID | Required. ID of the workflow process |
## Elements

This component can be configured to use dynamic performers in which the workflow initiator or designer must select a dynamic performer value for an alias. By default, Web Publisher displays all groups and all users as alias suggestion values. You can create your own alias suggestion values to display for any existing default or custom alias.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;performers&gt;</td>
<td>Contains the definition of custom performers in the following elements: &lt;aliasperformerfilters&gt;, &lt;aliasperformerinit&gt;, &lt;dynamicperformerfilters&gt;, &lt;dynamicperformerinit&gt;</td>
</tr>
<tr>
<td>&lt;aliasperformerfilters&gt;</td>
<td>Default alias values</td>
</tr>
<tr>
<td>&lt;aliasperformerfilters&gt;.&lt;class&gt;</td>
<td>Fully qualified filter class name that implements IAliasPerformerFilter. Can contain optional configuration information.</td>
</tr>
<tr>
<td>&lt;aliasperformerinit&gt;</td>
<td>Alias initializer classes</td>
</tr>
<tr>
<td>&lt;aliasperformerinit&gt;.&lt;class&gt;</td>
<td>Fully qualified initializer class name that implements IAliasPerformerInit.</td>
</tr>
<tr>
<td>&lt;aliasperformerinit&gt;.&lt;class&gt;.&lt;role&gt;</td>
<td>Specifies the minimum role the user must have in order to be assigned to the defined alias. Valid values: content author, content manager, web developer and administrator.</td>
</tr>
<tr>
<td>&lt;aliasperformerinit&gt;.&lt;class&gt;.&lt;aliasname&gt;</td>
<td>Users are assigned to the alias specified in this element when they have the role specified in the corresponding &lt;aliasperformerinit&gt;.&lt;class&gt;.&lt;role&gt;.</td>
</tr>
<tr>
<td>&lt;dynamicperformerfilters&gt;</td>
<td>Dynamic performer values</td>
</tr>
<tr>
<td>&lt;dynamicperformerfilters&gt;.&lt;class&gt;</td>
<td>Fully qualified filter class name that implements IDynamicPerformerFilter. Can contain optional configuration information.</td>
</tr>
<tr>
<td>&lt;dynamicperformerinit&gt;</td>
<td>Dynamic performers initializer classes</td>
</tr>
<tr>
<td>&lt;dynamicperformerinit&gt;.&lt;class&gt;</td>
<td>Fully qualified initializer class that implements IDynamicPerformerInit. Can contain optional configuration information.</td>
</tr>
</tbody>
</table>

## Usage

A dynamic workflow performer can be selected in the following ways

- The workflow initiator selects the actual performer.
A participant in the workflow specifies performers of future tasks. For information on configuring dynamic performers, see Chapter 8, Workflow Dynamic Performers.

**startwpwfsupportingfilesclassic**

**Purpose** Displays a list of supporting files for a workflow, allowing the user to add or remove supporting files. This component is contained within the startwpworkflowclassic, page 140 container.

**Parameters**

| objectId | Required. ID of the workflow process |

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component).

**startwpwfsupportingfilesstreamline**

**Purpose** Extends startwpwfsupportingfilesclassic, page 238 and displays a list of supporting files for a workflow, allowing the user to add or remove supporting files. This component is contained within the startwpworkflowclassic, page 140 container.

**Parameters**

| objectId | Required. ID of the workflow process |

**Elements**

This component has the configurable columns (see the webcomponent layer’s doclist component).
**startwpwftemplatelocatorcontainerclassic**

**Purpose**
Extends the webcomponent layer’s wftemplatelocatorcontainer component and contains one or more wpwftemplatelocator and container, page 287 components. This container is called by the startwpworkflownotemplate, page 139 action.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object Id</td>
<td>Required. ID of the selected attachment.</td>
</tr>
<tr>
<td>componentArgs</td>
<td>Required. This parameter is supplied by the LaunchComponent class.</td>
</tr>
<tr>
<td>attachmentMode</td>
<td>Specifies whether the attachment is an existing change set. Valid values = existingchangeset</td>
</tr>
</tbody>
</table>

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;startworkflowactionid&gt;</td>
<td>ID of the start workflow action to launch when user selects a template (onOK())</td>
</tr>
</tbody>
</table>

**startwpwftemplatelocatorcontainergenericclassic**

**Purpose**
Extends startwpwftemplatelocatorcontainerclassic, page 239 and starts a workflow with no template selected. This container can contain one or more wpwftemplatelocator and container, page 287 components. This component is called by the aclbrowse, page 107 action.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachmentMode</td>
<td>Specifies whether the attachment is an existing change set. Valid values = existingchangeset</td>
</tr>
</tbody>
</table>

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;startworkflowactionid&gt;</td>
<td>ID of the start workflow action to launch when user selects a template (onOK())</td>
</tr>
</tbody>
</table>
**startwpwftemplatelocatorcontainergenericstreamline**

**Purpose**
Extends `startwpwftemplatelocatorcontainerstreamline`, page 240 and starts a workflow with no template selected. This container can contain one or more `wpwftemplatelocator` and container, page 287 components. This component is called by `thestartwpworkflownotemplate`, page 139 and `startwpworkflownotemplategenericstreamline`, page 140 actions.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachmentMode</td>
<td>Specifies whether the attachment is an existing change set. Valid values = existingchangeset</td>
</tr>
</tbody>
</table>

**Elements**
This component inherits configurable elements from `startwpwftemplatelocatorcontainerstreamline`, page 240.

**startwpwftemplatelocatorcontainerstreamline**

**Purpose**
Extends the webcomponent layer’s `wftemplatelocatorcontainer` component and contains one or more `wpwftemplatelocator` and container, page 287 components. This container is called by the `startwpworkflownotemplategenericstreamline`, page 141 action.

**Parameters**
This component has the same parameters as `startwpwftemplatelocatorcontainerclassic`, page 239.

**Elements**
This component has the same configurable element as `startwpwftemplatelocatorcontainerclassic`, page 239.

**startwpworkflowclassic**

**Purpose**
Extends the webcomponent layer’s `startworkflow` component and displays information about a workflow. Includes the `startwpwfastachment`, page 235.
component. This component is called by the startwpworkflowclassic, page 140 action, which in turn is called in the request_translation component class.

Parameters

| objectId | Required. ID of the workflow process |

Elements

This component can be configured to specify the users who can and cannot change the supervisor of a workflow. The default is the template owner.

| <supervisor> | Contains settings for supervisor configuration. If empty, template owner is the default value. |
| <wprolepermittedtomodify> | Specifies the minimum role that is permitted to modify the supervisor. If the reverse attribute is set to true, the specified role will not be permitted to modify the supervisor. If null, content manager is the default value. |
| <filter> | Value of wprole attribute specifies the role that can change a workflow supervisor |
| <filter>.<defaultvalue> | Specifies the default role that can change a workflow’s supervisor. Valid values: template owner | workflow initiator. If null, template owner is the default value. Can contain value, or <user> or <alias> element. |
| <user> | Specifies a user who can change the supervisor of a workflow. Cannot be a group. Overrides <alias> value. |
| <alias> | Specifies an alias who can change the supervisor of a workflow. Is overridden by <user> value. |

startwpworkflowstreamline

Purpose

Extends startwpworkflowclassic, page 240 and displays information about a workflow. Includes the startwpfattachment, page 235 component. This component is called by the startwpworkflowstreamline, page 141 action and is contained within the startwpwfcontainerstreamline, page 236 container.
Parameters

The parameters are inherited from `startwpworkflowclassic, page 240`.

Elements

The elements are inherited from `startwpworkflowclassic, page 240`.

**statusbar**

**Purpose**

Extends the Webtop layer's statusbar component and adds a button to display task status.

**Parameters**

Inherits the parameters of the Webtop layer's statusbar component, if any.

**streamlineview**

**Purpose**

Extends the Webtop layer's streamlineview component and displays the streamline view of folders and content. This component is called from a tab in the tabbar component. This component is the default folder viewing component as defined in the component `main, page 209`.

**Parameters**

This component has no defined parameters.

**subscriptions**

**Purpose**

Displays a subscription UI for an object in which the user can subscribe, request notification on checkin, and unsubscribe to an object. This component is a node in the properties, page 221 component.

**Parameters**

This component has no defined parameters.
subscriptions_classic

**Purpose**  
Displays the user’s subscribed files and objects. This component is a node in the browsertree, page 168 component.

**Parameters**

This component has no defined parameters.

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component). Two types of columns are configurable: one for Web Publisher objects, one for non-WP objects. Additionally, you can configure the following elements:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext_content&gt;</td>
<td>Specifies the context for the actions that will be allowed on WP objects displayed in the component. Default: wpsubscriptions (see contextsensitiveview_config, page 181 for details).</td>
</tr>
<tr>
<td>&lt;filter&gt;</td>
<td>Specifies the context for actions on the displayed objects based on the user role. Default: web developer has access to objects having the wpnoncontent context, other roles have read-only access to non-WP content (see Configurable Roles, page 180 for details).</td>
</tr>
<tr>
<td>&lt;wpcontext_nonwp&gt;</td>
<td>Specifies the non-WP context, which resolves to a list of disallowed actions on objects that are not WP objects (see contextsensitiveview_config, page 181 for details).</td>
</tr>
</tbody>
</table>

subscriptions_streamline

**Purpose**  
Displays the user’s subscribed files and objects. This component is a tab in the tabbar, page 247 component.

**Parameters**

This component has no defined parameters.
**Elements**

This component has the same configurable elements as `subscriptions_classic`, page 243.

---

**supcategory_classic**

**Purpose**

Extends `category_classic`, page 170 and displays the user’s supporting files in a category folder. This component is launched by a link in the `changesetaddsupportingfilescontainer`, page 171 UI.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;showtitle&gt;</code></td>
<td>True to display the title for the list of categories</td>
</tr>
</tbody>
</table>

---

**supchannellist**

**Purpose**

Extends `channel_classic`, page 174 and displays the user’s supporting files in a channel. This component is launched by a link in the `changesetaddsupportingfilescontainer`, page 171 UI.

**Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;showtitle&gt;</code></td>
<td>True to display the title for the list of categories</td>
</tr>
<tr>
<td><code>&lt;showfilters&gt;</code></td>
<td>True to show filters for documents, folders, or objects</td>
</tr>
</tbody>
</table>

---

**supmyfiles_classic**

**Purpose**

Extends `myfiles_classic`, page 210 and displays the supporting files owned by the user. This component is launched by a link in the `changesetaddsupportingfilescontainer`, page 171 UI.

**Parameters**

This component has no defined parameters.
Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext_content&gt;</td>
<td>Context for actions that can be performed on WP content. By default, it is invalid all. See contextsensitiveview_config, page 181 for details.</td>
</tr>
<tr>
<td>&lt;wpcontext&gt;</td>
<td>Specifies the context for the actions that will be allowed on non-WP objects displayed in the component.</td>
</tr>
</tbody>
</table>

supportingfiles

Purpose
Extends the webcomponent layer’s doclist component and displays the supporting files in the application.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtualpath</td>
<td>Sets the breadcrumb virtual path to child components, for example, a/b/c</td>
</tr>
<tr>
<td>components</td>
<td>Specifies a list of dot-separated components to be displayed as the virtual path in the breadcrumb, for example, a.b.c</td>
</tr>
</tbody>
</table>

Elements
This component has configurable columns (see the webcomponent layer’s doclist component). You can also configure the following elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext&gt;</td>
<td>Specifies the context for the actions that will be allowed on WP objects displayed in the component. Default: wpsubscriptions (see contextsensitiveview_config, page 181 for details).</td>
</tr>
</tbody>
</table>

supportingtemplateassetlocator and container

Purpose
Extends wptemplatelocator and container, page 284 and locates dm_document objects in a supporting templates folder. The container extends wptemplatelocator and container, page 284 container.
Parameters

The parameters are the same as those for category assetlocator and container, page 169.

**supportingtemplatesproperties**

**Purpose**
Extends the WDK layer’s propertiesheetcontainer component and displays the properties of supporting templates. This component is a container and is nested to by the properties, page 130 action if the object is a rules, layout, or publishing template.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object for which properties will be displayed</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
</tbody>
</table>

**Elements**

This container contains the attributes, permissions, subscriptions, and history components.

| <contains> | <component> elements within this element specify the components to be displayed within the properties container |

**supsubscriptions_classic**

**Purpose**
Extends subscriptions_classic, page 243 and displays the user’s subscribed supporting files and objects. This component contained within changesetaddsupportingfilescontainer, page 171.

**Parameters**

This component inherits the parameter of subscriptions_classic, page 243, if any.

**Elements**

This component has the same configurable elements as supmyfiles_classic, page 244.
**tabbar**

**Purpose**
Extends the Webtop layer’s tabbar component and adds Web Publisher component tabs.

**Parameters**
This component inherits the parameters of the Webtop layer’s tabbar component.

**Elements**
This container has configurable tabs (see the Webtop layer’s tabbar component).

**taskmgrcontainerclassic**

**Purpose**
Displays the task manager components: wptaskmanagerclassic (task info), wptasksupportingfilesclassic (task supporting files), taskcomments, and taskprogress. This component is called by selecting a task in the inboxclassic component.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. Task ID.</td>
</tr>
<tr>
<td>containertabid</td>
<td>Required. Default component to be displayed first.</td>
</tr>
</tbody>
</table>

**taskmgrcontainerstreamline**

**Purpose**
Extends taskmgrcontainerclassic, page 247 and displays the task manager components: wptaskmanagerclassic (task info), wptasksupportingfilesclassic (task supporting files), taskcomments, and taskprogress. This component is called by selecting a task in the inboxclassic, page 201 component.

**Parameters**
The parameters are the same as those for taskmgrcontainerclassic, page 247.
**taskstatus**

**Purpose**
Displays the status of a task. This component is nested within the UI (JSP pages) of the icestatusbar, page 199 and statusbar, page 242 components.

**Parameters**
This component has no defined parameters.

**taxonomies**

**Purpose**
Displays taxonomies in the application. This component is called by the action taxonomies, page 142 in the component UI of menubar, page 209.

**Parameters**
This component has no defined parameters.

**Elements**
A <filter> element filters for WP role. Non wcm_administrator role is limited to a read-only view of taxonomies by the <filter>.<readOnly> element.

**taxonomywebcabinetslist**

**Purpose**
Displays category cabinets in the Docbase. This component is called by the action webcabinets, page 147 from the UI of menubar, page 209.

**Parameters**
This component has no parameters.

**Elements**
A <filter> element filters for WP role. Non wcm_administrator role is limited to a read-only view of taxonomies by the <filter>.<readOnly> element.
templatefilter

**Purpose** Displays a filter for templates within the container filtercontainer, page 192. The filter menu is defined in the UI (/wp/classic/templates/filter.jsp) with the following options: files and folders, folders only, files only, type, status, and reset menu to default. This component is called from a link in the UI of the component templates, page 249.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Set to true to display a filter for template type</td>
</tr>
<tr>
<td>status</td>
<td>Set to true to display the change set status menu item</td>
</tr>
</tbody>
</table>

templates

**Purpose** Extends the webcomponent layer’s doclist component and displays templates in the application. This component is called as a node in the sitemanager, page 234 component.

**Parameters**

This component has the same parameters as presentationsfolder, page 220.

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component). You can also configure the following elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wpcontext&gt;</td>
<td>Specifies the context for the actions that will be allowed on WP objects displayed in the component. Default: wpcategorytemplate (see contextsensitiveview_config, page 181 for details).</td>
</tr>
<tr>
<td>&lt;wpcontext_root&gt;</td>
<td>WP context for actions that will be allowed on templates in the root folder (default = null).</td>
</tr>
</tbody>
</table>
**testbed component**

**Purpose**
Extends the webcomponent layer’s testbed component and provides a testing environment for individual actions and components. Any action or component from the WDK library, Web Publisher application library, or custom library built on WDK can be tested. Also provides a common GUI for test automation.

**Parameters**
Inherits the parameter of the webcomponent layer’s testbed component.

**Elements**
See the webcomponent layer’s testbed component.

**Usage**
See the webcomponent layer’s testbed component.

**thumbnailpreview**

**Purpose**
Displays a thumbnail preview if a DAM server is connected.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>ID of the object that has an associated thumbnail</th>
</tr>
</thead>
</table>

**toolbar**

**Purpose**
Extends the Webtop layer’s toolbar component and adds Web Publisher controls and action buttons.

**Parameters**
This component has no defined parameters.
transformcontainer

**Purpose**
Extends the DAM transformcontainer component and launches either the `propertiestransform`, page 224 component, to change properties, or the DAM `configuretransform` component, to change DAM configuration. This component is launched by the `menuItemtransform`, page 125 action in the `menubar`, page 209, `toolbar`, page 250, and `renditions`, page 228 components.

**Parameters**
This component inherits the following parameters from the DAM transformcontainer component:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Required. The contained component.</td>
</tr>
<tr>
<td>folderId</td>
<td>Docbase path to the selected object</td>
</tr>
<tr>
<td>renditionId</td>
<td>ID of the rendition. If null, the object is considered to be the primary object and its format is used as the source format.</td>
</tr>
</tbody>
</table>

translationslist

**Purpose**
Extends the Webtop layer’s `objectlist` component and displays a list of attributes for a selected translation. This component is launched by the `translationslist`, page 143 action in the `menubar`, page 209 component UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>parentId</td>
<td>ID of the document to which the translation is attached.</td>
</tr>
<tr>
<td>type</td>
<td>Required. Type of the selected object.</td>
</tr>
</tbody>
</table>

**Elements**
This component has configurable columns (see the webcomponent layer’s `doclist` component). You can also configure the following element:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;objecttype&gt;</td>
<td>Specifies the object type to be displayed by default</td>
</tr>
</tbody>
</table>
updatecontent

Purpose
Updates all XML documents that are based on a selected template by using an instructions file. This component is launched by the action `updatecontent`, page 143 in the container `wpmultiargdialogcontainer`, page 278. The process is asynchronous, with a notification in the user's inbox upon update completion. A log file for updates is generated in the Docbase folder `WebPublisher Configuration/WcmLog`. Files that are checked out or in a change set with the state “staging” will not be updated.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of selected content files to be updated.</td>
</tr>
<tr>
<td>templateId</td>
<td>Required. Template ID.</td>
</tr>
</tbody>
</table>

Elements
This component has configurable columns (see the webcomponent layer's `doclist component`).

Usage
You should not update content unless the template and rules have first been updated. If you update content without changing the rules and template that the content is based on, you will not be able to use the template editor to edit the changed content in the future. Your content will be associated with a template and rules file that do not include your changes.

To avoid this version mismatch, edit your rules and template files to reflect the planned updates to content.

uploadcontent and container

Purpose
Uploads content to the Docbase. This component and container are launched by the `uploadcontent`, page 143 action, which in turn is called from the `changesetaddsupportingfilescontainer`, page 171 container and the `newcontent and container`, page 215 component. The container extends the webcomponent layer's `contentxfercontainer` component.
Parameters

The parameters are the same as those for the uploadcontent, page 143 action.

usersupportingfiles

Purpose
Extends publicsupportingfiles, page 225 and displays the user’s supporting files. This component is contained in the changesetaddsupportingfilescontainer, page 171 container.

Parameters
This component inherits the parameters of publicsupportingfiles, page 225, if any.

versions

Purpose
Extends the Webtop layer’s versions component and displays the versions for a selected object. This component is scoped by default to the dm_sysobject type. The component is called by the versions, page 144 from the menubar, page 209 component.

Parameters

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Required. Type of the selected object.</td>
</tr>
</tbody>
</table>

Elements
This component has configurable columns (see the webcomponent layer’s doclist component).

viewcontainer

Purpose
Extends the webcomponent layer’s viewcontainer component and allows the user to view a selected object.

Parameters
The parameters are inherited from the webcomponent layer’s viewcontainer component.
Elements
This component has configurable columns (see the webcomponent layer’s doclist component).

**view_all_of_type**

**Purpose**
Extends the Webtop layer’s objectlist component and allows the user to view all manual links of a supported type in the application. This component is jumped to from the component list_manual_link_component, page 205.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. Specifies the ID of the link type.</th>
</tr>
</thead>
</table>

Elements
This component has configurable columns (see the webcomponent layer’s doclist component).

**viewclipboard**

**Purpose**
Allows the user to view the clipboard.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the target folder into which the items will be moved.</th>
</tr>
</thead>
</table>

Elements
This component has configurable columns (see the webcomponent layer’s doclist component).

**viewmanager**

**Purpose**
In a portal environment, calls the view component to display an HTML view in a separate window and closes it after the user exits.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the viewed object.</td>
</tr>
<tr>
<td>chronicleId</td>
<td>Object ID of the original version of the object.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the viewed object.</td>
</tr>
</tbody>
</table>

viewmenu

Purpose
Displays a UI that allows the user to configure menu actions that are available in each component view. For more information about menu configuration, see contextsensitiveview_config, page 181. This component is launched by user selection in the hiddentool, page 194 UI.

Parameters
This component has no defined parameters.

viewselectedfileslist

Purpose
Displays a list of files that have been selected for addition to a change set.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csFolderId</td>
<td>ID of the change set folder</td>
</tr>
<tr>
<td>selectedListName</td>
<td>Not used.</td>
</tr>
<tr>
<td>WPView</td>
<td>Boolean flag that indicates that the list is a Web Publisher view</td>
</tr>
</tbody>
</table>

Elements
This component has configurable columns (see the webcomponent layer’s doclist component) with the following additional element:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;objecttype&gt;</td>
<td>Specifies the object type to be displayed by default</td>
</tr>
</tbody>
</table>
web_author_preferences

**Purpose**
Allows the Web author to set preferences including selecting an HTML editor and default locale filter.

**Parameters**
There are no parameters defined for this component.

webcabinetslocator

**Purpose**
Extends the webcomponent layer’s alldocumentlocator component and allows the user to locate Web cabinet. This component is called within the container add_manual_link, page 159 the add_manual_link, page 107 action.

**Parameters**
The parameters are inherited from the webcomponent layer’s alldocumentlocator component.

**Elements**
This component inherits configurable views (see the webcomponent layer’s objectlocator component).

web_developer_preferences

**Purpose**
Allows the Web developer to set preferences including selecting a rules editor, temporary directory, link checker application, differencing application, and application for viewing source.

**Parameters**
There are no parameters defined for this component.
**webhtml_add_hyperlink**

**Purpose**
Extends the webcomponent layer's documentlocatorcontainer and allows the user to select a document to be added as a hyperlink in eWebEditPro. When the user clicks the button to add hyperlinks from eWebEditPro's editor, a JavaScript event handler calls the webhtml_add_hyperlink component.

**Parameters**
The parameters, if any, are inherited from the webcomponent layer's documentlocatorcontainer.

**Customization**
The container class retrieves the following parameters: path for selected object, selected target, and the hyperlink text. Then the parameters are validated and passed to add_hyperlink_action.jsp to add the hyperlink.

**webhtml_add_image_action and container**

**Purpose**
Called by the container webhtml_add_image, page 258 to add an image to a document during editing by eWebEditPro. The container extends the webcomponent layer's contentxfercontainer component.

**Parameters**
The following values are supplied to this component from one of the webhtml image locator components.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cabinetName</td>
<td>Required. Name of the cabinet that contains the image.</td>
</tr>
<tr>
<td>imageLinkValue</td>
<td>Required. Path to image.</td>
</tr>
</tbody>
</table>

The container inherits the parameters of the WDK layer's combocontainer component and adds the following parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the selected image</td>
</tr>
<tr>
<td>filenameWithPath</td>
<td>Required. Selected folder path and file name.</td>
</tr>
</tbody>
</table>
**webhtml_add_image**

**Purpose**
Extends the webcomponent layer's documentlocatorcontainer and allows the user to select an image to be added when using eWebEditPro. When the user clicks the button to add images from eWebEditPro's editor, a JavaScript event handler calls the webhtml_add_image component.

**Parameters**
The parameters, if any, are inherited from the webcomponent layer's documentlocatorcontainer.

**Customization**
The container class retrieves the path for selected object, performs validation, and calls add_image_action.jsp to add the image.

**webhtml_subscription_image_locator**

**Purpose**
Extends the webcomponent layer's documentsubscriptionlocator and allows the user to select a subscribed image to be added as an image in eWebEditPro. This component is contained within the container webhtml_add_image, page 258.

**Parameters**
The parameters, if any, are inherited from the webcomponent layer's documentsubscriptionlocator.

**Elements**
This component inherits configurable views (see the webcomponent layer's objectlocator component). Note that the base document type is configured to be dm_document.

**webhtml_webcabinets_image_locator**

**Purpose**
Extends wpsysobjectlocator and container, page 281 and allows the user to select an image (sysobject) in a Web cabinet to be added to a page edited by eWebEditPro. This component is contained within the container webhtml_add_image, page 258.
**Parameters**

The parameters, if any, are inherited from `wpsysobjectlocator` and `container`, page 281.

**Elements**

This locator has configurable columns (see the webcomponent layer’s `doclist` component) and configurable views (see the webcomponent layer’s `objectlocator` component).

---

**webhtml_wpmyfiles_image_locator**

**Purpose**

Extends the webcomponent layer’s `mydocumentlocator` and allows the user to select one of the user’s own images to be added as a hyperlink in eWebEditPro. This component is contained within the container `webhtml_add_image`, page 258.

**Parameters**

The parameters, if any, are inherited from the webcomponent layer’s `mydocumentlocator` component.

**Elements**

This component inherits configurable views (see the webcomponent layer’s `objectlocator` component).

---

**webhtml_subscription_hyperlink_locator**

**Purpose**

Extends the webcomponent layer’s `documentsubscriptionlocator` and allows the user to select a subscribed document to be added as a hyperlink in eWebEditPro. This component is contained within the container `webhtml_add_hyperlink`, page 257.

**Parameters**

The parameters, if any, are inherited from the webcomponent layer’s `documentsubscriptionlocator`.

**Elements**

This component inherits configurable views (see the webcomponent layer’s `objectlocator` component).
webhtml_webcabinets_hyperlink_locator

Purpose
Extends the webcomponent layer’s alldocumentlocator component and allows the user to select a document in a Web cabinet to be added as a hyperlink in eWebEditPro. This component is contained within the container webhtml_add_hyperlink, page 257.

Parameters
The parameters, if any, are inherited from the webcomponent layer’s alldocumentlocator component.

Elements
This locator has configurable queryfilters (see the webcomponent layer’s objectlocator component). This component inherits configurable views (see the webcomponent layer’s objectlocator component). Note that the base document type is configured to be dm_document.

webhtml_wpmyfiles_hyperlink_locator

Purpose
Extends the webcomponent layer’s mydocumentlocator and allows the user to select one of the user’s own documents to be added as a hyperlink in eWebEditPro. This component is contained within the container webhtml_add_hyperlink, page 257.

Parameters
The parameters, if any, are inherited from the webcomponent layer’s mydocumentlocator.

Elements
This component inherits configurable views (see the webcomponent layer’s objectlocator component). Note that the base document type is configured to be dm_document.

webview_component

Purpose
Allows the user to preview a file in Web view. This component is launched by the webview, page 147 action.
whereused

**Parameters**

This component has the same parameters as [webview](#) page 147 with the following additional parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insamewindow</td>
<td>In portal environment, set to true to display the component in the same window as the calling portlet</td>
</tr>
</tbody>
</table>

**Purpose**

Displays where the selected object of type dm_document, such as a presentation file, template, or rule, is used. This component is called by the [whereused](#) page 148 action in the [menubar](#) page 209 and [toolbar](#) page 250 components.

**Parameters**

The parameters are the same as [whereused](#) page 148.

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component).

**workflowassetlocator and container**

**Purpose**

Extends [wpwftemplateloader and container](#) page 287 and locates workflow assets.

**Parameters**

This component inherits the contract parameters for the webcomponent layer’s sysobjectlocator component.

**Elements**

This component inherits the configurable elements of the webcomponent layer’s sysobjectlocator component.
**workflowreport**

**Purpose**
Displays a list of workflows in the delivery cabinet. This component is not called in the Web Publisher application.

**Parameters**
This component has the same parameters as the alerts, page 164 component.

**workflowreportcriteria**

**Purpose**
Displays a UI for setting criteria to be used for a report of workflows in the delivery cabinet. This component is called from an option in the reportselection, page 229 UI.

**Parameters**
This component has the same parameters as the alerts, page 164 component.

**workflowstatusclassic**

**Purpose**
Extends the webcomponent layer’s workflowstatuslist component and lists the status of workflows in a classic view. This component is called by the workflowstatusclassic, page 149 in the menubar, page 209 component UI.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processId</td>
<td>If supplied, only workflows with the specified process ID are displayed</td>
</tr>
<tr>
<td>currentPerformer</td>
<td>If supplied, only workflows for the current performer are displayed</td>
</tr>
<tr>
<td>creatorName</td>
<td>If supplied, only workflows created by the current user are displayed</td>
</tr>
<tr>
<td>supervisorName</td>
<td>If supplied, only workflows for which the current user is supervisor are displayed</td>
</tr>
<tr>
<td>workflowFilter</td>
<td>Displays send to distribution workflows, regular workflows, or both. Valid values: SHOW_SENDTO_DISTRIBUTION, SHOW_REGULAR_WORKFLOWS, SHOW_SENDTO_AND_REGULAR_WORKFLOWS</td>
</tr>
<tr>
<td>showAll</td>
<td>True to display all workflows, false to display only workflows supervised by the current user</td>
</tr>
</tbody>
</table>

Elements

This component inherits the configurable elements of the webcomponent layer’s sysobjectlocator component.

workflowstatusstreamline

Purpose

Extends the webcomponent layer’s workflowstatusdrilldown component and lists the status of workflows in a streamline view. This component is called from a link in the inboxstreamline, page 201 component.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processId</td>
<td>If supplied, only workflows with the specified process ID are displayed</td>
</tr>
<tr>
<td>currentPerformer</td>
<td>If supplied, only workflows for the current performer are displayed</td>
</tr>
<tr>
<td>creatorName</td>
<td>If supplied, only workflows created by the current user are displayed</td>
</tr>
<tr>
<td>supervisorName</td>
<td>If supplied, only workflows for which the current user is supervisor displayed</td>
</tr>
<tr>
<td>workflowFilter</td>
<td>Displays send to distribution workflows, regular workflows, or both. Valid values: SHOW_SENDTO_DISTRIBUTION, SHOW_REGULAR_WORKFLOWS, SHOW_SENDTO_AND_REGULAR_WORKFLOWS</td>
</tr>
<tr>
<td>showAll</td>
<td>True to display all workflows, false to display only workflows supervised by the current user</td>
</tr>
</tbody>
</table>

Elements

This component inherits the configurable elements of the webcomponent layer’s sysobjectlocator component.

wp_applauncher

Purpose

Launches an external application, such as a third-party editor, a source viewing application, or a differencing tool. This component is nested within the differencing component container, the view source container, and the publishingconfig component.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Command that initializes and launches the external source view application</td>
</tr>
</tbody>
</table>

Usage

The wp_applauncher component is used by Web Publisher to launch a third-party applet for editing content files, viewing source, or differencing files. To use a third-party editor, specify the editor name in the component definition for the wp_edit and container,
The differencing tool and view source application are selected by the user from the UI.

**wp_cancelcheckout and container**

**Purpose**
Cancels checkout on a document for a user with version_permit access to the document. This component is called by the action `wpcancelcheckout`, page 150 in the UI of `menubar`, page 209. The container extends the webcomponent layer's `contentxfercontainer` component.

**Parameters**
- **objectId**
  - Required. ID of the object that is checked out.
- **lockOwner**
  - Name of the user who has checked out the document

**wp_ceedit**

**Purpose**
Content editor component that launches a browser plugin to edit objects of type `dm_document` or a subclass.

**Parameters**
The parameters are the same as those for `wp_edit and container`, page 267.

**wp_checkin and container**

**Purpose**
Extends the webcomponent layer's `contentxfercontainer` component and checks in one or more objects to the Docbase.

**Parameters**
- **objectId**
  - Required. The ID of the object to be checked in
Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;filter&gt;</td>
<td>Specifies a filter for the the minimum Docbase role that has checkin permission</td>
</tr>
<tr>
<td>&lt;prompt_comment&gt;</td>
<td>True to prompt the user for a comment on checkin</td>
</tr>
<tr>
<td>&lt;require_comment&gt;</td>
<td>True to require the user to enter a comment on checkin</td>
</tr>
<tr>
<td>&lt;show_version&gt;</td>
<td>True to display UI options that set the version on checkin</td>
</tr>
</tbody>
</table>

Customization

For customizing the list of attributes for checkin, see the webcomponent layer’s checking component.

wp_checkout and container

Purpose

Checks one or more objects out of the Docbase. The user must have write permission on the object to check it out. The container wp_checkoutcontainer extends the webcomponent layer’s contentxfercontainer component.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. The ID of the object to be checked out.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Name of the user who has locked the object</td>
</tr>
</tbody>
</table>

wp_diffcontent and container

Purpose

Launches a differencing application, selected by the user in the preferences component, to compare a checked out file with the Docbase copy to determine whether the local checked out copy has been changed. The component checks whether the file is checked out to the user and then exports another local copy for comparison. This component and the wp_diffcontent_container are launched by the actions diffversions, page 117 and difflocal, page 117 in the UI of the components menubar, page 209 and toolbar, page 250. The wp_diffcontent_container extends the webcomponent layer’s contentxfercontainer component.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the checked out object.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object</td>
</tr>
</tbody>
</table>

wp_edit and container

Purpose

Checks out an object of type dm_document or a subtype and opens an editing application on the user's local machine. The container wp_editcontainer extends the webcomponent layer's contentxfercontainer component.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be edited.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>Specifies the user who has locked the object</td>
</tr>
</tbody>
</table>

The editcontainer has the usual container arguments (component and componentArgs) plus the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folderPath</td>
<td>Path to the object in the Docbase</td>
</tr>
</tbody>
</table>

Elements

The component has no configurable elements. The container has the following configurable elements that specify editors for content:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;editingapps&gt;</td>
<td>Contains &lt;editingapp&gt; elements that specify editing applications</td>
</tr>
<tr>
<td>&lt;editingapp&gt;</td>
<td>Specifies the editing application to be used for a given content type. Valid values for editing_app_ind attribute include: CONTENT_EDITOR, RULES_EDITOR, WEB_HTML_EDITOR. The value of the action_id attribute specifies the action that launches the editing application.</td>
</tr>
</tbody>
</table>

wp_export and container

Purpose

Exports an object. The wpexportcontainer extends the webcomponent layer's contentxfercontainer component and is launched by the export, page 119 action.
wp_export_rendition and container

**Purpose**
Exports a rendition. The wp_export_renditioncontainer extends the webcomponent layer’s contentxfercontainer component and is launched by the exportrendition, page 119 action.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the object to be exported.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
</tbody>
</table>

wp_ice

**Purpose**
Displays the in-context editing node in the browsertree, page 168 component. The user can select from a list of sites to launch the editor. The UI page, selectsite.jsp, calls the JSP page iceframes.jsp, which loads the iceconsole, page 198 component.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>ID of the selected site</td>
</tr>
<tr>
<td>category_id</td>
<td>ID of the category for the selected Web page. This parameter is passed to the ice_custom_config, page 195 component.</td>
</tr>
<tr>
<td>template_id</td>
<td>ID of the template to be used in editing. This parameter is passed to the ice_custom_config, page 195 component.</td>
</tr>
<tr>
<td>action</td>
<td>Action to be launched when the component is initialized</td>
</tr>
</tbody>
</table>
wp_import_content and importcontainer

**Purpose**
Imports content into the Docbase. The wp_import_content component is called by the action import, page 121 on wcm_channel type folders. The wp_importcontainer component extends the webcomponent layer’s contentxfercontainer component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the folder that will contain the imported content.</td>
</tr>
<tr>
<td>filenameWithPath</td>
<td>Required. Full path including file name and extension on the local system.</td>
</tr>
</tbody>
</table>

The wp_importcontainer component adds a folderPath parameter to the standard container parameters. The folderPath parameter specifies the path that should be displayed in the breadcrumb. This path must be a complete path from the Docbase root, for example, /Doc%20Library/subfolder

**Elements**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Sets the base type for the type dropdown list. For example, dm_sysobject would list all subtypes of dm_sysobject. Default = dm_document.</td>
</tr>
<tr>
<td>enable_full_text_index</td>
<td>Boolean flag. Set to true to enable full-text indexing for the object.</td>
</tr>
<tr>
<td>enable_a_category_attr</td>
<td>Boolean flag. Set to true to display the category panel that allows the user to select a category for the imported object.</td>
</tr>
</tbody>
</table>

**Customization**

For customizing the list of attributes for import, see the webcomponent layer’s import component.

wp_import_content_template

**Purpose**
Imports a template into a category. The component is called by the action import, page 122 on wcm_category type folders.
wp_import_instructions and container

Parameters

The parameters are the same as those for \texttt{wp\_import\_content} and \texttt{importcontainer}, page 269.

Elements

The configurable elements are the same as those for \texttt{wp\_import\_content} and \texttt{importcontainer}, page 269.

\textbf{wp\_import\_instructions and container}

Purpose

Imports an XML instructions file for updating all content based on a selected template. The file is validated against the updatexmlcontent application in the Docbase. The component is called by the action \texttt{import}, page 121 for the \texttt{wpcontext} "wpinstructions."

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{objectId}</td>
<td>Required. ID of the folder that will contain the imported content.</td>
</tr>
<tr>
<td>\texttt{filenameWithPath}</td>
<td>Required. Full path including file name and extension on the local system.</td>
</tr>
</tbody>
</table>

The \texttt{wp\_instructions\_importcontainer} component extends the webcomponent layer’s \texttt{contentxfercontainer} component.

Elements

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{&lt;default_formats&gt;}</td>
<td>Contains \texttt{&lt;format&gt;} elements that should be matched to DOS file extensions. For example, there can be 7 \texttt{dm_format} objects that have the extension .doc.</td>
</tr>
<tr>
<td>\texttt{&lt;format&gt;}</td>
<td>Each format describes a default format that is used when a user imports a file with a filename extension that matches the value of the dos_extension attribute. If a configured format cannot be found, the import component will try to find a format with the file’s dos extension. If no format can be found, the user can select a format. Attributes: dos_extension (for example, ’doc’) or name (for example, ’msw8’).</td>
</tr>
</tbody>
</table>
**wp_import_template**

**Purpose**
Imports a template into the application. This component is scoped to objects of type dm_folder and its subtypes. The component is called by the action import, page 121.

**Parameters**
The parameters are the same as those for wp_import_content and importcontainer, page 269.

**Elements**

| enable_full_text_index | Boolean flag. Set to true to enable full-text indexing for the object. |

**wp_reedit**

**Purpose**
Rules editor component that launches a browser plugin to check out and edit a rules file. This component is launched by addnewrule and addnewruledialogcontainer, page 160. If the application is configured to use a Web-based rules editor, this component will not be launched.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the rules file to be edited.</th>
</tr>
</thead>
<tbody>
<tr>
<td>useRulesForAllTranslations</td>
<td>Set to true to use a rules file for all translations of a document.</td>
</tr>
</tbody>
</table>

**wp_unziprendition**

**Purpose**
Unzips a selected rendition on the user’s local machine. This component is nested within the wp_export_rendition and container, page 268 container.
wp_viewsource and container

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unziparchive</td>
<td>True to unzip the archived rendition. Default = null</td>
</tr>
<tr>
<td>objectName</td>
<td>Name of rendition</td>
</tr>
</tbody>
</table>

Elements

<debug_applet> | Set to true to turn on applet debugging. In debugging mode, temporary content transfer files on the WDK server and client machine are not deleted.

**wp_viewsource and container**

**Purpose**
Launches a viewing application to view the source of a Web page. This component is called by the viewsource, page 146 on objects of type dm_document. The source file is exported to the local file system for viewing. The container wp_viewsource_container extends the webcomponent layer’s contentxfercontainer component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the Web page.</td>
</tr>
<tr>
<td>contentType</td>
<td>File format to use when opening the object (passed to the operating system)</td>
</tr>
</tbody>
</table>

**Usage**
You must extend this component to support viewing the source for objects that are not dm_document type.

**wp_wheedit**

**Purpose**
Launches a Web HTML editor to edit a selected dm_document object. This component is launched by the wp_editwithwhe, page 150 action.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the document.</td>
</tr>
<tr>
<td>lockOwner</td>
<td>User who has locked the document.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Required. Path to the document to be edited.</td>
</tr>
</tbody>
</table>

Usage

You must extend this component to support viewing the source for objects that are not dm_document type.

wpallfolderlocator and container

Purpose

Extends the webcomponent layer’s sysobjectlocator component and allows the user to locate a folder. This component is contained within the wpallfolderlocatorcontainer. This component and its container are nested within the advsearch, page 163 component.

Parameters

The parameters are inherited from the webcomponent layer’s sysobjectlocator component.

Elements

This component inherits configurable views (see the webcomponent layer’s objectlocator component).

wpcategorylocator and container

Purpose

Extends the component wpsysobjectlocator and container, page 281 and locates all Web Publisher categories. This component and wpcategorylocatorcontainer are nested within categories, page 169. The container extends wpsysobjectlocatorcontainer.

Parameters

The parameters are the same as those for the component categoryassetlocator and container, page 169.
Elements
The configurable <views> element is the same as <view> in the webcomponent layer’s objectlocator component. The configurable <columns> element and its child elements are inherited from the webcomponent layer’s objectgrid component. The following additional element is defined:

```
<containerselectable> Boolean flag that specifies whether the container type is selectable when the container is a subtype of the objecttype.
```

wpchannelfolderlocator and container

Purpose
Extends the webcomponent layer’s allfolderlocator component and allows the user to locate a Web Publisher channel (delivery cabinets) by browsing from the root cabinets. This component is nested within the propertiestransform, page 224 component. The wpchannelfolderlocatorcontainer extends the webcomponent layer’s locatorcontainer component and contains two components in addition to wpchannelfolderlocator: foldersubscriptionlocator and recentfolderlocator.

Parameters
This component inherits parameters from the webcomponent layer’s sysobjectlocator component.

Elements
This component has the same configurable elements as the webcomponent layer’s sysobjectlocator component.

wpclipboard and container

Purpose
Displays the contents of the clipboard. The container wpclipboardcontainer extends the WDK layer’s combocontainer component. The container is launched by the actions copyonwcm, page 114 and wpviewclipboardaction, page 153.

Parameters
The component has no defined parameters. The container inherits parameters from the WDK layer’s combocontainer.

Elements
The component has the same configurable elements as wpcopy, page 275.
wpcontenttemplatelocator and container

**Purpose**
Extends wpsysobjectlocator and container, page 281 and allows the user to locate a content template. The container extends the container wpsysobjectlocator and container, page 281. The wpcontenttemplatelocatorcontainer (and contained component) is called by the templatebrowse, page 142 action.

**Parameters**
The parameters are inherited from wpsysobjectlocator and container, page 281.

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component). The component also has configurable views, which are described in the webcomponent layer’s objectlocator component.

wpcopy

**Purpose**
Allows the user to select objects from the clipboard to copy. The clipboard of WCM objects is displayed when the user selects a filter in the UI. If one or more selections cannot be copied, the Copy button is disabled.

**Parameters**

| objectId | Required. ID of the target folder into which the items will be copied. |

**Elements**
This component has configurable columns (see the webcomponent layer’s doclist component).

wpfolderlocatorcontainer

**Purpose**
Extends the webcomponent layer’s locatorcontainer component and allows the user to locate a a folder. This container are nested within the advsearch, page 163 component.
Parameters

The parameters are inherited from the webcomponent layer’s sysobjectlocator component.

Elements

This component inherits configurable views (see the webcomponent layer’s objectlocator component).

wpinstructionlocatorand container

Purpose

Extends wpsysobjectlocator and container, page 281 and allows the user to locate an instruction XML file. The wpinstructionlocator is nested to from the updatecontent, page 143 action. The container extends the container for wpsysobjectlocator and container, page 281.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootpath</td>
<td>Root folder path in which to begin searching for instructions files</td>
</tr>
<tr>
<td>objecttype</td>
<td>Type of the selected object</td>
</tr>
<tr>
<td>flatlist</td>
<td>Boolean: true to list all objects in a flat list, false to display a drop down list.</td>
</tr>
<tr>
<td>multiselect</td>
<td>Boolean: true to support multiple object selection</td>
</tr>
<tr>
<td>selectedobjectids</td>
<td>Comma-delimited or semicolon-delimited list of object IDs that are initially selected</td>
</tr>
</tbody>
</table>

wplifecyclelocator and container

Purpose

Extends the webcomponent layer's allifecyclelocator component and allows the user to locate a lifecycle. The wplifecyclelocatorcontainer (and contained component) is called by the lifecyclebrowse, page 123 action in the attributes (dm_document, dm_category), page 166 component UI.

Parameters

The parameters are inherited from the webcomponent layer’s sysobjectlocator component.
**Elements**

This component has configurable columns (see the webcomponent layer's doclist component) and views (see the webcomponent layer's objectlocator component).

**Usage**

The wpcopy, wplink, and wpmove components have a default set of standard sysobject attributes that are displayed and formatted in columns. You can also configure columns to be displayed dynamically based on other attributes. See “Configuring Dynamic Data Columns” in *Web Development Kit and Client Applications Development Guide* for more information.

**wplink**

**Purpose**

Allows the user to select items from the clipboard to link to a target folder. If one or more selections cannot be linked, the Link button is disabled. This component is called from wplinkaction.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the target folder into which the items will be linked.</th>
</tr>
</thead>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer's doclist component).

**Usage**

The wpcopy, wplink, and wpmove components have a default set of standard sysobject attributes that are displayed and formatted in columns. You can also configure columns to be displayed dynamically based on other attributes. See “Configuring Dynamic Data Columns” in *Web Development Kit and Client Applications Development Guide* for more information.

**wplocalelist**

**Purpose**

Displays a list of locales with language name, country name, and encoding name. This component is called by the action viewwplocalelist, page 146 from the menubar, page 209 and toolbar, page 250 components.
Parameters

This component has no defined parameters.

Elements

This component has configurable columns (see the webcomponent layer's doclist component). The component is configured by default to display the language_name, country_name, and encoding_name attributes. One additional element is defined:

| objecttype          | Object type to display by default (default = wcm_locale) |

wpmove

Purpose

Allows the user to select items from the clipboard to move to a target folder. If one or more selections cannot be moved, the Move button is disabled.

Parameters

| objectId  | Required. ID of the target folder into which the items will be moved. |

Elements

This component has configurable columns (see the webcomponent layer's doclist component).

Usage

By default, the move operation moves all versions of the selected object. To change this behavior, set the flag clipboardservice.moveallversions to false in the file DOCUMENTUM_HOME/config/WcmApplicationConfig.properties.

wpmultiargdialogcontainer

Purpose

Extends the webcomponent layer's multiargdialogcontainer component and supports multiple sets of arguments for the component within the container, for example, for multiple selections in the classic view.
Parameters

The parameters are the same as those for the WDK layer's combocontainer component.

Usage

If your component supports multiple selection and does not have its own container, you can use the multiargdialogcontainer to pass multiple sets of arguments to a single instance of the component. If you use the dialogcontainer, only one set of arguments is passed to the component. If you use the combocontainer, the container will instantiate the contained component for each set of arguments.

wpmyfileslocator

Purpose

Extends the webcomponent layer's mydocumentlocator and locates all documents owned by the user. This component is called within the container add_manual_link, page 159 by the add_manual_link, page 107 action.

Parameters

This component inherits the parameters of the webcomponent layer's mydocumentlocator.

Elements

This component inherits the configurable elements of the webcomponent layer's mydocumentlocator.

wpnavigationcontainer

Purpose

Displays a breadcrumb, title, and drop-down list above the included component. This container is called by the translationslist, page 143, versions, page 144, and renditions, page 134 actions.
wppresentationtemplatelocator and container

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Required. Included component.</td>
</tr>
<tr>
<td>objectId</td>
<td>Required. ID of the selected object.</td>
</tr>
<tr>
<td>folderId</td>
<td>ID of the folder that contains the selected object.</td>
</tr>
<tr>
<td>folderPath</td>
<td>Path to the folder that contains the selected object. This path must be a complete path from the Docbase root, for example, /Documentation%20Library/subfolder</td>
</tr>
</tbody>
</table>

**Purpose**

Extends the component wpsysobjectlocator and container, page 281 and locates all internal and external presentation templates. The wppresentationtemplatelocatorcontainer extends wpsysobjectlocatorcontainer.

**Parameters**

The parameters are inherited from the component wpsysobjectlocator and container, page 281 and are described in categoryassetlocator and container, page 169. An additional required parameter is the objectId parameter.

**Elements**

The configurable elements are the same as those for the webcomponent layer’s objectlocator component.

wpsubscriptionlocator

**Purpose**

Extends the component the webcomponent layer’s documentsubscriptionlocator and locates all objects subscribed to by the user. This component is called within the container add_manual_link, page 159 by the add_manual_link, page 107 action.

**Parameters**

This component inherits the parameters of the webcomponent layer’s mydocumentlocator.
Elements
This component inherits the configurable elements of the webcomponent layer’s mydocumentlocator.

wpsysobjectlocator and container

Purpose
Extends the webcomponent layer’s alldocumentlocator component and locates all Web Publisher objects. This component and container are nested within the components addpresentation and container, page 161, addnewruledialogcontainer, page 160, and thumbnailpreview, page 250. The container extends the webcomponent layer’s locatorcontainer component.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objecttype</td>
<td>Type of the selected object</td>
</tr>
<tr>
<td>flatlist</td>
<td>Boolean: true to list all objects in a flat list, false to display a drop down list.</td>
</tr>
<tr>
<td>multiselect</td>
<td>Boolean: true to support multiple object selection</td>
</tr>
<tr>
<td>selectedobjectids</td>
<td>Comma-delimited or semicolon-delimited list of object IDs that are initially selected</td>
</tr>
</tbody>
</table>

Elements
The configurable elements are the same as those for the webcomponent layer’s objectlocator component. The configurable <columns> element and its child elements are inherited from the webcomponent layer’s objectgrid component.

wpsysteminfo

Purpose
Displays the following information about the system: eContent Server version and Docbase name for the current Docbase, versions of DFC, WDK, Webtop, and Web Publisher on the Web Publisher host, user name, and date. This component is included in the adminconsole, page 162 UI.

Parameters
This component has no defined parameters.
wptaskattachment

**Purpose** Displays attachments for a task. This component is included in the wptaskmanagerclassic, page 283 and wptaskmanagerstreamline, page 283 components.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. Change set ID.</th>
</tr>
</thead>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component). Additionally, you can configure additional attributes of the attachments to be added to the query.

<table>
<thead>
<tr>
<th>&lt;attributes&gt;</th>
<th>Contains &lt;attribute&gt; elements used to build the query for attachment attributes to be displayed in the list</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;attribute&gt;</td>
<td>The value corresponds to a Docbase attribute of dm_sysobject</td>
</tr>
</tbody>
</table>

wptaskcomponentcontainer

**Purpose** Extends the webcomponent layer’s taskcomponentcontainer component and displays task components for Web Publisher workflows, including finishwpworkflowtask, forwardwpworkflowtask, rejectwpworkflowtask, delegateworkflowtask, repeatwpworkflowtask, rerunfailedautoworkflowtask, completefailedautoworkflowtask, abortfailedautoworkflowtask. This container is launched by the actions with the same name as the aforementioned components.

**Parameters**

This component has no defined parameters.
wptaskmanagerclassic

**Purpose**
Extends the webcomponent layer's taskmanager container component and displays the task info. It is the main component in the group of task manager components, which are all held in the taskmgrcontainerclassic, page 247 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the task</td>
</tr>
</tbody>
</table>

wptaskmanagerstreamline

**Purpose**
Extends the webcomponent layer's taskmanager container component and displays the task info. It is the main component in the group of task manager components, which are all held in the taskmgrcontainerstreamline, page 247 component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the task</td>
</tr>
</tbody>
</table>

wptaskmgrabortworkflow

**Purpose**
Aborts a failed automatic workflow. This component is called by the abortfailedautowpworkfowtask, page 106 action in the taskmgrcontainerclassic, page 247 and taskmgrcontainerstreamline, page 247 UI.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the task</td>
</tr>
</tbody>
</table>

wptasksupportingfilesclassic

**Purpose**
Displays a list of supporting files for a workflow task. This component is contained within the taskmgrcontainerclassic, page 247 container.
### wptasksupportingfilesstreamline

**Purpose**  
Extends wptasksupportingfilesclassic, page 283 and displays a list of supporting files for a workflow task. This component is contained within the taskmgrcontainerstreamline, page 247 container.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Required. ID of the task</td>
</tr>
</tbody>
</table>

**Elements**

This component has configurable columns (see the webcomponent layer’s doclist component). Additionally, you can configure additional attributes of the attachments to be added to the query.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;attributes&gt;</td>
<td>Contains &lt;attribute&gt; elements used to build the query for attachment attributes to be displayed in the list</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>The value corresponds to a Docbase attribute of dm_sysobject</td>
</tr>
</tbody>
</table>

### wptemplatelocator and container

**Purpose**  
Extends the webcomponent layer’s alldocumentlocator component and allows the user to locate a template. The container (and contained component) is nested in the addpresentation and container, page 161, addrule and container, page 161, and thumbnailpreview, page 250 components.
Parameters

The parameters are the same as those for categoryassetlocator and container, page 169.

Elements

This component has configurable columns (see the webcomponent layer’s doclist component). The component also has configurable views (see the webcomponent layer’s objectlocator component).

wpuseronlylocator and container

Purpose

Extends the webcomponent layer’s useronlylocator component and allows the user to locate a user for a workflow. The container (and contained component) is nested in the finishwpworkfloptask, page 192, the WP layer’s forwardwpworkfloptask, rejectwpworkfloptask, page 227, and repeatwpworkfloptask, page 228 components. The container contains the wpuseronlylocator and the webcomponent layer’s recentuseronlylocator component.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objecttype</td>
<td>Type of the selected object</td>
</tr>
<tr>
<td>flatlist</td>
<td>Boolean: true to list all objects in a flat list, false to display a drop</td>
</tr>
<tr>
<td></td>
<td>down list.</td>
</tr>
<tr>
<td>multiselect</td>
<td>Boolean: true to support multiple object selection</td>
</tr>
<tr>
<td>filterPerformers</td>
<td>The values for this parameter are supplied by the component that nests</td>
</tr>
<tr>
<td></td>
<td>this locator.</td>
</tr>
<tr>
<td>filterSubquery</td>
<td>Query to filter users or groups. The subquery narrows down the results that</td>
</tr>
<tr>
<td></td>
<td>are displayed by the locator.</td>
</tr>
</tbody>
</table>

wpuserorgrouplocator and container

Purpose

Extends the webcomponent layer’s userorgrouplocator component and allows the user to locate a user or group for a workflow. The container (and contained component) is nested in the startwpwfperformers, page 236 component. The container contains the wpuseronlylocator and the webcomponent layer’s recentuserorgrouplocator component.

Parameters

This component has the same parameters as wpuseronlylocator and container, page 285.
wpwfgrouponlylocator and container

**Purpose**
Extends the webcomponent layer's wfgrouponlylocator container component and allows the user to locate a group for a workflow. The container (and contained component) is nested in the `startwpwfperformers`, page 236, `finishwpworkflowtask`, page 192, the `WP` layer's `forwardwpworkflowtask`, `rejectwpworkflowtask`, page 227, and `repeatwpworkflowtask`, page 228 components. The container extends the webcomponent layer's wfgrouponlylocator container component.

**Parameters**
This component has the same parameters as `wpuseronlylocator and container`, page 285.

wpwfuserorgroupfromgrouplocator and container

**Purpose**
Extends the webcomponent layer's wfsuserorgroupfromgrouplocator component and allows the user to locate for a workflow users who are members of a specific group. The container (and contained component) is nested in the `startwpwfperformers`, page 236, `finishwpworkflowtask`, page 192, the `WP` layer's `forwardwpworkflowtask`, `rejectwpworkflowtask`, page 227, and `repeatwpworkflowtask`, page 228 components. The container extends the webcomponent layer's wfsuserorgroupfromgrouplocator component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objecttype</code></td>
<td>Type of the selected object</td>
</tr>
<tr>
<td><code>groupName</code></td>
<td>Name of group in which to locate users or groups</td>
</tr>
<tr>
<td><code>flatlist</code></td>
<td>Boolean: true to list all objects in a flat list, false to display a drop down list.</td>
</tr>
<tr>
<td><code>multiselect</code></td>
<td>Boolean: true to support multiple object selection</td>
</tr>
<tr>
<td><code>filterPerformers</code></td>
<td>The values for this parameter are supplied by the component that nests this locator.</td>
</tr>
<tr>
<td><code>filterSubquery</code></td>
<td>Query to filter users or groups. The subquery narrows down the results that are displayed by the locator.</td>
</tr>
</tbody>
</table>
**wpwftemplatelocator and container**

**Purpose**
Extends the webcomponent layer’s allwftemplatelocator component and allows the user to locate a workflow template. The container (and contained component) is launched by the action. It is also nested in the addLocale and container, page 158 component. The container extends the webcomponent layer’s locatorcontainer component.

**Parameters**
This component inherits the contract parameters for the webcomponent layer’s sysobjectlocator component.

**Elements**
This component inherits the configurable elements of the webcomponent layer’s sysobjectlocator component.

**wpworkflowhistory**

**Purpose**
Displays the history (audit trail) of a workflow. This component is called by the action workflowhistory, page 148.

**Parameters**

<table>
<thead>
<tr>
<th>objectId</th>
<th>Required. ID of the selected workflow.</th>
</tr>
</thead>
</table>

**wpworkflowreportclassic**

**Purpose**
Extends workflowstatusclassic, page 262 and generates a report of Web Publisher workflows. This component is nested within the workflowreportcriteria, page 262 component.

**Parameters**
This component inherits the contract parameters for workflowstatusclassic, page 262.
Elements

This component inherits the configurable elements of the webcomponent layer’s sysobjectlocator component.

wpworkflowtemplatelist

Purpose

Extends the Webtop layer’s objectlist component and displays a list of workflow templates. This component is called as a node in the adminconsole, page 162 component.

Parameters

This component inherits the parameters of the Webtop layer’s objectlist component.

Elements

<table>
<thead>
<tr>
<th>&lt;objecttype&gt;</th>
<th>Object type to display by default (default = wcm_locale)</th>
</tr>
</thead>
</table>
abortfailedautowpworkflowtask action (wp), 106
abortworkflow action (wp), 106
abortwpworkflow component (wp), 156
about component (wp), 157
aboutactivecriteria component (wp), 157
abouttoexpirecriteria component (wp), 157
acceptworkflowtask action (wp), 106
Access Control Lists, 19
See also permissions
aclbrowse action (wp), 107
ACLs, 19
See also permissions
actionbutton control (wp), 95
actions
Web Publisher, 105
wpcontext scope, 40
actions, context-sensitive (wp), 181
active, 20
activeobjectscriteria component (wp), 158
add_locale component and container (wp), 158
add_manual link component (wp), 159
add_manual_link action (wp), 107
add_manual_link_type action (wp), 107
add_manual_link_type component (wp), 159
add_translation action (wp), 107
add_translation component and
addtranslationcontainer (wp), 159
addedition action (wp), 108
addedition component and container (wp), 160
addlocale action (wp), 108
addnewrule action (wp), 108
addnewrule component and
addnewruledialog container (wp), 160
addpresentation component and container (wp), 161
addrule action (wp), 109
addrule component and container (wp), 161
addtochangeset action (wp), 109
addtochangeset component (wp), 161
addtoclipboard action (wp), 109
adminconsole component (wp), 162
administration component (wp), 162
advsearch component (wp), 163
advsearchcs component and container (wp), 163
advsearchsup component and container (wp), 164
alerts action (wp), 110
alerts component (wp), 164
aliassetbrowse action (wp), 110
annotations
iMarkup, 23
appicon control (wp), 95
applauncher action (wp), 110
applications
configuration, Web Publisher, 33
applylifecycleall component (wp), 156, 165
applypresentation action (wp), 111
applypresentationcomponent and container (wp), 165
approved, 20
associateworkflow action (wp), 111
associateworkflow component (wp), 165
associations action (wp), 111
associations container (wp), 166
associationslocatorcontainer (wp), 166
attributes (wp dm_document, dm_category), 166
attributes (wp dm_taxonomy), 167
attributes (wp wcm types), 167
Index

attributes action (wp), 112
authenticate (wp), 168
availableformatter control (wp), 96

B
browsertree (wp), 168

C
categories (wp), 169
categories action (wp), 112
category_classic (wp), 170
category_streamline (wp), 170
categoryassietlocator component and container (wp), 169
categorylocaleslist action (wp), 112
categorylocaleslist component (wp), 171
changeset_files_list component (wp), 172
changeset_supportingfiles_list component (wp), 173
changesetaddfilescontainer component (wp), 171
changesetaddsupportingfilescontainer component (wp), 171
changesetdetailscontainer action (wp), 113
changesetdetailscontainer component (wp), 172
changesetfilter component (wp), 173
changesetheader control (wp), 96
changesetindicon control (wp), 96
changesetremovefiles action (wp), 113
changesetremovesupportingfiles action (wp), 113
changesets_list component (wp), 173
changesetstatusmenu control (wp), 97
channel_classic component (wp), 174
channel_streamline component (wp), 175
channelclassificfilter component (wp), 175
channelfilter component (wp), 175
channellocations control (wp), 97
cis component and container (wp), 176
CIS mapping, 51
clearallalerts action (wp), 113
completefailedautoworkflowtask action (wp), 114
components
  Web Publisher, 155
configurations component (wp), 177
contained_in_content_list component (wp), 177
contains_content_list component (wp), 178
content
  in Docbase, 16
Content Rendition Services, 15
Content Server, 15
centfilter component (wp), 178
contentheader control (wp), 97
contentlist component (wp), 178
contenttemplateassetlocator component and container (wp), 179
contenttemplatecriteria component (wp), 179
contenttemplatelocator component and container (wp), 180
contenttemplatereport component (wp), 180
context
  WCM application, 91
contextsensitiverole_config component (wp), 180
contextsensitiveview_config component (wp), 181
controls
  Web Publisher, 95
cookies, 34
copynonwcm action (wp), 114
copynonwcm component (wp), 183
CRS, 15
  See also Content Rendition Services
cscategory_classic (wp), 184
cschannellist_classic (wp), 184
csmfiles_classic (wp), 184
cssubscriptions_classic (wp), 185

D
debugtools component (wp), 185
deeppromote component and container (wp), 185
delegatetworkflowtask action (wp), 114
delete (locale) action (wp), 116
delete action (wp), 115
delete action (wp, dm_category), 115
delete component and container (wp), 186
delete_alerts action (wp), 115
delete_locale component (wp), 186
delete_translation action (wp), 116
deleterenditions (wp), 116
deleteversion action (wp), 117
demote component and container (wp), 186
descriptive names, 22
difflocal action (wp), 117
diffversions action (wp), 117
Docbases
federated, 17
overview, 16
document lifecycles, 17
See also lifecycles
drlview action (wp), 118
dropdownlist control (wp), 98
dynamic performers, 47
custom classes, 83
filter classes, 86
initialization classes, 83

e
edit_manual_link_type component (wp), 188
editfile action (wp), 118
editing applications
external (third-party), 20
internal, 20
editionlist component (wp), 156, 187
deditionslist component (wp), 187
editoptions component (wp), 188
Editor rules files
used in editing application, 20
editor, Web Publisher
configuring, 53
custom controls, 69
editorpresentations component (wp), 189
editors
eWebEditPro, 23
effective dates
in publishing, 21
environment settings, 33
eWebEditPro, 20, 23
expiration dates, 21
expirenow action (wp), 118
expirenow component and container (wp), 189
export action (wp), 119
exportedition action (wp), 119
exportedition component (wp), 190
exportrendition (wp), 119
external editing applications, 20
externalpresentations component (wp), 190
externalresources_settings component (wp), 191
F
federated Docbases, 17
fileformatcriteria component (wp), 191
fileformatreport component (wp), 191
fileformats_settings component (wp), 192
files
local, 17
name, 22 to 23
filtercontainer component (wp), 192
filtervalue control (wp), 98
finishwpworkflowtask action (wp), 120
finishwpworkflowtask component (wp), 192
folderheader control (wp), 98
foldermaps component (wp), 193
forwardwpworkflowtask action (wp), 120
forwardwpworkflowtask component (wp), 193
G
general_preferences component (wp), 194
general_settings component (wp), 194
H
help-index component (wp), 194
hiddentool component (wp), 194
history action (wp task), 121
history action (wp), 120
history component (wp), 195
home cabinets, 17
HTML editors
eWebEditPro, 23
HTML source
eWebEditPro, 20
WYSIWYG editor, 20
I
IAliasPerformerFilter, 86
IAliasPerformerInit, 83
ice_custom_config component (wp), 195
iceconsole component (wp), 198
icedummy component (wp), 198
iceeditcomponent (wp), 198
iceeditpage (wp), 198
icelist (wp), 199
iceproperties (wp), 199
icestatusbar (wp), 199
icetitlebar (wp), 200
IDynamicPerformerFilter, 87
IDynamicPerformerInit, 84
image control (wp), 98
iMarkup, 23
import action (wp wcм category), 122
import action (wp), 121
importrendition action (wp), 122
importrendition component and container (wp), 200
importrenditionbutton action (wp), 122
importsupportfiles component (wp), 200
in-context editing
overview, 20
inboxclassic component (wp), 201
inboxstreamline component (wp), 201
indicate_ w p v i e w component (wp), 201
instructionslist component (wp), 202
internal editing applications, 20
invalidobjectscriteria component (wp), 202
invalidobjectsreport component (wp), 203

J
jobcriteria component (wp), 203
jobreport component (wp), 203
jobs, 21
See also publishing jobs
in publishing, 21

L
latestalerts component (wp), 204
lifecyclesassetsetlocator component and container (wp), 204
lifecyclebrowse action (wp), 123
lifecyclereport component (wp), 204
lifecyclereportcriteria component (wp), 205
lifecycles
overview, 17
states, 19
lifecycles_settings component (wp), 205
linking, 18
list_manual link component (wp), 205
list_manual_link action (wp), 123
list_related_content action (wp), 124
list_related_content component (wp), 206
listobjectsreport component (wp), 207
local files, 17
localeformatter control (wp), 99
localelist component (wp), 207
localemenu control (wp), 99
locationsview action (wp), 124
login component (wp), 208
logoff component (wp), 208

M
main component (wp), 209
makeavailable action (wp), 124
makeunavailable action (wp), 125
makewftemplate_available action (wp), 125
makewftemplate_unavailable action (wp), 125
manuallinktypelist component (wp), 209
menubar component (wp), 209
menugroup control (wp), 99
menuitemtransform action (wp), 125
menuitemtransform_button action (wp), 126
message component (wp), 210
multisite_effectivity component and container (wp), 210
myfiles_classic component (wp), 210
myfiles_streamline component (wp), 211
nameuniquevalidator control (wp), 100
newcategory action (wp), 126
newcategory component and container (wp), 211
newcategorycontainer (wp), 212
newcategoryfolder action (wp), 126
newcategoryfolder component (wp), 212
newcategoryfoldercontainer component (wp), 213
newchangeset action (wp), 127
newchangeset component (wp), 213
newchannel action (wp), 127
newchannel component (wp), 213
newchannelcontainer component (wp), 213
newchannelfolder action (wp), 127
newchannelfolder component (wp), 214
newchannelfoldercontainer component (wp), 214
newchannelpublish component (wp), 215
newcontent action (wp), 128
newcontent component and container (wp), 215
newcontentmanager component (wp), 215
newcontentoptions component (wp), 216
newcontenttemplate action (wp), 128
newfolder component (wp), 156, 216
newtaxonomy action (wp), 128
newtaxonomy component (wp), 217
newtaxonomycontainer component (wp), 217
properties
  in Docbase, 16
properties action (wp dmr_content), 130
properties action (wp), 130
properties component (dm_document, wp), 222
properties component (wp dm_document), 223
properties component (wp wcm_edition, wcm_locale), 222
properties component (wp), 221
propertiescomponent (wp), 224
protected_assets action (wp), 131
protectedassetlist component (wp), 224
protectedassets component (wp), 224
protectedassetsproperties component (wp), 224
publicsupportingfiles component (wp), 225
publish action (wp), 131
publish component and container (wp), 225
publishing
  overview, 21
publishing action (wp), 132
publishing component (wp), 225
publishingconfig component (wp), 226
publishingaction (wp), 131
quicklinks component (wp), 226
readonlypermissions component (wp), 226
reinstate action (wp), 132
reinstate component (wp), 227
rejectwpworkflowtask action (wp), 132
rejectwpworkflowtask component (wp), 227
remove_manual_link action (wp), 133
remove_manual_link_type action (wp), 133
removedition component (wp), 227
removeedition action (wp), 133
removefilefromchangeset action (wp), 133
removeprotected... actions (wp), 134

O
objectlist component (wp), 218
objects, 16 to 17
type, 17

P
performers, dynamic, 47
permissions, 19
permissions action (wp), 128
portlet control (wp), 100
portletnewcontentcontainer (wp), 218
powerpromote action (wp), 129
powerpromote component and container (wp), 218
preferences component (wp), 156, 219
presentations action (wp), 129
presentations component (wp), 219
presentationsfolder component (wp), 220
preview control (wp), 100
previews component (wp), 220
privileges, 19
  See also permissions
promote action (wp), 129
promote component and container (wp), 221
promotetoactivecriteria component (wp), 221
Index

removesupportingfilefromchangeset
action (wp), 134
renditionpublishedname control
(wp), 101
renditions action (wp), 134
renditions component (wp), 228
repeatwpworkflowtask action, 135
repeatwpworkflowtask component
(wp), 228
replicas, 18
replications, 18
reports component (wp), 228
reportselection component (wp), 229
request_translation action (wp), 135
request_translation component (wp), 229
rerunfailedautoworkflowtask action
(wp), 135
restartchangeset action (wp), 136
rmpresentation action (wp), 136
rmrule action (wp), 136
rmtaxonomywebcabinet action (wp), 136
rolemenu component (wp), 229
roles
  adding in WP, 44
  change workflow supervisor (WP), 45
  role-based actions (WP), 43
  role-based components (WP), 44
  set login locale (WP), 45
  Web Publisher administration, 41
  Web Publisher default, 41
roles, context—sensitive (wp), 180
rulelist component (wp), 229
rules action (wp), 137
rules component (wp), 230
Rules Editor, 20

S

SCS, 15
  See also Site Caching Services
search component (wp), 230
searchcs component (wp), 231
searchsup component (wp), 232
security, 19
  See also permissions
selectactions component (wp), 232
sendlocator action (wp), 137
sessionping component (wp), 233
setdefault_locale action (wp), 137
settings component (wp), 233
shortcuts, 18
showpreferences action (wp), 137
showtaskstatus action (wp), 138
singleselectmenu control (wp), 101
singleselectmenuitem control (wp), 101
Site Caching Services, 15, 21
sitemap component (wp), 234
siteoverview action (wp), 138
siteoverview component (wp), 234
sites
  publishing, 21
specialapp, 102
specialapp (wp), 97
stagedobjectscriteria component (wp), 235
staging, 20
standardview action (wp), 138
startworkflownotemplate action (wp), 138
startwpwftemplatelocatorcontainerclassic
component (wp), 239
startwpwftemplatelocatorcontainergenericclassic
component (wp), 239
startwpwftemplatelocatorcontainergenericstreamline
component (wp), 240
startwpwftemplatelocatorcontainerstreamline
component (wp), 240
startwpworkflow classic action (wp), 140
startwpworkflow classic component
(wp), 240
startwpworkflownotemplate action
(wp), 139
startwpworkflownotemplatecs action
(wp), 139
startwpworkflownotemplategeneric
action (wp), 140
startwpworkflownotemplategenericstreamline
action (wp), 140

294 Web Publisher Development Guide
startwpworkflownotemplatestreamline action (wp), 141
startwpworkflowstreamline action (wp), 141
startwpworkflowstreamline component (wp), 241
status control (wp), 102
statusbar component (wp), 242
statusmenu control (wp), 103
streamlineview component (wp), 242
streamlineviewcomponent (wp), 156
subscriptions action (wp), 141
subscriptions component (wp), 242
subscriptions_classic component (wp), 243
subscriptions_streamline component (wp), 243
supcategory_classic component (wp), 244
supchannellist component (wp), 244
supmyfiles_classic component (wp), 244
supporting files, 20
See also templates
supportingfiles component (wp), 245
supportingtemplateassetlocator component and container (wp), 245
supportingtemplateproperties component (wp), 245
supsubscriptions_classic component (wp), 246
system, 16

T

tabbar component (wp), 247
taskmgrcontainerclassic component (wp), 247
taskmgrcontainerstreamline component (wp), 247
taskstatus component (wp), 248
taxonomies (wp), 248
taxonomies action (wp), 142
taxonomywebcabinetslist component (wp), 248
template supporting files, 20
See also templates
templatebrowse action (wp), 142
templatefilter component (wp), 249
templates
creating, 20
templates component (wp), 249
testbed component (wp), 250
thumbnailpreview action (wp), 142
thumbnailpreview component (wp), 250
thumbnailview action (wp), 142
toolbar component (wp), 250
tracing
  WCM flags, 65
  Web Publisher, 63
  Web Publisher flags, 64
transformcontainer component (wp), 251
translationslist action (wp), 143
translationslist component (wp), 251

U

unsafeurlsubstitutor control (wp), 103
updatecontent action (wp), 143
updatecontent component (wp), 252
uploadcontent action (wp), 143
uploadcontent component and container (wp), 252
urlsafevalidator control (wp), 103
usersupportingfiles component (wp), 253

V

validatecontenttemplate action (wp), 144
version labels, 17
versionlabelformatter control (wp), 103
versionnumberformatter control (wp), 104
versions, 17
versions action (wp), 144
versions component (wp), 253
view action (wp task), 146
view action (wp), 144
view_all_of_type action (wp), 145
view_all_of_type component (wp), 254
viewclipboard component (wp), 254
viewcontainer component (wp), 253
viewmanager component (wp), 254
viewmenu component (wp), 255
viewselectedfileslist component (wp), 255
viewsourcem action (wp), 146
viewwplocalelist action (wp), 146

W

WcmApplicationConfig, 33
Web pages, 20
See also files
Index

overview, 20
Web Publisher
actions, 105
CIS mapping, 51
components, 33, 155
configuring editors, 53
context, 39
controls, 95
custom editor controls, 69
dynamic workflow performers, 47
files, 37
roles, 41
Web view customization, 59
Web Publisher Editor, 20
Web View (Web Publisher), 59
web_author_preferences component (wp), 256
Web-based HTML editors, 20
Web-based XML editors, 20
web_developer_preferences component (wp), 256
Web-safe names, 23
webcabinets action (wp), 147
webcabinetslocator component (wp), 256
webhtml_add_hyperlink component (wp), 257
webhtml_add_image component (wp), 258
webhtml_add_image_action component and container (wp), 257
webhtml_subscription_hyperlink_locator component (wp), 259
webhtml_subscription_image_locator component (wp), 258
webhtml_webcabinets_hyperlink_locator component (wp), 260
webhtml_webcabinets_image_locator component (wp), 258
webhtml_wpmyfiles_hyperlink_locator component (wp), 260
webhtml_wpmyfiles_image_locator component (wp), 259
webreradyicon control (wp), 104
webview action (wp), 147
webview_component (wp), 260
wftemplatebrowse action (wp), 148
whereused (wp), 261
whereused action (wp), 148
WIP, 19
workflow
dynamic performers, 47
workflow templates, 18
workflowassetlocator component and container (wp), 261
workflowhistory action (wp), 148
workflowreport component (wp), 262
workflowreportcriteria component (wp), 262
workflows, 18
workflowstatusclassic action (wp), 149
workflowstatusclassic component (wp), 262
workflowstatusstreamline component (wp), 263
wp_applauncher component (wp), 264
wp_cancelcheckout component and container (wp), 265
wp_cedit component (wp), 265
wp_checkin component and container (wp), 265
wp_checkout component and container (wp), 266
wp_diffcontent component and container (wp), 266
wp_edit component and container (wp), 267
wp_editwithce action (wp), 149
wp_editwithre action (wp), 149
wp_editwithrenested action (wp), 149
wp_editwithwe action (wp), 150
wp_export component and container (wp), 267
wp_export_rendition component and container (wp), 268
wp_ice component (wp), 268
wp_import_content component and importcontainer (wp), 269
wp_import_content_template component (wp), 269
wp_import_instructions component and container (wp), 270
wp_import_template component (wp), 271
wp_reedit component (wp), 271
wp_unziprendition component (wp), 271
wp_viewsource component and container (wp), 272
wp_wheedit component (wp), 272
wpallfolderlocator component and container (wp), 273
wpcancelcheckout action (wp), 150
wpcategorylocator component and container (wp), 273
wpchannelfolderlocator component and container (wp), 274
wpcheckin action (wp), 150
wpcheckout action (wp), 151
wpclipboard component and container (wp), 274
wpcontenttemplatelocator component and container (wp), 275
wpcontext, 39
configuring actions, 40
wpcopy component (wp), 275
wpcopyaction (wp), 151
wpdeeppromote action (wp), 151
wpdemote action (wp), 152
wpfolderlocatorcontainer component (wp), 275
wpinstructionlocator component and container (wp), 276
wpcontext, 39
configuring actions, 40
wpmove component (wp), 278
wpmoveaction (wp), 152
wplocalelist component (wp), 277
wpmovecomponent (wp), 278
wpmoveaction (wp), 152
wpmultiargdialogcontainer component (wp), 278
wpmyfileslocator component (wp), 279
wpnavigationcontainer component (wp), 279
wppresentationtemplatelocator component and container (wp), 280
wppromote action (wp), 152
wpsubscriptionlocator component (wp), 280
wpsysobjectlocator component and container (wp), 281
wpsysteminfo component (wp), 281
wptaskattachment component (wp), 282
wptaskcomponentcontainer component (wp), 282
wptaskmanagerchangesetremovefiles action (wp), 153
wptaskmanagerclassic component (wp), 283
wptaskmanagerstreamline component (wp), 283
wptaskmgrabortworkflow component (wp), 283
wptasksupportingfilesclassic component (wp), 283
wptasksupportingfilesstreamline component (wp), 284
wpwftemplatelocator component and container (wp), 286
wpwfvuserorgroupfromgrouplocator component and container (wp), 286
wpworkflowhistory component (wp), 287
wpworkflowreportclassic component (wp), 287
wpworkflowtemplatelist component (wp), 288
WYSIWYG editors, 20
XML editors
eWebEditPro, 23
XML source
eWebEditPro, 20