Oracle® Fusion Middleware

Developing with Oracle WebCenter Content

11*g* Release 1 (11.1.1)

E26694-01

March 2013

ORACLE®

Oracle Fusion Middleware Developing with Oracle WebCenter Content, 11*g* Release 1 (11.1.1)

E26694-01

Primary Author: Bonnie Vaughan

Contributing Authors: Sean Cearley, Sandra Christiansen, Will Harris, Karen Johnson, Jean Wilson

Contributors: Sharmarke Aden, Scott Nelson, Matt Shannon, Matt Shannon, David Truckenmiller, Ron van de Crommert

Contents

6 Creating Dynamic Server Pages

Part V Customizing Content Server with Components

11 Getting Started with Content Server Components

12 Enabling and Disabling Components for Content Server

13 Updating Component Configurations

14 Customizing Content Tracker

29 Using RIDC to Access Content Server

30 Using the Content Server JCR Adapter

31 Configuring Web Services with WSDL, SOAP, and the WSDL Generator

B Building a Website

Preface

Oracle WebCenter Content Server is highly functional out of the box, but you can tailor it to your site requirements in many different ways. This developer's guide provides information to help you customize your Content Server instance.

Audience

This guide is intended for developers and administrators who want to customize Oracle WebCenter Content Server software to suit content management needs that are specific to their business or organization.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following Oracle WebCenter Content 11*g* Release 1 (11.1.1) documents, which are part of the Oracle Fusion Middleware documentation set:

- *Oracle Fusion Middleware Administering Oracle WebCenter Content*
- *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*
- *Oracle Fusion Middleware Managing Oracle WebCenter Content*
- *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*

For additional information, also see the following Oracle Fusion Middleware 11*g* Release 1 (11.1.1) documents:

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Getting Started With JAX-WS Web Services for Oracle WebLogic Server*
- *Oracle Fusion Middleware Interoperability Guide for Oracle Web Services Manager*
- *Oracle Fusion Middleware Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*
- *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*
- *Oracle Fusion Middleware Third-Party Application Server Guide*

Conventions

This document uses the following text conventions.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

New and Changed Features

This section introduces the new and changed features of Oracle WebCenter Content that are covered in this guide for Oracle WebCenter Content Server developers.

New Features for 11*g* Release 1 (11.1.1)

11*g* Release 1 (11.1.1) includes the following new features:

- This guide combines information that was previously contained in the following documents:
  - *Dynamic Server Pages Guide*
  - *Getting Started with the Stellent Developer's Kit (SDK)*
  - *Idc Command Reference Guide*
  - *Modifying the Content Server Interface*
  - *Oracle Fusion Middleware JCR Adapter Guide for Content Server*
  - *Oracle Fusion Middleware Developer's Guide for Remote Intradoc Client (RIDC)*
  - *Using WSDL Generator and SOAP*
  - *Working with Components*
- Web services

  WebCenter Content uses Oracle WebLogic Server web services. For more information, see Chapter 25, "Configuring WebCenter Content Web Services for Integration."
- ComponentTool

  The ComponentTool utility has been added to provide a command-line tool for installing, enabling, and disabling components. For more information, see Chapter 11, "Getting Started with Content Server Components."
- Oracle WebCenter Content Server deployment: Content Server is deployed with WebCenter Content to an Oracle WebLogic Server domain, which means changes in configuring and administering Content Server. For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

Changed Features for 11*g* Release 1 (11.1.1)

11*g* Release 1 (11.1.1) includes the following changes:

- Oracle WebCenter Content directories and files: WebCenter Content 11*g* Release 1 (11.1.1) is provided as part of a full media installation of WebCenter Content, with the WebCenter Content directories and files. The directory structure for a WebCenter Content 11*g* instance is different from an Oracle Universal Content Management 10*g* instance. For a description of terms and path names that are important for understanding and working with the WebCenter Content structure, see Section 1.1.1.1, "Terminology for WebCenter Content Directories."
- SOAP: SOAP is provided with Oracle WebLogic Server, not in Oracle WebCenter Content.
- Web Form Editor: The Web Form Editor user interface and FCKEditor are not supported.
- CIS deprecated: Content Integration Suite (CIS) has been deprecated. Developers and system integrators are directed to use Remote Intradoc Client (RIDC), which provides a thin communication API for communication with Oracle WebCenter Content Server. For details, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*. For more information, see Section 29, "Using RIDC to Access Content Server."
- Oracle WebCenter Content is now part of the Oracle WebCenter product stack to provide the most complete, open, and unified enterprise content management platform. The Oracle WebCenter Content Server software and documentation have been rebranded accordingly.
- Idoc Script information has been added to Part II, "Working with the Idoc Script Custom Scripting Language" and descriptions of Idoc Script functions and variables have been added to Appendix A, "Idoc Script Functions and Variables." This Idoc Script documentation was previously in the *Oracle WebCenter Content Idoc Script Reference Guide*.

Part I

Getting Started with Customizing Oracle WebCenter Content

This part provides an overview of customizing Oracle WebCenter Content, including Oracle WebCenter Content Server, and describes the tools and resources that are available for customization. This part also describes the Oracle Fusion Order Demo sample application.

Part I contains the following chapters:

1

Introduction to Oracle WebCenter Content

This chapter provides an overview of Oracle WebCenter Content and describes skills, tools, and resources for customizing Oracle WebCenter Content Server.

This chapter includes the following sections:

- Section 1.1, "Overview of WebCenter Content Architecture"
- Section 1.2, "Customization Types"
- Section 1.3, "Customization Planning"
- Section 1.4, "Recommended Skills and Tools for Customizing Content Server"
- Section 1.5, "Content Server Behavior"

For information about troubleshooting aids, see Appendix C, "Troubleshooting."

1.1 Overview of WebCenter Content Architecture

Before beginning a customization project, you need to understand the architecture of WebCenter Content and how it works. To create a customization efficiently and effectively, you should have an understanding of the WebCenter Content directories and files, available resources, and Content Server behavior.

Content Server, the web user interface for WebCenter Content, is deployed as an application to an Oracle WebLogic Server domain. For information about how Content Server works, see Section 1.5, "Content Server Behavior."

WebCenter Content can also be deployed to an IBM WebSphere Application Server. For more information, see "Managing Oracle WebCenter Content on IBM WebSphere" in the *Oracle Fusion Middleware Third-Party Application Server Guide*.

1.1.1 WebCenter Content Directories and Files

When you create custom components or dynamic server pages, you work primarily with WebCenter Content files in these directories:

- `bin`
- `config`
- `components`
- `resources`
- `weblayout`

Caution: Modifying the default variables in these files can cause WebCenter Content to malfunction. For more information about configuration variables, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

1.1.1.1 Terminology for WebCenter Content Directories

Oracle WebCenter Content documentation uses the following terms when referring to variables in the directories associated with the Oracle WebCenter Content installation, configuration, and deployment:

- *IdcHomeDir*: This variable refers to the `ucm/idc/` directory in the WebCenter Content Oracle home, where the Content Server media is located. The server media can run Oracle WebCenter Content Server, Oracle WebCenter Content: Inbound Refinery, or Oracle WebCenter Content: Records. This is essentially a read-only directory. The default location is *WCC_ORACLE_HOME*`/ucm/idc/`. The variable portion of the default location can be changed, but the path cannot be changed from `ucm/idc/`.
- *DomainHome*: The user-specified directory where an Oracle WebCenter Content application is deployed to run on an application server. The *DomainHome*`/ucm/`*short-product-id*`/bin/` directory contains the `intradoc.cfg` file and executables. The default location for *DomainHome* is *MW_HOME*`/user_projects/domains/base_domain/`, but you can change the path and domain name (`base_domain`) during the deployment of an Oracle WebCenter Content application to an application server.
- *short-product-id*: An abbreviated name for the type of Oracle WebCenter Content application deployed to an application server. This name is used as the context root (default `HttpRelativeWebRoot` configuration value). Possible values follow:
  - `cs` (Content Server)
  - `ibr` (Inbound Refinery)
  - `urm` (Records)
- *IntradocDir*: The root directory for configuration and data files specific to a Content Server instance that is part of an Oracle WebCenter Content application deployed to an application server. This Idoc Script variable is configured for one type of Content Server instance: Content Server (`cs`), Inbound Refinery (`ibr`), or Records (`urm`). The default location of *IntradocDir* is *DomainHome*`/ucm/`*short-product-id*`/`, but the *IntradocDir* directory can be located elsewhere (as defined in the `intradoc.cfg file`). The specified directory must be an absolute path to the instance directory and must be unique to a particular server or node. The directory files include startup files (`intradoc.cfg` and executables).

1.1.1.2 The bin Directory

The `bin` directory is the root directory for Content Server startup files. It contains the `intradoc.cfg` file and the executable files that run Content Server services, applets, and utilities. It is located at *DomainHome*`/ucm/`*short-product-id*`/bin/`, in which *short-product-id* specifies whether it is for Content Server (`cs`), Inbound Refinery (`ibr`), or Records (`urm`). Table 1–1 describes the contents of the `bin/` directory.

Table 1–1 Contents of the bin Directory for Startup Files

Element	Description
Executables	**Services** ■ `IdcServer` ■ `IdcServerNT` **Applet** ■ `IntradocApp` (launches all Admin tools) **Utilities** ■ `BatchLoader` ■ `Installer` ■ `IdcAnalyze` ■ `ComponentWizard` ■ `SystemProperties` ■ `IdcCommand`
`intradoc.cfg` file	Configuration file that contains the settings for Content Server services, applets, and utilities

Note: If Content Server is set up as an automatic service and you attempt to start a Content Server service (`IdcServer` or `IdcServerNT`) from the command line, you will receive an error message: `The port could not be listened to and is already is use.`

The `intradoc.cfg` file is used to define system variables for Content Server, including directory, Internet, and Inbound Refinery settings. Several of these variables can be set using the WebCenter Content System Properties utility. Example 1–1 shows a typical `intradoc.cfg` file.

Example 1–1 intradoc.cfg File

```
<?cfg jcharset="Cp1252"?>
#Content Server Directory Variables
IntradocDir=C:/oracle/idcm1/
WebBrowserPath=C:/Program Files/Internet Explorer/iexplore.exe
CLASSPATH=$COMPUTEDCLASSPATH;$SHAREDDIR/classes/jtds.jar

#Additional Variables
HTMLEditorPath=C:/Program Files/Windows XP/Accessories/wordpad.exe
JAVA_SERVICE_EXTRA_OPTIONS=-Xrs
```

1.1.1.3 The config Directory

The `config` directory is located at *`IntradocDir`*`/config/`. The default location of *`IntradocDir`* is *`DomainHome`*`/ucm/`*`short-product-id`*, but the *`IntradocDir`* directory can be located elsewhere (as defined in the `intradoc.cfg file`). Table 1–2 describes the contents of the `config` directory.

Table 1–2 Contents of the config Directory

File	Description
`config.cfg`	Defines system configuration variables.

The `config.cfg` file is used to define global variables for the Content Server system. Several of these variables can be set using the WebCenter Content System Properties utility or by modifying the variables on the Admin Server General Configuration page. Example 1–2 shows a typical `config.cfg` file.

Example 1–2 config.cfg File

```
<?cfg jcharset="Cp1252"?>
#Content Server System Properties
IDC_Name=idcm1
SystemLocale=English-US
InstanceMenuLabel=JeanWTestSystem
InstanceDescription=idcm1
SocketHostAddressSecurityFilter=127.0.0.1|10.10.1.14

#Database Variables
IsJdbc=true
JdbcDriver=com.internetcds.jdbc.tds.Driver
JdbcConnectionString=jdbc:freetds:sqlserver://jwilsonnote:1433/oracle1;charset=UTF
8;TDS=7.0
JdbcUser=sa
JdbcPassword=UADle/+jRz7Fi8D/VzTDaGUCwUaQgQjKOQQEtI0PAqA=
JdbcPasswordEncoding=Intradoc
DatabasePreserveCase=0

#Internet Variables
HttpServerAddress=jwilsonnote
MailServer=mail.example.com
SysAdminAddress=sysadmin@example.com
SmtpPort=25
HttpRelativeWebRoot=/oracle/
CgiFileName=idcplg
UseSSL=No
WebProxyAdminServer=true
NtlmSecurityEnabled=No
UseNtlm=Yes

#General Option Variables
EnableDocumentHighlight=true
EnterpriseSearchAsDefault=0
IsDynamicConverterEnabled=0
IsJspServerEnabled=0
JspEnabledGroups=

#IdcRefinery Variables

#Additional Variables
WebServer=iis
UseAccounts=true
IdcAdminServerPort=4440
SearchIndexerEngineName=DATABASE
VIPApproval:isRepromptLogin=true
Vdk4AppSignature=SF37-432B-222T-EE65-DKST
Vdk4AppName=INTRANET INTEGRATION GROUP
IntradocServerPort=4444
```

1.1.1.4 The components Directory

The *IntradocDir*/data/components/ directory contains the files that Content Server uses to configure system components. Table 1–3 describes the contents of the components directory.

Table 1–3 Contents of the data/components/ Directory

File	Description
idc*short-product-id*_components.hda	Identifies components that have been added to the Content Server system and whether they are enabled or disabled. Example: idccs_components.hda.
component.hda	Identifies the configuration status for a component.

Example 1–3 shows a *component*.hda file that defines the configuration status for a component called help.

Example 1–3 component.hda File

```
<?hda version="11.1.1.2.0-dev idcprod1 (091209T125156)" jcharset=UTF8
encoding=utf-8?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
@end
@ResultSet Components
2
name
location
help
components/help/help.hda
@end
```

1.1.1.5 The resources Directory

The *IdcHomeDir*/resources/ directory contains two directories: admin/ and core/.

The resources/core/ directory contains files that Content Server uses to assemble web pages. Table 1–4 describes the subdirectories of the resources/core/ directory.

Table 1–4 Contents of the resources/core/ Directory

Subdirectory	Description
config/	Holds base configuration files for Content Server.
idoc/	Holds Idoc Script dynamichtml and dynamicdata definitions.
install/	Holds files that are used by the installer and related applications.
javascript/	Holds files that are processed by the publishing engine and end up in the weblayout directory as raw JavaScript files.
jspserver/	Holds jspserver XML files.
lang/	Holds localized string definitions for Content Server.
reports	Holds templates for Content Server reports.
resources	Holds resource definitions (queries, page resources, services, and other resource data) for Content Server.
tables	Holds Idoc Script resource table definitions.

Table 1–4 (Cont.) Contents of the resources/core/ Directory

Subdirectory	Description
`templates`	Holds templates for all Content Server pages (except reports).

Table 1–5 describes the subdirectories of the `resources/admin/` directory.

Table 1–5 Contents of the resources/admin/ Directory

Subdirectory	Description
`idoc`	Holds Idoc Script `dynamichtml` and `dynamicdata` definitions.
`tables`	Holds Idoc Script resource table definitions.
`templates`	Holds templates for all Content Server pages (except reports).

1.1.1.6 The weblayout Directory

The *`DomainHome`*`/ucm/`*`short-product-id`*`/weblayout/` directory contains the files that are available to the web server for display on the various pages of the Content Server website. Table 1–6 describes the subdirectories of the `weblayout/` directory.

Table 1–6 Contents of the weblayout Directory

Subdirectory	Description
`groups`	Holds the web-viewable content items and dynamic server pages.
`images`	Holds images, such as icons and home page graphics.
`resources`	Holds layouts, skins, and schema information.

1.1.2 Resources

Resources are files that define and implement the actual customization you make to Content Server. They can be pieces of HTML code, dynamic page elements, queries that gather data from the database, services that perform Content Server actions, or special code to conditionally format information.

Resources are a critical part of the Content Server software, so you must be familiar with them before you attempt to create a custom component or dynamic server page. You can create, edit, or remove a resource file by using the Component Wizard. You also can use the Component Wizard as a starting point for creating custom resources.

Resources fall into eight distinct categories, which table describes. The first five types listed in the table are also called *resource*-type resources. Table 1–7 describes these resource types.

Table 1–7 Resource Types

Resource Type	Description	Example of Standard Resource
HTML Include	Defines pieces of HTML markup and Idoc Script code that are used on multiple Content Server web pages.	*IdcHomeDir*/resources/core/idoc/std_page.idoc
Dynamic Data Table	Defines a table of data in a `dynamicdata` include from within Idoc Script to load an HTML table definition, interface menu actions, or information about metadata fields or from within Java code as an alternative to static tables loaded into `SharedObjects`.	*IdcHomeDir*/resources/core/idoc/std_data.idoc
String	Defines localized strings for the user interface and error messages.	*IdcHomeDir*/resources/core/lang/cs_strings.htm
Dynamic Table (HDA format)	Provides dynamic (frequently changed) content in table format to Content Server.	*IdcHomeDir*/resources/core/templates/templates.hda
Static Table (HTML format)	Provides static (seldom changed) content in table format to Content Server.	*IdcHomeDir*/resources/core/tables/std_locale.htm
Query	Defines database queries.	*IdcHomeDir*/resources/core/tables/query.htm
Service	Defines scripts for services that can be performed by Content Server.	*IdcHomeDir*/resources/core/tables/std_services.htm
Template	Defines templates, which contain the code that Content Server uses to assemble a particular web page.	*IdcHomeDir*/resources/core/templates/checkin_new.htm
Environment	Defines configuration settings for Content Server.	*IntradocDir*/config/config.cfg

1.2 Customization Types

Three major types of alterations can be made to Content Server:

- Altering the look and feel of the product

  You can customize the look and feel of the Content Server interface to meet your organization's specifications. Interface modifications can be as simple as replacing the icons that appear on the standard Content Server web pages or as complex as a complete redesign of the interface.

- Modifying the functionality of the product

By changing how the product functions, you can tailor Content Server to the way your business or organization functions. For example, you can change the default date and time stamp, or change aspects of check-in behavior.

- Integrating the product into your environment

  You can use shell scripts, SOAP, J2EE, JSP, and clusters to more fully integrate Content Server into your site's current environment.

1.3 Customization Planning

Before approaching customization, it is important to clarify exactly *why* the customization is being undertaken. For example, to add corporate branding, you can use the Modify Layout Samples to do so. Or to change security features, you can use components to modify the default security settings.

Customization often occurs to make Content Server match the business practices of an organization. Often, after evaluating your business processes, you may find that sometimes it is more efficient to slightly alter your procedures before customizing Content Server.

There are six major stages in customization:

1. Determine why you want to customize.

   Is there corporate personalization to be done? Is there a better way to present navigation options or material? Depending on what type of need you find, you can determine which tools are best to use.

   Oftentimes the cosmetic details that you change are the ones that can most satisfy your users; changing items such as layout, colors, and images often provide the effect that users are looking for.

2. Plan the customization carefully.

   Take into account those aspects of the Content Server environment that might be changed even peripherally by the customization. All customization should be done in a test environment, separate from the site's production environment.

3. Check to see if a solution may be available.

   The samples on the Support website contain many types of customizations. You might be able to use an existing component with just a little editing. A number of samples are provided on an as-is basis. These are components or files that demonstrate, enhance, or extend the functionality of WebCenter Content.

4. Evaluate the problem and how essential it is to solve.

   Some problems might require more effort to fix than the resulting payback. Perhaps customization is not needed, but simply a minor change in business practices.

5. Test the customization thoroughly in a separate environment.

   If possible, have end users assist with the testing. When the testing has passed all criteria for release, inform users about the changes and how to implement them.

6. Document the customization that you create.

   All alterations should be documented as completely as possible, both within the actual customization (for example, as a comment in a dynamic server page or in a component) and as a separate `README` document. This documentation provides a historical audit trail for others who might need to add to the customization later.

1.4 Recommended Skills and Tools for Customizing Content Server

Content Server brings together a wide variety of technologies to deliver advanced functionality. To modify Content Server, certain experience and skills with some or all of these technologies are required.

The technical skills required to customize Content Server can vary depending on the complexity of the customization. For example, much customization can be accomplished with knowledge of HTML and Idoc Script.

This list describes, in descending order of importance, the technologies and experience you might need to modify Content Server:

- Content Server architecture

  You should thoroughly understand how Content Server works and how components and dynamic server pages function before you begin customizing your system.

- HTML and cascading style sheets (CSS)

  You will need a good understanding of HTML and CSS to make changes to the Content Server web page templates. The templates are not complex in their use of HTML, but they make constant use of HTML tables and frequent use of forms. The `std_page.idoc` and `std_css.idoc` files include cascading style sheets to control the look and feel of the default templates, including fonts and layouts.

- Idoc Script

  Idoc Script is the custom, server-side scripting language for Content Server. Almost every Content Server web page includes some Idoc Script code, which provides the methods for processing various page elements.

- JavaScript

  The internal content of most Content Server pages do not use JavaScript, but the Search, Check-In, and Update pages are notable exceptions. You must have an understanding of JavaScript before you create a customization that is called in place of these pages. Also, you must understand JavaScript to alter layouts. Changing layouts relies heavily on JavaScript and cascading style sheets for design and navigation.

- Structured Query Language (SQL)

  Content Server uses SQL to perform queries on the database. Knowledge of SQL can help you understand the standard queries and create your own custom queries.

- Java programming

  Content Server is implemented with Java classes. You should have a thorough understanding of Java and the Content Server Java class files before attempting to make any changes to the underlying functionality. However, you can customize the product extensively without working with Java.

- Other programming

  Experience with other tools, such as Visual Basic, COM, .Net, C++, or VBScript, might be helpful if you are doing complex customization or integrating WebCenter Content with other systems.

You may find the following tools useful when customizing Content Server:

- Text editor

Most product customizing can be done with a normal text editor, such as Microsoft WordPad or vi.

- HTML editor

  Caution: Graphical HTML editor programs often change the source HTML, which can cause Idoc Script tags to be converted into strings of characters that are no longer recognized by Content Server. If you use a graphical editor, make sure you edit in a nongraphical mode.

  If you prefer to use a graphical HTML editor to work with HTML pages, use a nongraphical mode for editing.

- Multiple browsers

  You should test customization on multiple versions of any web browsers that might be used to interface with the content management system. Internet Explorer, Firefox, and Chrome do not display content in the same manner, and different versions of the same browser may exhibit different behaviors.

- JavaScript debugger

  A JavaScript debugger can ease the task of JavaScript development. A number of JavaScript debuggers are available for download from the Internet.

- Integrated Development Environment (IDE) for Java

  If your customization requires the development of Java code, you need an appropriate Java development environment.

1.5 Content Server Behavior

Before you customize WebCenter Content, you need to understand the behavior of Oracle WebCenter Content Server:

- Startup behavior
- Resource caching
- Page assembly
- Database interaction
- Localized string resolution
- Application integrations

For an overview of Content Server administration, see "Overview of System Administration Tasks" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

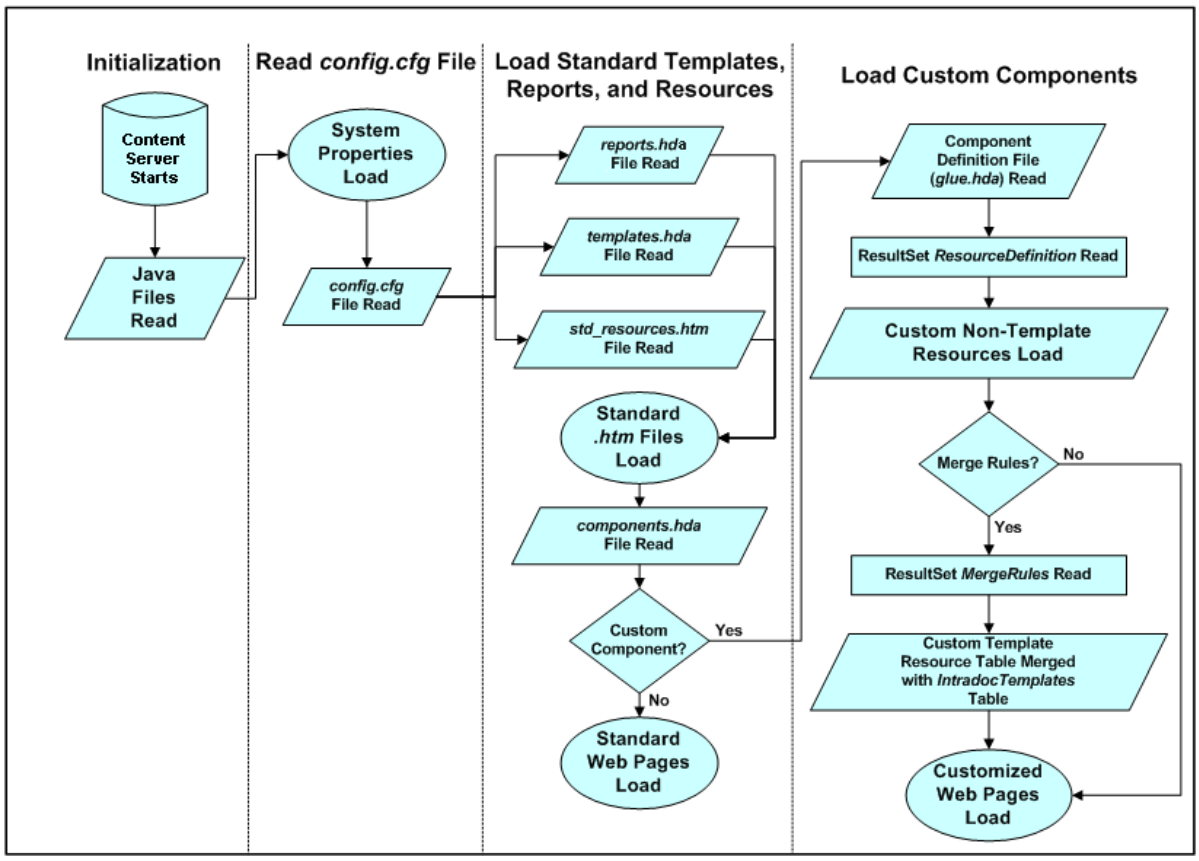
1.5.1 Startup Behavior

During startup, a Content Server instance performs internal initialization and loads these items:

1. Configuration variables
2. Standard templates, resources, and reports
3. Custom components, including templates, resources, configuration variables, and reports

Figure 1–1 illustrates the four steps that Content Server goes through during startup. Section 1.5.1.1, "Startup Steps," describes each step in more detail.

Figure 1–1 Content Server Startup Behavior



1.5.1.1 Startup Steps

During startup, Content Server goes through these steps:

1. Internal initialization occurs.

   When Content Server initializes internally, the Java class files from Content Server are read and the Java Virtual Machine (JVM) is invoked. Any variables in the *DomainHome*`/ucm/`*short-product-id*`/intradoc.cfg` file are initialized as well.

2. Configuration variables load.

   After initialization, Content Server loads the `config.cfg` file from the *IntradocDir*`/config/` directory. This file stores the system properties and configuration variables, which are defined by name/value pairs (such as `SystemLocale=English-US`).

   The default information contained within the configuration file was supplied during the Oracle WebCenter Content installation process, but you can modify this file in several ways:

   - Use the Admin Server General Configuration page, accessible from the **Administration** menu

- Run the SystemProperties executable, located in the bin directory of the Oracle WebCenter Content installation (UNIX operating system)
- Edit the configuration files directly
- Use a custom component

Any time changes are made to the config.cfg file, you must restart Content Server for the changes to take effect.

3. Standard resources, templates, and reports load.

   For Content Server to function properly, many standard resources, templates, and reports must be loaded. After the configuration settings have been loaded, Content Server reads the entries in the *IdcHomeDir*/resources/core/templates/templates.hda file and the *IdcHomeDir*/resources/core/reports/reports.hda file. These files tell Content Server which templates to load, which in turn loads any standard resources referenced in the template and report pages.

4. Custom components load.

   Content Server loads all of the components listed in *IntradocDir*/data/components/idc*short_product_id_*components.hda, and in turn that loads system components out of *IdcHomeDir*/components/*ComponentName*/*ComponentName*.hda or, for custom components, out of *IntradocDir*/custom/*ComponentName*/*ComponentName*.hda.

1.5.1.2 Effects of Configuration Loading

It is important to understand the effect of the load order for the different configuration files because if a variable is set in more than one file, the last variable loaded takes precedence. For example, the *IntradocDir*/config/config.cfg file is loaded before the *IntradocDir*/data/components/*component_name*/config.cfg file, so the *component_name*/config.cfg can change the value of a variable that was set by the config/config.cfg file.

Files are loaded in this order (not all files exist for each component):

1. *DomainHome*/ucm/*short-product-id*/bin/intradoc.cfg
2. *IntradocDir*/config/config.cfg
3. *DomainHome*/ucm/*short-product-id*/custom/*component_name*/*_environment.cfg

   Some components might not have this file, or it might be named environment.cfg.

4. *IntradocDir*/data/components/*component_name*/install.cfg
5. *IntradocDir*/data/components/*component_name*/config.cfg
6. *DomainHome*/ucm/*short-product-id*/bin/intradoc.cfg

   This file is reread at the end of startup to allow overrides to other settings.

If, for example, the same variable was set in each of the files in the list, the variable's value in intradoc.cfg would take precedence because this file was loaded last.

To view the configuration, you can use the GET_SYSTEM_AUDIT_INFO service to show all configuration entries and where they were set.

To change a component variable, you can use the **Update Component Configuration** area in the Advanced Component Manager. This area displays a dropdown list of components that have editable configurations in the *component_name*/config.cfg file. You can choose a component and click **Update** to get to the Update Component Configuration page in Content Server.

You can also edit the configuration file manually. If a component is not displayed in the **Update Component Configuration** list in the Advanced Component Manager, you can make your changes directly in one of the configuration files.

1.5.2 Resource Caching

Content Server handles caching for template pages and resources as follows:

- When Content Server loads template pages and resources, they are cached in memory to accelerate page presentation.
- If you change a template page, report page, or HTML include resource, or you check in a revision to a dynamic server page, your changes go into effect immediately.

  The next request for the associated web page or refresh of the page reflects the changes, and the new information is cached. This occurs because pages are assembled dynamically for each page request. You can disable this behavior to improve performance by setting the configuration variable DisableSharedCacheChecking.
- If you change any other component file (including services, queries, environment variables, tables, the idc*short-product-id*_components.hda file, and the template.hda file), you must restart Content Server before the changes go into effect. Such changes could cause Content Server to malfunction if they were implemented immediately.

  You do not need to restart Content Server after changing strings; however, you must republish the ww_strings.js files by clicking **publish string and dynamic files** or **publish string, static, and dynamic files** in the Weblayout Publishing area of the Admin Actions page. For more information, see Chapter 11, "Getting Started with Content Server Components."

1.5.3 Page Assembly

Content Server uses the following information from the files in the *IdcHomeDir*/resources/ directory to assemble dynamic web pages:

- The structure and format of a web page

  This structure and format are defined by a particular HTML template file. The template files are primarily in the resources/core/templates directory, and some templates are in the resources/core/reports and resources/admin/templates directories.
- The templates reference *resources*

  These resources are located in .htm and .idoc files in subdirectories of the resources directory. Resources can include HTML and Idoc Script markup, localized strings, queries to gather information from the database, and special code to conditionally format the information.

As a rule, each web page includes the following resources:

- Standard page header

- Standard page beginning
- Standard page ending

Because all of the Content Server resources are cached in memory at startup, Content Server has a definition for the standard pieces that appear on the page. Content Server then combines the standard resources with the unique resources specified in the template to create the web page.

For dynamic server pages, the template page and custom resource files are checked into Content Server. When one of these pages is requested by a web browser, Content Server recognizes the file extension as a dynamic server page, which enables special processing. At that point, the page assembly process is the essentially the same as the standard process, except that the page can use both the standard resources in the `resources` directory and the custom resources that are checked in to Content Server.

1.5.4 Database Interaction

Some databases, such as Oracle Database, return all column names in uppercase characters. Therefore, when Content Server receives query results from these databases, column names must be mapped from the uppercase characters to the values used in Content Server.

Because of this case mapping issue, custom components created for a Content Server instance using one database might not work correctly on a Content Server instance using a different database.

To map column names, the *`IdcHomeDir`*`/resources/core/resources/upper_clmns_map.htm` file contains a mapping table named `ColumnTranslation`. Add the query values to this file when you create a component that accesses fields that are not WebCenter Content database fields (for example, if you create a component that accesses a custom table within the WebCenter Content database).

For information about using the `upper_clmns_map.htm` file, see Section 7, "Changing System Settings."

1.5.5 Localized String Resolution

Localized strings are the means by which the user interface and error messages are presented in the language specified by the user's locale. Content Server loads the string resource files for a base language and also loads resource files for every supported language. Instead of presenting hard-coded text, the template pages, applets, and error messages reference string IDs in these resource files, which are then resolved using the ExecutionContext that contains the locale information for the user.

1.5.6 Application Integrations

Content Server not only serves as a content management solution for content-centric websites, but also provides a scalable content management infrastructure that supports multiple enterprise applications in many diverse environments and platforms. The integration solutions enable other enterprise applications to access content managed by the content management system and provides these applications with critical content management capabilities such as full-text and metadata searching, library services, workflow, subscription notifications, and content conversion capabilities through a wide array of integration methods.

For information about integrating Content Server with enterprise applications, see Section 24, "Getting Started with Integrating WebCenter Content into Your Environment."

2

Introduction to the Oracle Fusion Order Demo Sample Application

This chapter describes the Oracle Fusion Order Demo sample application, how to set it up, and how to run the Suppliers Module.

This chapter includes the following sections:

- Section 2.1, "About Fusion Order Demo and the Suppliers Module"
- Section 2.2, "Setting Up the Fusion Order Demo Application"
- Section 2.3, "Running the Suppliers Module"

The instructions in this chapter are for setting up and running the sample application on Oracle WebLogic Server. Oracle WebCenter Content can also be deployed to an IBM WebSphere Application Server. For more information, see "Managing Oracle WebCenter Content on IBM WebSphere" in the *Oracle Fusion Middleware Third-Party Application Server Guide*.

2.1 About Fusion Order Demo and the Suppliers Module

In the Oracle Fusion Order Demo sample application, electronic devices are sold through a storefront web application. Customers can visit the website, register, and place orders for products.

To view and run the demo, you need to install Oracle JDeveloper 11*g*. Then you need to download the application for the demonstration. For instructions to complete these tasks, see Section 2.2, "Setting Up the Fusion Order Demo Application."

After the application is installed and running, you can view the code using Oracle JDeveloper. You can view the application at runtime by logging in as an existing customer and placing an order.

The Suppliers module of Fusion Order Demo demonstrates the use of Java EE to create web applications for a web supplier management system. The demonstration application is used to illustrate points and provide code samples.

The Suppliers module consists of a business services project named `Model` and a web user interface project named `ViewController`. You run the Suppliers module of the Fusion Order Demo application in JDeveloper by running the `ViewController` project. The `ViewController` project uses JavaServer Faces (JSF) as the view technology, and the project relies on the ADF Model layer to interact with the EJBs in the `Model` project. To learn more about the Suppliers module and to understand its implementation details, see Section 2.3.1, "Suppliers Module Code" and Section 2.3.2, "Suppliers Module Pages."

2.2 Setting Up the Fusion Order Demo Application

The Fusion Order Demo application runs using an Oracle database and Oracle JDeveloper 11*g*. The platforms supported are the same as those supported by JDeveloper.

To set up the Fusion Order Demo application:

1. Install Oracle JDeveloper 11*g* and meet the installation prerequisites. The Fusion Order Demo application requires an existing Oracle database.
2. Install the Fusion Order Demo application from the Oracle Technology Network.
3. Install Mozilla FireFox, version 2.0 or higher, or Internet Explorer, version 7.0 or higher.
4. Run the application on a monitor that supports a screen resolution of 1024 X 768 or higher.

For more information, see "Setting Up the Fusion Order Demo Application" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

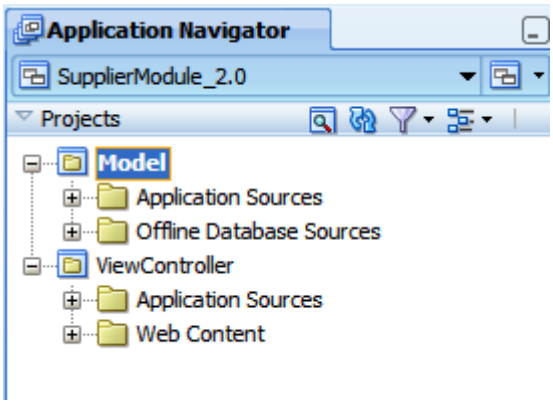
2.3 Running the Suppliers Module

To run the Suppliers module of the Fusion Order Demo application:

1. Download and install the Fusion Order Demo application as described in "Setting Up the Fusion Order Demo Application" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Open the application in Oracle JDeveloper:
   a. From the JDeveloper main menu, choose **Open** from the **File** menu.
   b. Navigate to the location to which you extracted the demo ZIP file, select the **SupplierModule_2.0.jws** application workspace from the **SupplierModule** directory, and click **Open**.

      Figure 2–1 shows the Application Navigator after you open the file for the application workspace. For a description of each of the projects in the workspace, see Section 2.3.1, "Suppliers Module Code."

Figure 2–1 The Suppliers Module Projects in Oracle JDeveloper



3. In the Application Navigator, click the **Application Resources** accordion title to expand the panel.
4. In the Application Resources panel, expand the **Connections** and **Database** nodes.
5. Right-click the **FOD** connection, and choose **Properties**.
6. In the Edit Database Connection dialog, modify the connection information shown in Table 2–1 for your environment.

Table 2–1 Connection Properties Required to Run the Fusion Order Demo Application

Property	Description
Host Name	The host name for your database. For example: `localhost`
JDBC Port	The port for your database. For example: `1521`
SID	The SID of your database. For example: `ORCL` or `XE`

   Do not modify the user name and password `fod/fusion`. These must remain unchanged. Click **OK**.
7. In the Application Navigator, right-click **Model** and choose **Rebuild**.
8. In the Application Navigator, right-click **ViewController** and choose **Run**.

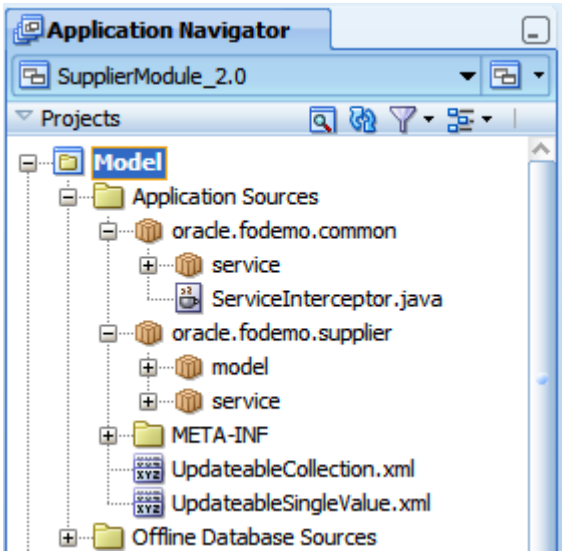
   The `login.jspx` page is displayed. Because of the way security is configured in this module, you must first log in.
9. Enter `SHEMANT` for **User Name** and `welcome1` for **Password**.

Once you log in, the browse page appears, which allows you to search for products. Once you select a product in the results table, you can edit or remove the product information. Using the command links at the top of the page, you can edit the corresponding supplier's information, or add a new supplier. For more information about the Suppliers module at runtime, see Section 2.3.2, "Suppliers Module Pages."

2.3.1 Suppliers Module Code

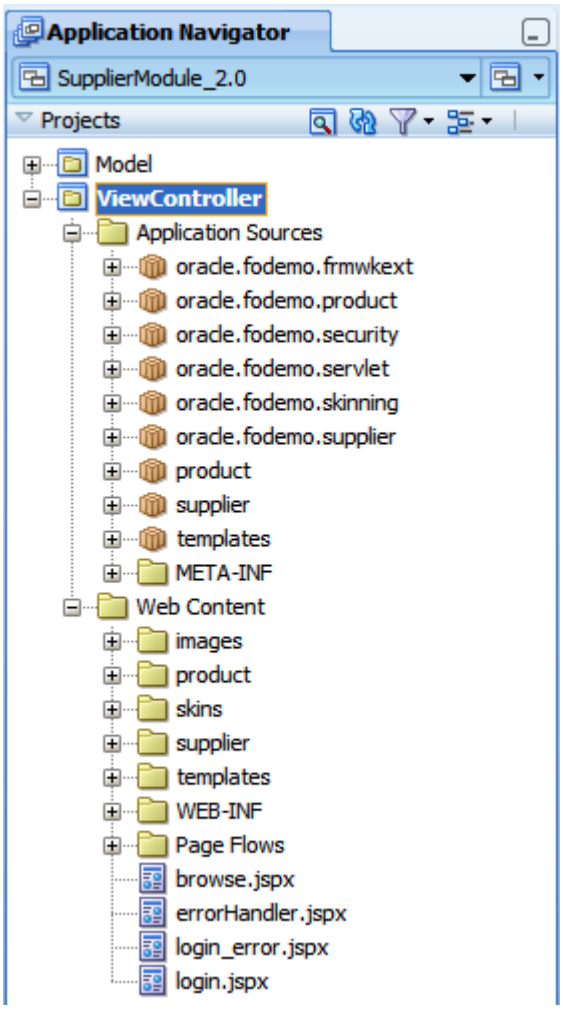
Once you have opened the projects in Oracle JDeveloper, you can then begin to review the artifacts within each project. The `Model` project contains the Java classes and metadata files that allow the data to be displayed in the web application. The `oracle.fodemo.common` project contains components used by multiple classes in the application. The `oracle.fodemo.supplier` project contains the components used to access the supplier data. Figure 2–2 shows the `Model` project and its associated directories.

Figure 2–2 The Model Project in JDeveloper



The `ViewController` project contains the files for the web interface, including the backing beans, deployment files, and JSPX files. The Application Sources node contains the code used by the web client, including the managed and backing beans, property files used for internationalization, and the metadata used by Oracle ADF to display bound data. The Web Content node contains web files, including the JSP files, images, skin files, deployment descriptors, and libraries. Figure 2–3 shows the `ViewController` project and its associated directories.

Figure 2–3 The ViewController Project in JDeveloper

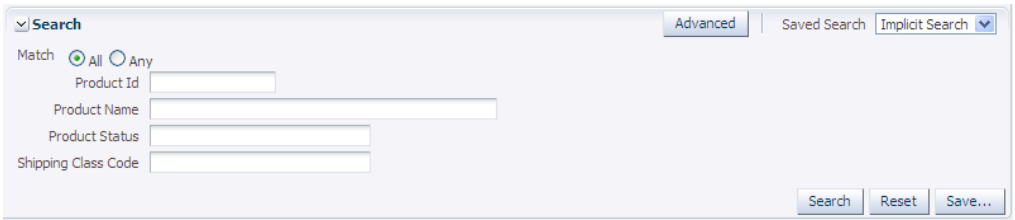


2.3.2 Suppliers Module Pages

The Supplier module contains eight main pages that enable a user to perform these tasks:

- Search for products: The `browse.jspx` page enables a user to search for products. Search results are displayed in a table. Figure 2–4 shows the search form on the `browse` page.

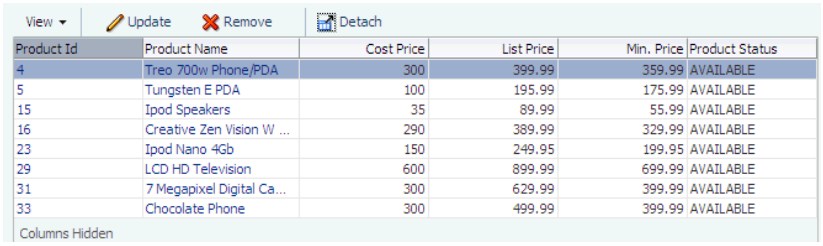
Figure 2–4 Search Form in Supplier Module



- Edit row data in a table: From the table on the `browse.jspx` page, a user can select a product and choose **Update** to navigate to the `productInfo.jspx` page (clicking the product link also navigates to this page). From the table, a user can

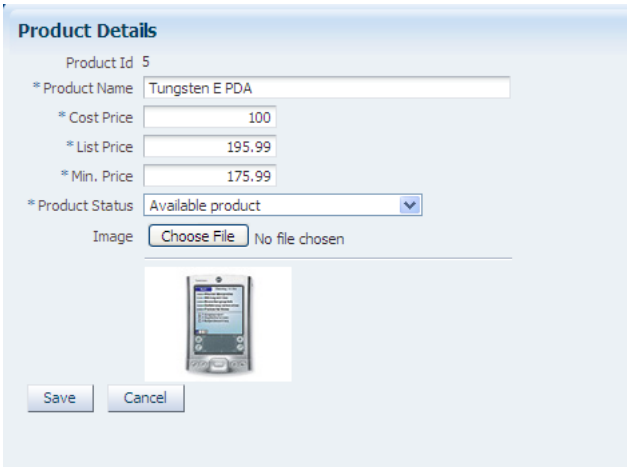
also click **Remove**, which launches a popup window that allows the removal of the selected product. Figure 2–5 shows the table on the `browse` page.

Figure 2–5 Table on the browse Page



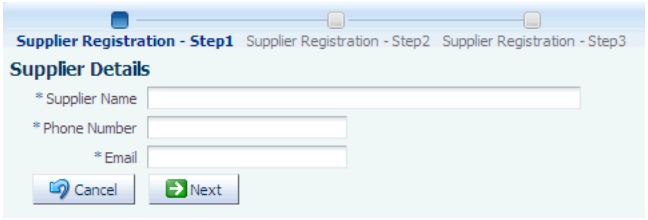
- Edit row data in a form: From the `productInfo.jspx` page, a user can change the data for a row. A selection list contains valid values for the product status. The **Choose File** button enables a user to upload a graphic file, which is then displayed below the form. Figure 2–6 shows a `productInfo.jspx` page.

Figure 2–6 Product Details Page



- The **Add Supplier** link takes the user to a series of pages contained within the `regisrationDetails.jspx` page that are used to create a new supplier, as shown in Figure 2–7.

Figure 2–7 Create a Supplier Train



- Log in to the application: The `login.jspx` page allows users to log in to the application. For more information, see "Enabling ADF Security in a Fusion Web Application" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Part II

Working with the Idoc Script Custom Scripting Language

This part describes how to use the Idoc Script Custom Scripting Language for customizing Oracle WebCenter Content.

Part II contains the following chapters:

3

Introduction to the Idoc Script Custom Scripting Language

This chapter describes the Idoc Script Custom Scripting Language, which you can use to customize Oracle WebCenter Content Server. Idoc Script is the server-side custom scripting language for the Content Server system. Idoc Script is used primarily for the presentation of HTML templates and configuration settings.

This chapter includes the following sections:

3.1 Idoc Naming Conventions

Idoc variables (sometimes called *configuration variables* or *environment variables*) can be used in Idoc Script and in configuration files.

In general, if the variable is part of a configuration, it begins with a capital letter. Those variables specified in the `config.cfg` file or `intradoc.cfg` file usually have an initial capital letter. For an example, see "DefaultFilterInputFormat" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*. Many parameters in service requests also begin with uppercase letters.

Variables defined on a page, such as those that are derived and then used in files such as `std_page.htm`, begin with a lowercase letter. For an example, see "executeService Function" in Section 6.2.2, "Idoc Script Functions." The variables are calculated from environment variables or service variables, then used for presentation.

If a variable is used to define an object, it begins with lowercase letters designating the type of object it is defining. For an example of a workflow-specific variable, see "wfSet()" in Appendix A, "Idoc Script Functions and Variables." In addition, all functions start with a lowercase letter and many start with a prefix to describe the type of function. For example, most string manipulation functions begin with `str`, or ResultSet functions begin with `rs`.

All database column names that are not custom metadata fields begin with the lowercase letter `d`. All custom metadata database column names created by the Content Server system begin with the lowercase letter `x`.

3.2 Idoc Script Syntax

Idoc Script follows these basic syntax rules:

- Section 3.2.1, "Idoc Script Tags"
- Section 3.2.2, "Idoc Script Comments"

3.2.1 Idoc Script Tags

Idoc Script commands begin with `<$` and end with `$>` delimiters. For example:

```
<$dDocTitle$>
<$if UseGuiWinLook and isTrue(UseGuiWinLook)$>
```

If you are using Idoc Script in an HCSP or HCSF page, you must use the syntax `<!--$script-->` for Idoc Script tags.

3.2.2 Idoc Script Comments

You can use standard HTML comments or Idoc Script comments in Idoc Script code. An Idoc Script comment begins with `[[%` and closes with `%]]` delimiters. For example:

```
<!-- HTML Comment -->
[[%My Comment%]]
```

An HTML comment is parsed as plain text to the Idoc Script engine. The engine only looks for Idoc Script constructs. If you want the comment to appear in the generated page, use the HTML/XML comment syntax; otherwise, Idoc Script comment syntax is recommended.

If you are writing Idoc Script that generates other Idoc Script and you want the source page to look readable, you can use the comment syntax to comment out `dynamichtml` constructs and other resource specifiers, such as string resources, in Idoc Script resource files. For example:

```
[[% Commenting out resource includes
<@dynamichtml myinclude@>
<@end@>
End comment %]]
```

3.3 Idoc Script Uses

There are six basic uses for Idoc Script:

- Includes enable you to reuse pieces of Idoc Script and HTML code.
- Variables enable you to define and substitute variable values.
- Functions enable you to perform actions, including string comparison and manipulation routines, date formatting, and ResultSet manipulation.
- Conditionals enable you to evaluate *if* and *else* clauses to include or exclude code from an assembled page.

- Looping enables you to repeat code for each row in a ResultSet that is returned from a query.
- The Administration Interface enables you to use Idoc Script in Content Server applets and customizations.

3.3.1 Includes

An *include* defines pieces of code that are used to build the Content Server web pages. Includes are defined once in a resource file and then referenced by as many template files as necessary. The system leverages includes very heavily.

Includes make it easier for you to customize your instance using component architecture and dynamic server pages. For more information on includes and customization, see Section 6.3, "Creating an IDOC File with Custom Includes for Dynamic Server Pages."

- An include is defined in an HTM resource file using the following format:

```
<@dynamichtml name@>
    code
<@end@>
```

- An include is called from an HTM template file using the following Idoc Script format:

```
<$include name$>
```

- Includes can contain Idoc Script and valid HTML code, including JavaScript, Java applets, cascading style sheets, and comments.
- Includes can be defined in the same file as they are called from, or they can be defined in a separate file.
- Standard includes are defined in the *IdcHomeDir*/resources/core/idoc/std_page.idoc file.
- HDA and CFG files are not script enabled, therefore using an include statement in either of these types of files is not supported.

The `includes` are global, available to all parts of the system. Dynamic scripting pages in HCSP files can use `includes`. The `.idoc` files can do localized `includes` that are not global. HCSP files can call both global `includes` or localized `includes` with the proper syntax.

An `include` can override an existing `include`, if one of the same name exists.

For more information, see the following sections:

- Section 3.3.1.1, "Include Example"
- Section 3.3.1.2, "Super Tag"
- Section 3.3.1.3, "Super Tag Example"

3.3.1.1 Include Example

One of the most common includes is the body definition element `<@dynamichtml body_def>`. This include sets the page background color, the color of hyperlinks, and the background image. The following is an example of code located in the *IdcHomeDir*/resources/core/idoc/std_page.idoc file:

```
<@dynamichtml body_def@>
<!--Background image defined as part of body tag--->
<body
    <$if background_image$>
        background="<$HttpImagesRoot$><$background_image$>"
    <$elseif colorBackground$>
        bgcolor="<$colorBackground$>"
    <$endif$>
    <$if xpedioLook$>
        link="#663399" vlink="#CC9900"
    <$else$>
        link="#000000" vlink="#CE9A63" alink="#9C3000"
    <$endif$>
    marginwidth="0" marginheight="0" topmargin="0" leftmargin="0"
>
<@end@>
```

Most of the standard template resource files (for example, *IdcHomeDir*/resources/core/templates/pne_home_page.htm) contain the following Idoc Script code near the top of the page:

```
<$include body_def$>
```

When the Content Server system resolves a template page containing this code, it looks for the <@dynamichtml body_def@> definition and replaces the placeholder code with the code in the definition.

3.3.1.2 Super Tag

The super tag is used to define exceptions to an existing include. The super tag tells the include to start with an existing include and then add to it or modify it using the specified code.

- The super tag uses the following syntax:

  ```
  <@dynamichtml my_resource@>
      <$include super.my_resource$>
      exception code
  <@end@>
  ```

- You can use the super tag to refer to a standard include or a custom include. The super tag incorporates the include that was loaded last.
- The resource name defined after super should match the include name in which you include super. In the preceding syntax example, my_resource is the include name, so the matching call should be super.my_resource.
- You can specify multiple super tags to call an include that was loaded earlier than the last version. For example, to make an exception to the standard body_def include in two different components, you can use the following syntax in the one that is loaded last:

  ```
  <$include super.super.body_def$>
  ```

  Caution: If you use multiple super tags in one include, ensure that you know where the resources are loaded from and the order they are loaded in.

- The `super` tag is particularly useful when making small customizations to large includes or when you customize standard code that is likely to change from one software version to the next. When you upgrade to a new version of Content Server software, the `super` tag ensures that your components are using the most recent version of the include, modifying only the specific code you require to customize your instance.

3.3.1.3 Super Tag Example

In this example, a component defines the `my_resource` include as follows:

```
<@dynamichtml my_resource@>
    <$a = 1, b = 2$>
<@end@>
```

Another component that is loaded later enhances the `my_resource` include using the `super` tag. The result of the following enhancement is that `a` is assigned the value 1 and `b` is assigned the value 3:

```
<@dynamichtml my_resource@>
    <$include super.my_resource$>
    <!--Change "b" but not "a" -->
    <$b = 3$>
<@end@>
```

3.3.2 Variables

A *variable* enables you to define and substitute variable values.

The following sections describe how to work with Idoc Script variables:

- Section 3.3.2.1, "Variable Creation"
- Section 3.3.2.2, "Variable References"
- Section 3.3.2.3, "Variable Values"
- Section 3.3.2.4, "Comma Separators"
- Section 3.3.2.5, "Variable Reference in a Conditional"
- Section 3.3.2.6, "Variable Reference Search Order"
- Section 3.3.2.7, "Regular Variables"

3.3.2.1 Variable Creation

Idoc Script variables are created in one of the following ways:

- Many variables are predefined.
- You can define your own custom variables.
- Some variable values must be generated using queries and services. Some variable information is not automatically available from the database, so it must be asked for by defining a query and service.

For more information on the types of Idoc Script variables, see Chapter 4, "Using Idoc Script Variables and Functions with Oracle WebCenter Content."

3.3.2.2 Variable References

You can reference a variable in templates and other resource files with the following Idoc Script tag:

```
<$variable_name$>
```

When you reference a variable name like this, the generated page will replace the Idoc Script tag with the value of that variable, at the time it was referenced.

3.3.2.3 Variable Values

- A value can be assigned to a variable using this structure:

  ```
  <$variable=value$>
  ```

  For example, `<$i=0$>` assigns the value of 0 to the variable *i*.

- Variable values can also be defined in an environment resource (CFG) file using the following name/value pair format:

  ```
  variable=value
  ```

  For example, standard configuration variables are defined in the *IntradocDir*`/config/config.cfg file`.

  Note: Not all configuration variable values set by code are listed in the *IntradocDir*`/config/config.cfg` file by default.

3.3.2.4 Comma Separators

Idoc Script supports multiple clauses separated by commas in one script block in variable references.

For example, you can use `<$a=1,b=2$>` rather than two separate statements: `<$a=1$>` and `<$b=2$>`.

3.3.2.5 Variable Reference in a Conditional

The following structure can be used to evaluate the existence of a variable:

```
<$if variable_name$>
```

If the variable is defined and not empty, this conditional is evaluated as TRUE. If the variable is not defined or it is defined as an empty (null) string, it is evaluated as FALSE.

For an example of how this type of reference is typically used, see Section 3.3.4.1, "Conditional Example."

3.3.2.6 Variable Reference Search Order

When a variable is referenced to fulfill a service request, the substituted value will be the first match found in the DataBinder from the following default order:

1. `LocalData`
2. `Active ResultSets`
3. `Nonactive ResultSets`
4. `Environment`

For example, if a particular variable exists in the environment but is also the name of a field in the active ResultSet, the value in the current row of the active ResultSet will be used.

3.3.2.7 Regular Variables

A regular variable that does not have special evaluation logic (such as Conditional Dynamic Variables) is equivalent to using the `#active` keyword prefix.

For example, the tag `<$variable$>` is equivalent to `<$#active.variable$>`. However, if `#active` is not explicitly stated and the variable is not found, an error report is printed to the debug output.

The `#active` qualifier means that a variable reference searches the DataBinder, as described in Section 3.3.2.6, "Variable Reference Search Order," whereas `#env` lets you select only from the environment, and `#local` always references `LocalData`. The difference between explicitly using `#active` versus without the prefix is that an error is reported only when (1) you are not using any qualifier and (2) the variable reference is not found in the DataBinder.

3.3.3 Functions

Idoc Script has many built-in global functions. Functions perform actions, including string comparison and manipulation routines, date formatting, and ResultSet manipulation. Some functions also return results, such as the results of calculations or comparisons.

Information is passed to functions by enclosing the information in parentheses after the name of the function. Pieces of information that are passed to a function are called *parameters*. Some functions do not take parameters; some functions take one parameter; some take several. There are also functions for which the number of parameters depends on how the function is being used.

For a list of Idoc Script functions, see Section 4.1.4, "Global Functions."

3.3.3.1 Personalization Functions

Personalization functions refer to user properties that are defined in personalization files, also called *user topic files*. Each user's User Profile settings, personal links in the left navigation bar, and workflow in queue information are all defined in user topic files, which are HDA files located in the *WC_CONTENT_ORACLE_HOME*`/data/users/profiles/`*us*`/`*username*`/` directories.

The following global functions reference user topic files:

- utGetValue()
- utLoad()
- utLoadResultSet()

For example, the Portal Design link in a user's left navigation bar is generated from the following code in the `pne_nav_userprofile_links` include (located in the *WC_CONTENT_ORACLE_HOME*`/shared/config/resources/std_page.htm` resource file). If the `portalDesignLink` property in the *WC_CONTENT_ORACLE_HOME*`/data/users/profiles/`*us*`/`*username*`/pne_portal.hda` file is `TRUE`, the link is displayed:

```
<$if utGetValue("pne_portal", "portalDesignLink") == 1$>
    <$hasUserProfileLinks=1$>
    <tr>
    <td colspan=2 nowrap align="left">
        <a class=pneLink href="<$HttpCgiPath$>?IdcService=GET_PORTAL_
PAGE&Action=GetTemplatePage&Page=PNE_PORTAL_DESIGN_PAGE">
        <$lc("wwPortalDesign")$></a>
    <td>
```

```
    </tr>
<$endif$>
```

3.3.4 Conditionals

A *conditional* enables you to use *if* and *else* clauses to include or exclude code from an assembled page.

- Use the following Idoc Script keywords to evaluate conditions:
  - `<$if condition$>`
  - `<$else$>`
  - `<$elseif condition$>`
  - `<$endif$>`
- Conditional clauses use this general structure:

```
<$if conditionA$>
    <!--Code if conditionA is true-->
<$elseif conditionB$>
    <!--Code if conditionB is true-->
<$else$>
    <!--Code if neither conditionA nor conditionB is true-->
<$endif$>
```

- A `condition` expression can be any Idoc Script function or variable.

  For more information, see Section 3.3.2.5, "Variable Reference in a Conditional."
- Boolean Operators can be used to combine conditional clauses. For example, you can use the `and` operator as follows:

```
<$if UseBellevueLook and isTrue(UseBellevueLook)$>
```

  The first expression tests whether the variable exists and is not empty, and the second expression checks to see if the value of that variable evaluates to `1` or if it starts with `t` or `y` (case-insensitive). If you just have the second clause, it may generate an error if the variable is not set, or empty. An equivalent expression follows:

```
<$if isTrue(#active.UseBellevueLook)$>
```

- If the `condition` expression is the name of a ResultSet available for inclusion in the HTML page, the conditional clause returns `true` if the ResultSet has at least one row. This ensures that a template page presents information for a ResultSet only if there are rows in the ResultSet.
- A conditional clause that does not trigger special computation is evaluated using the `#active` prefix. The result is `true` if the value is not null and is either a nonempty string or a nonzero integer.

For an example of conditional code, see Section 3.3.4.1, "Conditional Example."

3.3.4.1 Conditional Example

In this example, a table cell `<td>` is defined depending on the value of the variable `xDepartment`:

```
<$if xDepartment$>
    <td><$xDepartment$></td>
<$else$>
```

```
    <td>Department is not defined.</td>
<$endif$>
<$xDepartment=""$>
```

- If the value of `xDepartment` is defined, then the table cell contains the value of `xDepartment`.
- If the value of `xDepartment` is not defined or is an empty (null) string, a message is written as the content of the table cell.
- The last line of code clears the `xDepartment` variable by resetting it to an empty string.

3.3.5 Looping

Loop structures allow you to execute the same code a variable number of times. Looping can be accomplished in two ways with Idoc Script:

- Section 3.3.5.1, "ResultSet Looping"
- Section 3.3.5.3, "While Looping"

For information on exiting and ending a loop structure, see Section 3.3.5.5, "Ending a Loop."

3.3.5.1 ResultSet Looping

ResultSet looping repeats a set of code for each row in a ResultSet that is returned from a query. The name of the ResultSet to be looped is specified as a variable using the following syntax:

```
<$loop ResultSet_name$>
    code
<$endloop$>
```

- The code between the `<$loop$>` and `<$endloop$>` tags is repeated once for each row in the ResultSet.
- When inside a ResultSet loop, you can retrieve values from the ResultSet using the getValue() function. Substitution of values depends on which row is currently being accessed in the loop.
- When inside a ResultSet loop, that ResultSet becomes active and has priority over other ResultSets when evaluating variables and conditional statements.
- Infinite loops with ResultSet loops are not possible (finite lists), whereas `while` loops can cause infinite loops.
- You cannot use the `<$loop$>` tag to loop over a variable that points to a ResultSet. Instead you must loop over the ResultSet manually using the rsFirst() and rsNext() functions.

  For example, you cannot use the following code to loop over a ResultSet:

  ```
  <$name="SearchResults"$>
  <$loop name$>
      <!--output code-->
  <$endloop$>
  ```

  Instead, you must use the following code:

  ```
  <$name="SearchResults"$>
  <$rsFirst(name)$>
  <$loopwhile getValue(name, "#isRowPresent")$>
  ```

```
        <!--output code-->
        <$rsNext(name)$>
    <$endloop$>
```

3.3.5.2 ResultSet Looping Example

In this example, a search results table is created by looping over the `SearchResults` ResultSet, which was generated by the `GET_SEARCH_RESULTS` service.

```
<$QueryText="dDocType <matches> 'ADACCT'"$>
<$executeService("GET_SEARCH_RESULTS")$>
<table>
    <tr>
        <td>Title</td><td>Author</td>
    </tr>
<$loop SearchResults$>
    <tr>
        <td><a href="<$SearchResults.URL$>"><$SearchResults.dDocTitle$></a></td>
        <td><$SearchResults.dDocAuthor$></td>
    </tr>
<$endloop$>
</table>
```

3.3.5.3 While Looping

While looping enables you to create a conditional loop. The syntax for a while loop is:

```
<$loopwhile condition$>
    code
<$endloop$>
```

- If the result of the *`condition`* expression is `true`, the code between the `<$loopwhile$>` and `<$endloop$>` tags is executed.
- After all of the code in the loop has been executed, control returns to the top of the loop, where the *`condition`* expression is evaluated again.
  - If the result is `true`, the code is executed again.
  - If the code if the result is `false`, the loop is exited.

3.3.5.4 While Looping Example

In this example, a variable named `abc` is increased by 2 during each pass through the loop. On the sixth pass (when `abc` equals 10), the condition expression is no longer true, so the loop is exited.

```
<$abc=0$>
<$loopwhile abc<10$>
    <$abc=(abc+2)$>
<$endloop$>
```

3.3.5.5 Ending a Loop

There are two Idoc Script tags that will terminate a ResultSet loop or while loop:

- `<$endloop$>` returns control to the beginning of the loop for the next pass. All loops must be closed with an `<$endloop$>` tag.
- `<$break$>` causes the innermost loop to be exited. Control resumes with the first statement following the end of the loop.

3.3.6 Administration Interface

You can use Idoc Script in several areas of the administration interface, including:

- Section 3.3.6.1, "Workflow Admin"
- Section 3.3.6.2, "Web Layout Editor"
- Section 3.3.6.3, "Batch Loader"
- Section 3.3.6.4, "Archiver"
- Section 3.3.6.5, "System Properties"
- Section 3.3.6.6, "Email"

3.3.6.1 Workflow Admin

In the Workflow Admin tool, you can use Idoc Script to define the following:

- step events
- jump messages
- extra exit conditions
- tokens
- custom effects for jumps

For example, the following step entry script sends documents in the `Secure` security group to the next step in the workflow:

```
<$if dSecurityGroup like "Secure"$>
    <$wfSet("wfJumpName", "New")$>
    <$wfSet("wfJumpTargetStep", wfCurrentStep(1))$>
    <$wfSet("wfJumpEntryNotifyOff", "0")$>
<$endif$>
```

For more information, see Section 4.1.8, "Workflows."

3.3.6.2 Web Layout Editor

In the Web Layout Editor, you can use Idoc Script in the page titles, page descriptions, URL descriptions, query result pages, and content queries. For example:

- You can use Idoc Script tags in the query results page definition to specify the contents of each row in a search results table.
- To set the search results to return all content items up to 7 days, you could define the search query to be:

  ```
  dInDate > '<$dateCurrent(-7)$>'
  ```

- To define a report that returns results based on the current user, you could define `User Name is <$UserName$>` as part of the report query expression.

For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

3.3.6.3 Batch Loader

In the Batch Loader, you can use Idoc Script in a mapping file, which tells the BatchBuilder utility how to determine the metadata for file records. For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

3.3.6.4 Archiver

In Archiver, you can use Idoc Script in the following areas:

- **Export query values.** For example, to archive content more than one year old, you could use `<$dateCurrent(-365)$>` as the Release Date value.
- **Value map output values.** For example, to set the expiration date one week in the future for all imported revisions, you could use `<dateCurrent(7)$>` as the Output Value.

For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

3.3.6.5 System Properties

When you set values in the System Properties utility, in the Admin Server, or in the Local Configuration or Shared Configuration screens of the Oracle WebCenter Content: Inbound Refinery instance, you are actually setting Idoc Script configuration variables. For more information, see the documentation for these applications.

3.3.6.6 Email

You can use Idoc Script to customize the subject line and body of a notification email. For example:

```
Hello, <$UserName$>. Content item <$dDocName$> requires your review.
```

3.4 Special Keywords

The following keywords have special meaning in Idoc Script.

Keyword	Example	Description
#active	<$#active.*variable*$>	Retrieves the value of the specified variable from the DataBinder, searching in the following default order: 1. Active ResultSets 2. Local data 3. All other ResultSets 4. Environment Does not send an error report to the debug output if the variable is not found.
#local	<$#local.*variable*$>	Retrieves the value of the specified variable from the local data. Does not send an error report to the debug output if the variable is not found.
#env	<$#env.*variable*$>	Retrieves the value of the specified variable from the environment settings. Does not send an error report to the debug output if the variable is not found.
exec	<$exec *expression*$>	Executes an expression and suppresses the output (does not display the expression on the page). In earlier versions of Idoc Script, the *exec* keyword was required to suppress the *value* of any variable from appearing in the output file. In the current version, the *exec* keyword is needed only to suppress an *expression* from appearing in the output file.
include	<$include *ResourceName*$>	Includes the code from the specified resource. For more information, see Section 3.3.1, "Includes."
super	<$include super.<*include*>$>	Starts with the existing version of the include code. For more information, see Section 3.3.1.2, "Super Tag."

3.4.1 Keywords Versus Functions

Content Server pages use the `include` and `exec` special keywords and the `inc` and `eval` functions extensively. This section describes the differences between these commands and gives examples of how to use them.

The `include` and `exec` keywords are standalone commands that operate on defined parameters, but cannot take a variable as a parameter. The `inc` and `eval` functions have similar purposes, but they can take variables for parameters, which enables you to dynamically create Idoc Script code depending on the value of the variables.

The following sections describe these keywords and functions in detail:

- Section 3.4.1.1, "exec Keyword"
- Section 3.4.1.2, "eval Function"
- Section 3.4.1.3, "include Keyword"
- Section 3.4.1.4, "inc Function"

3.4.1.1 exec Keyword

The `exec` keyword executes an Idoc Script expression and suppresses the output (does not display the expression on the page). It is primarily used to set variables without writing anything to the page.

In earlier versions of Idoc Script, the exec keyword was required to suppress the value of any variable from appearing in the output file. In the current version, the exec keyword is needed only to suppress an expression from appearing in the output.

For example, if you used the following expression, the output value, 0 or 1, would appear in the output file:

```
<$rsFirst(name)$>
```

Instead, you can use the exec keyword before the expression to suppress the output:

```
<$exec rsFirst(name)$>
```

You can also use exec to suppress the output from rsNext:

```
<$exec rsNext(name)$>
```

For more information, see exec in Appendix A, "Idoc Script Functions and Variables."

3.4.1.2 eval Function

The eval function evaluates an expression as if it were actual Idoc Script.

In the following example, a variable named one is assigned the string Company Name, and a variable named two is assigned a string that includes variable one.

```
<$one="Company Name"$>
<$two="Welcome to <$one$>"$>
<$one$><br>
<$two$><br>
<$eval(two)$>
```

In the page output, variable one presents the string Company Name, variable two presents the string Welcome to <one>, and the function eval(two) presents the string Welcome to Company Name.

Note that the string to be evaluated must have the Idoc Script delimiters <$ $> around it, otherwise it will not be evaluated as Idoc Script.

Also note that too much content generated dynamically in this manner can slow down page display. If the eval function is used frequently on a page, it may be more efficient to put the code in an include and use the inc function in conjunction with the eval function.

For more information, see eval() in Appendix A, "Idoc Script Functions and Variables.".

3.4.1.3 include Keyword

The include keyword is the standard way in which chunks of code are incorporated into the current page. Because include is a keyword, it cannot take a variable as a parameter-the parameter must be the name of an include that already exists.

For more information, see Section 3.3.1, "Includes," and include in Appendix A, "Idoc Script Functions and Variables."

3.4.1.4 inc Function

The inc function does the same thing as the include keyword, except that it can take a variable as the parameter. This function is most useful for dynamically changing which include will be used depending on the current value of a variable.

For example, say you want to execute some Idoc Script for some, but not all, of your custom metadata fields. You could dynamically create includes based on the field names (such as `specific_include_xComments`) by executing this Idoc Script:

```
<$loop DocMetaDefinition$>
    <$myInclude = "specific_include_" & dName$>
    <$exec inc(myInclude)$>
<$endloop$>
```

Note the use of the exec Keyword, which suppresses the output of the include specified by the *inc* function. If you do not use `exec` before the `inc` function, the HTML inside the specified include will be displayed on the page.

Note that if the `specific_include_xComments` does not exist, this code will not throw an error because the output is not being displayed.

For more information, see inc() in Appendix A, "Idoc Script Functions and Variables."

3.5 Operators

Idoc Script supports several operators.

- Section 3.5.1, "Comparison Operators"
- Section 3.5.2, "Special String Operators"
- Section 3.5.3, "Numeric Operators"
- Section 3.5.4, "Boolean Operators"

3.5.1 Comparison Operators

Use the following comparison operators compare the value of two operands and return a *true* or *false* value based on the result of the comparison. These operators can be used to compare integers and Boolean values in Idoc Script.

If you are using Idoc Script in an HCSP or HCSF page, you must use special comparison operators. For more information, see Section 6.2.1.2, "Comparison Operators."

Operator	Description	Example
==	equality	`<$if 2 == 3$>` evaluates to false
!=	inequality	`<$if 2 != 3$>` evaluates to true
<	less than	`<$if 2 < 2$>` evaluates to false
<=	less than or equal	`<$if 2 <= 2$>` evaluates to true
>	greater than	`<$if 3 > 2$>` evaluates to true
>=	greater than or equal	`<$if 3 >= 2$>` evaluates to true

These are numeric operators that are useful with strings only in special cases where the string data has some valid numeric meaning, such as dates (which convert to milliseconds when used with the standard comparison operators).

- For string concatenation, string inclusion, and simple string comparison, use the Special String Operators.
- To perform advanced string operations, use strEquals(), strReplace(), or other string-related global functions.

3.5.2 Special String Operators

Use the following special string operators to concatenate and compare strings:

Operator	Description	Example

&	The *string join operator* performs string concatenation. Use this operator to create script that produces Idoc Script for a resource include.	`<$"<$include " & VariableInclude & "$>"$>`
		evaluates to:
		`<$include VariableName$>`

like	The *string comparison operator* compares two strings.
	■ The first string is compared against the pattern of the second string. (The second string can use asterisk and question mark characters as wildcards.)
	■ This operator is not case sensitive.
		■ Evaluates to FALSE:
		`<$if "cart" like "car"$>`
		■ Evaluates to TRUE:
		`<$if "cart" like "car?"$>`
		■ Evaluates to TRUE:
		`<$if "carton" like "car*"$>`
		■ Evaluates to TRUE:
		`<$if "Carton" like "car*"$>`

|	The *string inclusion operator* separates multiple options, performing a logical OR function.	Evaluates to TRUE: `<$if "car" like "car\|truck\|van"$>`

For example, to determine whether the variable `a` has the prefix `car` or contains the substring `truck`, this expression could be used:

```
<$if a like "car*|*truck*"$>
```

Important: To perform advanced string operations, use strEquals(), strReplace(), or other string-related global functions.

The `like` operator recognizes the following wildcard symbols:

Wildcard	Description	Example
*	Matches 0 or more characters.	■ `grow*` matches `grow`, `grows`, `growth`, and `growing` ■ `*car` matches `car`, `scar`, and `motorcar` ■ `s*o` matches `so`, `solo`, and `soprano`
?	Matches exactly one character.	■ `grow?` matches `grows` and `growl` but not `growth` ■ `grow??` matches `growth` but not `grows` or `growing` ■ `b?d` matches `bad`, `bed`, `bid`, and `bud`

3.5.3 Numeric Operators

Use the following numeric operators to perform arithmetic operations. These operators are for use on integers evaluating to integers or on floats evaluating to floats.

Operator	Description	Example
+	Addition operator.	`<$a=(b+2)$>`
-	Subtraction operator.	`<$a=(b-2)$>`
*	Multiplication operator.	`<$a=(b*2)$>`
/	Division operator.	`<$a=(b/2)$>`
%	Modulus operator. Provides the remainder of two values divided into each other.	`<$a=(b%2)$>`

3.5.4 Boolean Operators

Use the following Boolean operators to perform logical evaluations.

Operator	Description	Example
and	■ If both operands have nonzero values or are true, the result is 1. ■ If either operand equals 0 or is false, the result is 0.	`<$if 3>2 and 4>3$>` evaluates to 1
or	■ If either operand has a nonzero value or is true, the result is 1. ■ If both operands equal 0 or are false, the result is 0.	`<$if 3>2 or 3>4$>` evaluates to 1
not	■ If the operand equals 0 or is false, the result is 1. ■ If the operand has a nonzero value or is true, the result is 0.	`<$if not 3=4$>` evaluates to 1

Boolean operators evaluate from left to right. If the value of the first operand is sufficient to determine the result of the operation, the second operand is not evaluated.

3.6 Metadata Fields

This section includes these topics:

3.6.1 Metadata Field Naming

Each metadata field has an internal field name, which is used in code. In addition, many fields have descriptive captions which are shown on web pages.

- Use field captions when displaying metadata to the user.
- Use internal field names when batch loading files or scripting dynamic server pages (.hcst,.hcsp, and .hcsf pages).

- All internal metadata field names begin with either a `d` or an `x`:
  - *Predefined* or core field names begin with a `d`. For example, `dDocAuthor`.
  - *Custom* or add-on field names begin with an `x`. For example, `xDepartment`.

  Note: Add-on components from Oracle and custom components made by customers all start with `x`.

- When you create a custom metadata field in the Configuration Manager, the `x` is automatically added to the beginning of your field name.

  Important: In all cases, internal metadata field names are case sensitive.

3.6.2 Standard Metadata Fields

This section describes the standard metadata fields that the Content Server system stores for each content item. The fields are grouped as follows:

- Section 3.6.2.1, "Common Metadata Fields"
- Section 3.6.2.2, "Other Fields"

3.6.2.1 Common Metadata Fields

The following metadata fields are the most commonly used in customizing the interface. These fields appear by default on checkin and search pages.

Note: Add-on components, of which there are many, are not listed here. For example, the `FrameworkFolders` component is enabled in many configurations of Content Server, but `FrameworkFolders` fields are not in the list.

Do not confuse the Content ID (dDocName) with the dID. The dID is an internally generated integer that refers to a specific revision of a content item.

Internal Field Name	Standard Field Caption	Description
dDocAccount	Account	Security account.
dDocAuthor	Author	User who checked in the revision.
xComments	Comments	Explanatory comments.
dDocName	Content ID	Unique content item identifier.
dOutDate	Expiration Date	Date the revision becomes unavailable for searching or viewing.
dInDate	Release Date	Date the revision is scheduled to become available for searching and viewing (see also dCreateDate and dReleaseDate).
dRevLabel	Revision	Revision label (see also dRevisionID).
dSecurityGroup	Security Group	Security group.
dDocTitle	Title	Descriptive title.
dDocType	Type	Content type.

3.6.2.2 Other Fields

In addition to the Common Metadata Fields, the following metadata is stored for content items:

Internal Field Name	Standard Field Caption	Description
dCheckoutUser	Checked Out By (Content Information page)	User who checked out the revision.
dCreateDate	None	Date the revision was checked in.
dDocFormats	Formats (Content Information page)	File formats of the primary and alternate files.
dDocID	None	Unique rendition identifier.
dExtension	None	File extension of the primary file.
dFileSize	None	File size of the primary file (in kilobytes).
dFlag1	None	Not used.
dFormat	Format (checkin page, *Allow override format on checkin* enabled)	File format of the primary file.
dID	None	Unique revision identifier.

Internal Field Name	Standard Field Caption	Description
dIndexerState	None	State of the revision in an Indexer cycle. Possible values are: **X**: The revision has been processed by the rebuild cycle. **Y**: The revision has been processed by the rebuild cycle. **A, B, C, or D**: Values generated at run time that can be assigned to any of the following states: ■ Loading the revision for the active update cycle. ■ Indexing the revision for the active update cycle. ■ Loading the revision for the rebuild cycle. ■ Indexing the revision for the rebuild cycle. The specific definitions of these values are stored in the file *DomainHome*/ucm/cs/search/cyclesymbols.hda.
dIsCheckedOut	None	Indicates whether the revision is checked out. **0**: Not checked out **1**: Checked out
dIsPrimary	None	Indicates the type of file, primary or alternate. **0**: Alternate file **1**: Primary file
dIsWebFormat	None	Indicates whether the file is the web-viewable file in the weblayout directory. **0**: Not web-viewable file **1**: Web-viewable file
dLocation	None	Not used.
dMessage	None (Content Information page)	Indicates the success or reason for failure for indexing or conversion.
dOriginalName	Get Native File (Content Information page) Original File (revision checkin page)	Original file name of the native file.

Internal Field Name	Standard Field Caption	Description
dProcessingState	None	Conversion status of the revision. Possible values are: **I**: Incomplete Conversion; an error occurred in the conversion after a valid web-viewable file was produced and the file was full-text indexed. **Y**: Converted; the revision was converted successfully and the web-viewable file is available. **P**: Refinery PassThru; the Inbound Refinery system failed to convert the revision and passed the native file through to the web. **F**: Failed; the revision is deleted, locked, or corrupted, or an indexing error occurred. **C**: Processing; the revision is being converted by Inbound Refinery. **M**: MetaData Only; full-text indexing was bypassed and only the revision's metadata was indexed.
dReleaseDate	None	Date that the revision was actually released.
dReleaseState	None	Release status of a revision. **N**: New, not yet indexed **E**: In a workflow **R**: Processing, preparing for indexing **I**: Currently being indexed; the file has been renamed to the released name **Y**: Released **U**: Released, but needs to be updated in the index **O**: Old revision
dRendition1	None	Indicates whether the file is a thumbnail rendition. Possible values are: **null**: File is not a thumbnail rendition **T**: File is a thumbnail rendition
dRendition2	None	Not used.
dRevClassID	None	Internal integer that corresponds to the Content ID (dDocName). Used to enhance query response times.
dRevisionID	None	Internal revision number that increments by 1 for each revision of a content item, regardless of the value of dRevLabel.

Internal Field Name	Standard Field Caption	Description
dStatus	Status (Content Information page)	State of a revision in the system. Possible values are: **GENWWW**: The file is being converted to web-viewable format or is being indexed, or has failed conversion or indexing. **DONE**: The file is waiting to be released on its specified Release Date. **RELEASED**: The revision is available. **REVIEW**: The revision is in a workflow and is being reviewed. **EDIT**: The revision is at the initial contribution step of a workflow. **PENDING**: The revision is in a Basic workflow and is waiting for approval of all revisions in the workflow. **EXPIRED**: The revision is no longer available for viewing. The revision was not deleted, but it can be accessed only by an administrator. **DELETED**: The revision has been deleted and is waiting to be completely removed during the next indexing cycle.
dWebExtension	None	File extension of the web-viewable file.

3.6.3 Option Lists

An option list is a set of values that can be selected for a metadata field. Option lists can be formed from queries (dynamically built from the DB), or they can be hard coded and stored in Content Server files (HDA) on the file system.

The following topics describe the use of option lists:

- Section 3.6.3.1, "Internal Option Lists"
- Section 3.6.3.2, "Option List Script"
- Section 3.6.3.3, "Methods for Creating an Option List"

3.6.3.1 Internal Option Lists

The Content Server system maintains the following internal option lists by default:

Metadata Field	Option List
Author (dDocAuthor)	docAuthors
Security Group (dSecurityGroup)	securityGroups
Type (dDocType)	docTypes
Account (dDocAccount)	docAccounts
Role (dRole)	roles

The *securityGroups* and *docAccounts* option lists are filtered according to the current user's permissions.

3.6.3.2 Option List Script

The following Idoc Script variables and functions are used to generate and enable option lists:

Variable or Function	Description
optList() function	Generates the option list for a metadata field.
optionListName variable	Specifies the name of an option list.
fieldIsOptionList variable	Specifies that a metadata field has an option list.
fieldOptionListType variable	Specifies the type of option list (strict, combo, multi, or access).
hasOptionList variable	Set to the value of the fieldIsOptionList variable. This variable is used in conditional statements.
defaultOptionListScript variable	Defines a piece of Idoc Script that displays a standard option list field.
optionListScript variable	Overrides the standard implementation of option list fields (as defined by the defaultOptionListScript variable).
optionsAllowPreselect variable	Specifies that a metadata field option list can be prefilled with its last value.
addEmptyOption variable	Specifies that the first value in the option list is blank.
optionListResultSet variable	Specifies a ResultSet that contains option list values.
optionListKey variable	Specifies the name of a ResultSet column that contains option list values.
optionListValueInclude variable	Specifies an include that defines the values for an option list.

3.6.3.3 Methods for Creating an Option List

To create an option list, you can use one of the following methods:

- Use the optList() function to generate a basic option list. This function produces output only when used with a service that calls `loadMetaOptionsList`.

  For example, this code displays a list of possible authors as an HTML option list:

  ```
  <select name="dDocAuthors">
      <$optList docAuthors$>
  </select>
  ```

- Use the rsMakeFromList() function to turn the option list into a ResultSet, and then loop over the ResultSet.

  For example, this code creates a ResultSet called `Authors` from the `docAuthors` option list, and loops over the ResultSet to create an HTML option list. (Because the column name is not specified as a parameter for `rsMakeFromList`, the column name defaults to `row`.)

  ```
  <$rsMakeFromList("Authors","docAuthors")$>
  <select name="dDocAuthors">
      <$loop Authors$>
          <option><$row$>
  <$endloop$>
  </select>
  ```

This code sample is equivalent to the sample produced by using the `optList` function. Typically, you would use the `rsMakeFromList` function when you want to parse or evaluate the list options.

- Use the Configuration Manager applet to create an option list, without writing any code.

3.6.4 Metadata References in Dynamic Server Pages

For dynamic server pages, several metadata values are stored with a `ref:` prefix, which makes them available to the page but does not replace ResultSet values. (This prevents pollution of ResultSets by dynamic server pages.)

When you reference any of the following metadata values on a dynamic server page, you must include the `ref:` prefix:

- `hasDocInfo`
- `dDocName`
- `dExtension`
- `dSecurityGroup`
- `isLatestRevision`
- `dDocType`
- `dID`

For example, the following statement determines if the document type is `Page`:

```
<$if strEquals(ref:dDocType,"Page"))$>
```

For more information, see Section 6.2, "Altering the Appearance and Navigation of Web Pages".

3.7 Merge Includes for Formatting Results

You can use a MergeInclude to format your results from an Content Server request based on an Idoc Script include, rather than an entire template page.

A MergeInclude is a feature often used to integrate ASP pages using the IdcCommandX ActiveX module. The Content Server architecture is essentially a modular, secure, service-based application with multiple interfaces, although its architecture was designed to optimize the web interface. Services such as `GET_SEARCH_RESULTS` will generate response data based on the `QueryString` passed, and the user's security credentials. This response data is internally represented in the form of a HDA file. To see this in action, simply perform a search and then add `'IsJava=1'` or `'IsSoap=1'` (for XML-formatted data) to the URL. You can now see how data is internally represented for the response.

Because this HDA representation is not particularly useful for web-based users, we use Idoc Script includes and templates to format the response into a readable HTML page. A user can modify how this HTML is displayed by changing the template or a few resource includes with a component.

However, to retrieve only a small portion of this search result (for example, to display it on an ASP, JSP, or PHP page where the majority of the code is not Idoc Script), or

have an IFRAME or DIV element pop up and display the results, or to dynamically change how to display the results, you can simply add these parameters to your URL:

```
MergeInclude=my_custom_include&IsJava=1
```

This will cause the Content Server system to bypass formatting the response according to the template that is specified in the service. It will instead format the response based on the Idoc Script in `my_custom_include`. For example, if you executed a search, then added the above line to the URL, and the include looked like this in your component:

```
<@dynamichtml my_custom_include@>
<html>
<table width=300>
<tr>
    <td><b>Name</b></td>
    <td><b>Title (Author)</b></td>
</tr>
<$loop SearchResults$>
<tr><td><a href="<$URL$>"><$dDocName$></a></td>
    <td><$dDocTitle$> (<$dDocAuthor$>)</td></tr>
<$endloop$>
</table>
</html>
<@end@>
```

This would display a search result page devoid of all images and formatting that you may not need. Consequently, you can format any Content Server response with any Idoc Script include that you want. In theory, the Idoc Script include can contain any kind of formatting that you want: XML, WML, or simply plain text.

For example, if you wanted to return search results in a format that can be read in an Excel Spreadsheet, you could create a resource include that returns a comma-delimited list of entries. You could then save the returned file to your hard drive, and then open it up in Excel. Another useful trick would be to create a resource include that formats the response into a record set that can be read in as a file by the IdcCommandX utility, or the BatchLoader. Such an include could be used with a search result, or an Active Report created with the Web Layout Editor, to build up batch files specific to arbitrary queries against the database or against the search index.

- MergeInclude variables are cached differently than normal resource includes. Therefore, you must restart the Content Server instance if you make changes to the resource include. This can be bypassed if you execute a `docLoadResourceInclude()` function to dynamically load different includes from within the MergeInclude.
- The content type of the returned data is 'text/plain' and not 'text/html' for data returned by a MergeInclude. Some clients (such as Internet Explorer and many versions of Netscape) still display plain text as html if you have valid HTML in the response, others clients may not. If you experience problems, you may need to manually set the content type when you link to it.

3.8 Scoped Local Variables

Scoped local variables are a special kind of local variable, used to override how metadata is drawn to the page. These variables are scoped to a specific metadata field by separating them with a colon.

For example, to hide the title and comments fields, you would set the following flags:

```
dDocTitle:isHidden=1
xComments:isHidden=1
```

These flags must be set early in the page in the URL or by overriding the include `std_doc_page_definitions`.

In the following list, all flags affect the display of the field `xFieldName`:

- `xFieldName:groupHeader`: This is set in Content Profiles if this field is the first field in a group. It contains the HTML and Idoc Script to use for the group header.
- `xFieldName:hasOptionList`: Allows the field to contain a custom option list, instead of using the default option list. Must be used with the `xFieldName:optionListName` variable or `xFieldName:optionListScript` variable.
- `xFieldName:include`: Used to set the value for `fieldInclude` to the name of a custom resource include. This resource will be used throughout the page, including the JavaScript and the HTML. This flag is used rarely. If needed, use the `std_namevalue_field` include file as a guide for making a custom include.
- `xFieldName:isExcluded`: Set to true to exclude a field from the page completely. It will not be displayed as a field, or as a hidden input field. The field will be completely absent from the page.
- `xFieldName:isHidden`: Set to TRUE to hide a field on the page. On pages with form posts, the field will still be present. However, it will only exist as a hidden INPUT field. The value of the field will be blank, unless `xFieldName` or `fieldValue` is defined. This will enable you to create pages with default values that cannot be changed.
- `xFieldName:isInfoOnly`: Set to TRUE to display only the value of a field. This is used instead of `xFieldName:isHidden` to show the user what default values are being submitted.
- `xFieldName:isRelocated`: Set to TRUE to stop the automatic display of a field on the HTML page. By default, all fields on the page have a specific order. To reorder them, you must set this flag, then display the field manually.

  ```
  <!-- hide the comments field -->
  <$xComments:isRelocated = 1$>
  <$loop DocMetaDefinition$>
  <$strTrimWs(inc("std_meta_field_display"))$>
  <$endloop$>
  <!-- now turn off relocation, and display it -->
  <$xComments:isRelocated = ""$>
  <$fieldName="xComments", fieldCaption="Comments", fieldType="Memo"$>
  <$include std_display_field$>
  ```

- `xFieldName:isRequired`: Set to TRUE to turn this field into a required field. This flag must be set in `std_doc_page_definitions`, before the JavaScript validation code is drawn to the page.
- `xFieldName:maxLength`: Similar to `fieldWidth`, this sets the maximum length of a text input field. This is usually greater than `fieldWidth`, and must be less than the width of the field in the database.
- `xFieldName:noSchema`: Set to TRUE to disable a schema option list for a field. Required if you want to generate option lists in a custom, dynamic way.

- `xFieldName:optionListName`: This flag can only be set if a field is an option list. You can override which option list to use to display values:

```
<$xCountry:hasOptionList = 1$>
<$xCountry:noSchema = 1$>
<$xCountry:optionListName = "securityGroups"$>
<$loop DocMetaDefinition$>
<$strTrimWs(inc("std_meta_field_display"))$>
<$endloop$>
```

- `xFieldName:optionListScript`: Similar to `optionListName`, except it can be used to render Idoc Script instead of explicitly defined option lists. This allows the option list to be drawn with a ResultSet instead:

```
<$xCountry:hasOptionList = 1$>
<$xCountry:noSchema = 1$>
<$xCountry:optionListScript =
"<$rsMakeFromList('GROUPS', 'securityGroups')$>" &
"<select>\n" &
"<$loop GROUPS$>" &
" <option><$row$>" &
"<$endloop$>\n" &
"</select>"$>
<$loop DocMetaDefinition$>
<$strTrimWs(inc("std_meta_field_display"))$>
<$endloop$>
```

- `xFieldName:rowClass`: Used in `std_nameentry_row`. It sets a Cascading Style Sheet class for the table row that contains this field.

```
<$xComments:rowClass="xuiPageTitleText"$>
<$loop DocMetaDefinition$>
<$strTrimWs(inc("std_meta_field_display"))$>
<$endloop$>
```

- `xFieldName:rowStyle`: Same as `rowClass`, but this can be used to create inline styles. For example, to hide the Comments field with DHTML, use the following code:

```
<$xComments:rowStyle="display:none"$>
<$loop DocMetaDefinition$>
<$strTrimWs(inc("std_meta_field_display"))$>
<$endloop$>
```

  This is useful when you want to hide and display fields dynamically without a page reload.

4

Using Idoc Script Variables and Functions with Oracle WebCenter Content

This chapter describes the different types of Idoc Script variables and functions and how you can use them with Oracle WebCenter Content. You can use the Idoc Script Custom Scripting Language to customize Oracle WebCenter Content Server. Idoc Script enables you to reference variables, conditionally include content in HTML pages, and loop over results returned from queries. Because Idoc Script is evaluated on the server side rather than the client side, page elements are processed after the browser has made a request, but before the requested page is returned to the client.

This chapter includes the following sections:

4.1 Using Different Types of Idoc Script Variables and Functions

You can use the several different types of Idoc Script variables and functions to customize Content Server:

- Conditional Dynamic Variables
- Dynamic Variables
- Environment Variables
- Global Functions
- Page Variables
- Read-Only Variables
- Settable Variables
- Value Variables

4.1.1 Conditional Dynamic Variables

Some Idoc Script dynamic variables are conditional and can only be used within a conditional statement such as `if`, `while`, `elseif`, or `loop`. These variables have the following special features:

- Conditional variables are internal flags that are *gettable* but not *settable*.
- Conditional variables will only provide a Boolean response and do not return a value such as a string or integer.
- Conditional variables will not accept the `#active` keyword prefix. Thus, an error report is printed to the debug output if the variable is not found.

Appendix A, "Idoc Script Functions and Variables," describes these conditional dynamic variables:

AdminAtLeastOneGroup
AfterLogin
AllowCheckin
AllowCheckout
AllowReview
dcShowExportLink
EmptyAccountCheckinAllowed
HasPredefinedAccounts
HasUrl
HeavyClient
IsCheckinPreAuthed
isComponentEnabled
IsContributor
IsCriteriaSubscription
IsCurrentNav
IsDynamic
IsExternalUser
IsFilePresent
IsFullTextIndexed
isLinkActive
IsLocalSearchCollectionID
IsLoggedIn
IsMac
IsMaxRows
IsMultiPage
IsNotSyncRev
IsPromptingForLogin
IsRequestError
IsSubAdmin
IsSun
IsSysManager
IsUploadSockets
IsUserEmailPresent
IsWindows
IsWorkflow
SingleGroup
UserIsAdmin
UserLanguageID
UserLocaleId
UseXmlUrl

4.1.2 Dynamic Variables

A *dynamic* variable is evaluated on each occurrence of the variable. Each time the variable is encountered, the value is recalculated from code. (In contrast, a *value* variable is evaluated once at the beginning of the service call, and that value is used throughout the service call. See Section 4.1.9, "Value Variables.") Dynamic variables generally return a value such as a string or an integer.

Appendix A, "Idoc Script Functions and Variables," describes this dynamic variable:

AfterLogin

4.1.3 Environment Variables

Web server variables are the CGI environment variables that are set when the server executes the gateway program. To pass data about the information request from the server to the script, the server uses command-line arguments and environment variables. These environment variables can be used to output information to a log file or can be used within Idoc Script statements and as part of evaluations.

For example, this Idoc Script statement evaluates whether the remote host address matches a specific string:

```
<$if strEquals("207.0.0.1",REMOTE_HOST)$>
```

This HTML and Idoc Script markup displays a list of web server environment information on the page:

```
<P>HTTP_INTERNETUSER=<$HTTP_INTERNETUSER$></P>
<P>REMOTE_HOST=<$REMOTE_HOST$></P>
<P>SCRIPT_NAME=<$SCRIPT_NAME$></P>
```

Appendix A, "Idoc Script Functions and Variables," describes these web server variables:

CONTENT_LENGTH
GATEWAY_INTERFACE
HTTP_ACCEPT
HTTP_ACCEPT_ENCODING
HTTP_ACCEPT_LANGUAGE
HTTP_COOKIE
HTTP_HOST
HTTP_INTERNETUSER
HTTP_REFERER
HTTP_USER_AGENT
PATH_INFO
PATH_TRANSLATED
QUERY_STRING
REMOTE_ADDR
REMOTE_HOST
REQUEST_METHOD
SCRIPT_NAME
SERVER_NAME
SERVER_PORT
SERVER_PROTOCOL
SERVER_SOFTWARE

4.1.4 Global Functions

Idoc Script has many built-in global functions. Functions perform actions, including string comparison and manipulation routines, date formatting, and ResultSet manipulation. Some functions also return results, such as the results of calculations or comparisons.

Information is passed to functions by enclosing the information in parentheses after the name of the function. Pieces of information that are passed to a function are called *parameters*. Some functions do not take parameters; some functions take one parameter; some take several. There are also functions for which the number of parameters depends on how the function is being used.

In addition to the built-in global functions, you can define new global functions, including custom classes, with Java code. For more information, see Chapter 11, "Getting Started with Content Server Components," and Chapter 17, "Creating Custom Components."

Appendix A, "Idoc Script Functions and Variables," describes these Idoc Script built-in global functions:

abortToErrorPage()
break()
cacheInclude()
clearSchemaData()
computeDocUrl()
computeRenditionUrl()
dateCurrent()
ddAppendIndexedColumnResultSet()
ddAppendResultSet()
ddApplyTableSortToResultSet()
ddGetFieldList()
ddIncludePreserveValues()
ddLoadIndexedColumnResultSet()
ddLoadResultSet()
ddMergeIndexedColumnResultSet()
ddMergeResultSet()
ddMergeUsingIndexedKey()
ddSetLocal()
ddSetLocalByColumnsFromFirstRow()
ddSetLocalByColumnsFromFirstRowIndexed()
ddSetLocalEmpty()
ddSetLocalEmptyByColumns()
docLoadResourceIncludes()
docRootFilename()
docUrlAllowDisclosure()
dpGet()
dpPromote()
dpPromoteRs()
dpSet()
encodeHtml()
eval()
executeService()
forceExpire()
formatDate()
formatDateDatabase()
formatDateDisplay()
formatDateOnly()

formatDateOnlyDisplay()
formatDateOnlyFull()
formatDateWithPattern()
formatTimeOnly()
formatTimeOnlyDisplay()
getDebugTrace()
getErrorTrace()
getFieldViewDisplayValue()
getFieldViewValue()
getFreeMemory()
getParentValue()
getRequiredMsg()
getTextFile()
getTotalMemory()
getUserValue()
getValue()
getValueForSpecifiedUser()
getViewValue()
getViewValueResultSet()
hasAppRights()
idocTestForInclude()
inc()
incDynamicConversionByRule()
incGlobal()
incTemplate()
indexerSetCollectionValue()
isActiveTrace()
sawflies()
isLayoutEnabled()
isTrue()
isUserOverrideSet()
isValidateFile()
js()
jsFilename()
lc()
lcCaption()
LmDefaultLayout()
LmDefaultSkin()
lmGetLayout()
lmGetSkin()
loadCollectionInfo()
loadDocMetaDefinition()
loadDocumentProfile()
loadSchemaData()
loadSearchOperatorTables()
loadUserMetaDefinition()
optList()
parseDataEntryDate()
parseDateWithPattern()
pneNavigation()
regexMatches()
regexReplaceAll()
regexReplaceFirst()
rptDisplayMapValue()
rs()

rsAddFields()
rsAddFieldsWithDefaults()
rsAddRowCountColumn()
rsAppend()
rsAppendNewRow()
rsAppendRowValues()
rsCopyFiltered()
rsCreateReference()
rsCreateResultSet()
rsDeleteRow()
rsDocInfoRowAllowDisclosure()
rsExists()
rsFieldByIndex()
rsFieldExists()
rsFindRowPrimary()
rsFirst()
rsInsertNewRow()
rsIsRowPresent()
rsLoopInclude()
rsLoopSingleRowInclude()
rsMakeFromList()
rsMakeFromString()
rsMerge()
rsMergeDelete()
rsMergeReplaceOnly()
rsNext()
rsNumFields()
rsNumRows()
rsRemove()
rsRename()
rsRenameField()
rsSetRow()
rsSort()
rsSortTree()
setContentType()
setExpires()
setHttpHeader()
setMaxAge()
setResourceInclude()
setValue()
stdSecurityCheck()
strCenterPad()
strCommaAppendNoDuplicates()
strConfine()
strEquals()
strEqualsIgnoreCase()
strGenerateRandom()
strIndexOf()
strLeftFill()
strLeftPad()
strLength()
strLower()
strRemoveWs()
strReplace()
strReplaceIgnoreCase()

strRightFill()
strRightPad()
strSubstring()
strTrimWs()
strUpper()
toInteger()
trace()
url()
urlEscape7Bit()
userHasAccessToAccount()
userHasGroupPrivilege()
userHasRole()
utGetValue()
utLoad()
utLoadDocumentProfiles()
utLoadResultSet()
xml()

4.1.5 Page Variables

Page variables are set on a particular web page to enable specific page attributes or functionality. A page variable applies only to the page on which it is set.

This section includes the following topics:

- Page Display Variables
- Field Display Variables

4.1.5.1 Page Display Variables

Page variables that affect page display are typically set near the top of the page. Page display variables should be used as read-only variables; setting or changing the value of any of these variables will typically change the way all metadata is displayed on the page, which in most cases is not the desired effect.

Where a listed variable name references another document, it means the variable can be set in Idoc Script or in the URL for a web page.

Appendix A, "Idoc Script Functions and Variables," describes these page display variables:

generateUniqueId
isCheckin
isDocPage
isEditMode
isFormSubmit
isInfo
isQuery
isUpdate
isUploadFieldScript
localPageType
noMCPrefill

4.1.5.2 Field Display Variables

Field display variables can be grouped into the following types:

- Field Information Variables
- Common Field Display Variables
- Other Field Display Variables

4.1.5.2.1 Field Information Variables The following variables define information about a metadata field. The variable values are loaded or computed for each metadata field.

Appendix A, "Idoc Script Functions and Variables," describes these field information variables:

fieldCaption
fieldDefault
fieldIsOptionList
fieldName
fieldOptionListType
fieldType
fieldValue
fieldWidth
isRequired
requiredMsg
valueStyle

Example

The `std_prepare_metafield_include` include in the resource file *IdcHomeDir*`/resources/core/std_page.htm` loads a number of field information variables from the local data in preparation for displaying the current metadata field.

```
<@dynamichtml std_prepare_metafield_include@>
<!--Prepare for presenting field-->
<$fieldName=dName, fieldCaption=dCaption, fieldDefault=dDefaultValue$>
<$fieldType=dType, fieldIsOptionList=dIsOptionList,
fieldOptionListType=dOptionListType$>
<@end@>
```

4.1.5.2.2 Common Field Display Variables There are several commonly used page variables that affect the display of metadata fields. These variables can be set using different syntaxes at different places on a page, depending on how they are being used.

> **Note:** The Profiles and Rules engine in the Configuration Manager applet duplicates this functionality with less code.

The following formats can be used to set a special field display variable:

- **Name/Value pair:** The variable is set using the standard `variable=value` format. For example, `isHidden=1`. This format is typically used to set the display of the current metadata field at the point in the page where the field is being generated by looped code.
- **FieldName:Variable format:** The variable is set by defining it as a parameter for the metadata field it applies to. For example, `myMetadata:isHidden`. This format is typically used at the top of a page to set the global display of a particular metadata field.

If a common field display variable is set at the top of a template page, it should be placed before the <HEAD> tag. Placing the variable in or after the <HEAD></HEAD> section will result in the field being displayed (or not displayed) as you intended, but the JavaScript validation code in the header will still be evaluated, so an … is not an object error will be thrown when you attempt to display a checkin page.

Appendix A, "Idoc Script Functions and Variables," describes these comman field display variables:

isExcluded
isHidden
isInfoOnly
isRelocated
maxLength
optionListScript

If these common field display variables are not sufficient to provide the required flexibility, the entire implementation of a metadata field can be replaced by setting the field variable to the name of a resource include that should be used instead (for example, `myField:include=customInclude`).

The standard implementation is referred to by the variable `defaultFieldInclude`, whose value is different depending on whether the field is being generated on a checkin/update, query, or info page. It also varies considerably based on the type of field being displayed. If the standard field include is overridden, then the new implementation must take into consideration all the issues of the different pages, including JavaScript validation and the Upload applet.

Use this approach only as a last resort. It is preferable to extend existing functionality and set local variables to have custom functionality.

If you use the `include` tag in this way to insert custom HTML code for a special metadata field, you must place the include statement after the </HEAD> tag on the page. If you place it before the </HEAD> tag, the system will insert your custom HTML code into the header and attempt to read it as JavaScript.

4.1.5.2.3 Other Field Display Variables A number of other variables are available to affect the display of metadata fields. Generally, these are used to define the display of a metadata field depending on which field is currently being generated and the value of related common field display variables.

Appendix A, "Idoc Script Functions and Variables," describes these other field display variables:

addEmptyOption
captionEntryWidth
captionFieldWidth
defaultFieldInclude
defaultOptionListScript
fieldCaptionInclude
fieldCaptionStyle
fieldEditWidth
fieldEntryInclude
fieldExtraScriptInclude
fieldInclude
fieldMaxLength
fieldValueStyle
hasOptionList
is Field Excluded

isFieldHidden
isFieldInfoOnly
isFieldMemo
isMultiOption
isStrictList
optionListKey
optionListName
optionListResultSet
optionListValueInclude
optionsAllowPreselect

Example

This example shows how the `compute_std_field_overrides` include in the *IdcHomeDir*`/resources/core/templates/std_page.htm` resource file determines if the field currently being generated is hidden, information only, excluded, and/or relocated. This code is looped over during generation of each metadata field on a page.

```
<@dynamichtml compute_std_field_overrides@>
<$isCustomHidden = getValue("#active", fieldName & ":isHidden")$>
<$if isHidden or isCustomHidden$>
    <$isFieldHidden = 1$>
<$else$>
    <$isFieldHidden = ""$>
<$endif$>
<$isCustomInfo = getValue("#active", fieldName & ":isInfoOnly")$>
<$if isInfo or isCustomInfo or isFieldHidden or isInfoOnly$>
    <$isFieldInfoOnly = 1$>
<$else$>
    <$isFieldInfoOnly = ""$>
<$endif$>
<$isCustomExcluded = getValue("#active", fieldName & ":isExcluded")$>
<$isCustomRelocated = getValue("#active", fieldName & ":isRelocated")$>
<$if isCustomExcluded or (isCustomRelocated and not isRelocated) or isExcluded or
(isFieldHidden and not isFormSubmit)$>
    <$isFieldExcluded = 1$>
<$endif$>
<@end@>
```

4.1.6 Read-Only Variables

Read-only variables can be used to gather information about the current template, the user who is currently logged in, or other current settings. These variables are read-only and cannot be assigned a value.

- Section 4.1.6.1, "Template Read-Only Variables"
- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.1.6.3, "Content Read-Only Variable"
- Section 4.1.6.4, "Other Read-Only Variable"

4.1.6.1 Template Read-Only Variables

Template-related read-only variables make it possible to create conditional content in a template based on the identity of the template. These predefined variables allow you to display the class, file path, name, or type of any template on an Oracle WebCenter Content Server web page. This is particularly useful while you are developing your website.

Appendix A, "Idoc Script Functions and Variables," describes these read-only variables that are related to templates:

TemplateClass
TemplateFilePath
TemplateName
TemplateType

Example

In this example, the internal name of the template appears under the Administration link in the left sidebar of all Oracle WebCenter Content Server web pages. To accomplish this change, the predefined `TemplateName` variable was added to the `pne_nav_admin_links` include that defines the Administration links.

The following is an example of using the `TemplateName` predefined variable to display the internal template name on a web page.

```
<$if IsSubAdmin$>
<tr>
    <td>
        <a href="<$HttpCgiPath$>?IdcService=GET_ADMIN_PAGE&Action=
           GetTemplatePage&Page=ADMIN_LINKS"
        OnMouseOver="imgAct('admin')"
        OnMouseOut="imgInact('admin')">
        <img src="<$HttpImagesRoot$>
           <$button_admin_grey_ish_image$>" width="<$navImageWidth$>"
        height="<$navImageHeight$>" name="admin" border="0"
        alt="<$lc("wwProductAdministration", ProductID)$>"></a>
    </td>
        <td>
        <a class=pneHeader href="<$HttpCgiPath$>?IdcService=GET_ADMIN_
              PAGE&Action=GetTemplatePage&Page=ADMIN_LINKS"
        OnMouseOver="imgAct('admin')"
        OnMouseOut="imgInact('admin')">
        <$lc("wwAdministration")$></a>
    </td>
</tr>
<tr>
    <td colspan=2><font color=#FFFFFF style="Arial"
        size="-1"><$TemplateName$></font></td>
</tr>
<$endif$>
```

4.1.6.2 User Read-Only Variables

User-related read-only variables make it possible to gather information about the current user.

Appendix A, "Idoc Script Functions and Variables," describes these read-only variables that are related to users:

DelimitedUserRoles
ExternalUserAccounts
ExternalUserRoles
UserAccounts
UserAddress
UserAppRights
UserDefaultAccount
UserFullName

UserName
UserRoles

4.1.6.3 Content Read-Only Variable

One content-related read-only variable, SourceID, described in Appendix A, "Idoc Script Functions and Variables," makes it possible to retrieve the Content ID of the current dynamic server page.

This variable returns the same value as `ref:dID`.

4.1.6.4 Other Read-Only Variable

The SafeDir read-only variable, described in Appendix A, "Idoc Script Functions and Variables," is set only as an internal flag. It can be retrieved but not set directly.

4.1.7 Settable Variables

Settable variables can be set within script or used within a CGI string. For example, the variable IsPageDebug can be used as a parameter to a service call to display debug trace information on a page. Setting one of these variables can change the content of the page.

Appendix A, "Idoc Script Functions and Variables," describes these settable variables:

ClientControlled
coreContentOnly
getCookie
HasLocalCopy
HasLocalCopy
IsJava
IsPageDebug
IsSavedQuery
IsSoap
IsXml
isZoneSearchField
Json
setCookie

4.1.8 Workflows

Idoc Script includes predefined functions and variables that are used specifically for workflows.

- **Workflow Functions** perform actions or return results relative to a workflow.
- **Workflow Variables** enable you to set values for variables related to workflows.

For a detailed description of how workflows are implemented in Oracle WebCenter Content Server, see *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

The following points summarize the use of Idoc Script in workflows:

- Workflow jumps are initiated through the evaluation of Idoc Script that is defined for a particular step event (entry, update, or exit).
- As a revision moves from step to step, the system creates a *companion file* that maintains information about the state of the revision in the workflow. You can use the wfGet() and wfSet() functions to code data into the companion file, as needed. Along with user-defined options, the system also maintains the history of what

steps the revision has been to, the last entry time, and the number of times a revision has entered a particular workflow step.

  - Global state information is maintained as the revision moves from step to step.
  - Localized state information is stored with the step and becomes available when a revision is at that step.

- The companion file uses *keys* to keep track of workflow state information. The syntax for a key is:

  ```
  step_name@workflow_name.variable=value
  ```

  For example, the following keys define the value of the entry count and last entry variables for the *Editor* step of a workflow called *Marketing*:

  ```
  Editor@Marketing.entryCount=1
  Editor@Marketing.lastEntryTs={ts '2002-05-28 16:57:00'}
  ```

- All workflow script evaluation occurs inside a database transaction. The result is that any serious errors or aborts that are encountered cause no change to either the database or the companion file. This also means that no Idoc Script workflow function should take more than a negligible amount of time. Consequently, to trigger an external process, an Idoc Script function should be written to execute in a separate thread.

  Caution: If you are using Idoc Script or custom components to load workflow information into the local data, keep in mind that there is a risk of data pollution. This is particularly important if you are loading information for a different revision than the current one.

4.1.8.1 Workflow Functions

Workflow functions perform actions or return results relative to a workflow.

Appendix A, "Idoc Script Functions and Variables," describes these workflow functions:

wfAddActionHistoryEvent()
wfAddUser()
wfComputeStepUserList()
wfCurrentGet()
wfCurrentSet()
wfCurrentStep()
wfDisplayCondition()
wfExit()
wfGet()
wfIsFinishedDocConversion()
wfIsNotificationSuppressed()
wfIsReleasable()
wfLoadDesign()
wfNotify()
wfSet()
wfSetSuppressNotification()
wfUpdateMetaData()

4.1.8.2 Workflow Variables

Workflow variables are used to set values related to workflows.

Appendix A, "Idoc Script Functions and Variables," describes these workflow variables:

dWfName
dWfStepName
entryCount
lastEntryTs
wfAction
wfAdditionalExitCondition
wfGetStepTypeLabel
wfJumpEntryNotifyOff
wfJumpMessage
wfJumpName
wfJumpReturnStep
wfJumpTargetStep
wfMailSubject
wfMessage
wfParentList
wfReleaseDocument
WfStart

4.1.9 Value Variables

A *value* variable is evaluated once at the beginning of a service call and that value is used throughout the service call. The variable is then reevaluated on each new service call. In contrast, a *dynamic* variable is evaluated on each occurrence of the variable. For example, the value variable *isNew* evaluates whether the content item is new or a revision when performing a check in. That evaluation is used throughout the call to the checkin service.

Appendix A, "Idoc Script Functions and Variables," describes these value variables:

AuthorAddress
BrowserVersionNumber
CURRENT_DATE
CURRENT_ROW
DocTypeSelected
DocUrl
DownloadSuggestedName
fileUrl
FIRSTREV
HasOriginal
htmlRefreshTimeout
htmlRefreshUrl
HttpAbsoluteCgiPath
HttpAdminCgiPath
HttpBrowserFullCgiPath
HttpCgiPath
HttpCommonRoot
HttpEnterpriseCgiPath
HttpHelpRoot
HttpImagesRoot
HttpLayoutRoot
HttpRelativeAdminRoot

HttpSystemHelpRoot
HttpWebRoot
IsEditRev
IsFailedConversion
IsFailedIndex
isNew
IsNotLatestRev
MSIE
NoMatches
OneMatch
PageParent
ResultsTitle
StatusCode
StatusMessage
UseHtmlOrTextHighlightInfo

4.2 Using Idoc Script Variables and Functions with Different Features of Oracle WebCenter Content

You can use Idoc Script variables and functions with different features of Oracle WebCenter Content. Appendix A, "Idoc Script Functions and Variables," describes the Idoc Script variables and functions.

For information about system configuration variables that you can use in the WebCenter Content `config.cfg` file, `intradoc.cfg` file, and other `.cfg` files, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

You can use Idoc Script variables and functions with these Oracle WebCenter Content features:

- Batch Loader
- Clients
- Content Items
- Content Profiles
- Content Server
- Conversion
- Database
- Date and Time
- Debugging
- Directories and Paths
- Dynamicdata
- Field Display
- Idoc Script
- Indexing
- Localization
- Page Display
- Personalization

- ResultSets
- Schemas
- Searching
- Security
- Strings
- Templates
- Users
- Web Servers
- Workflow

4.2.1 Batch Loader

The following Idoc Script variable, described in Appendix A, "Idoc Script Functions and Variables," is related to the Batch Loader utility:

BatchLoaderPath

For information about other Idoc Script configuration variables that are related to the Batch Loader utility, see "Batch Loader" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

4.2.2 Clients

The following Idoc Script variables are related to client applications:

BrowserVersionNumber
ClientControlled
DownloadApplet
HasLocalCopy
HeavyClient
HttpBrowserFullCgiPath
IsMac
IsSun
IsUploadSockets
IsWindows
MSIE
MultiUpload
UploadApplet

4.2.3 Content Items

The following Idoc Script variables and functions are related to content items:

AutoNumberPrefix
computeDocUrl()
computeRenditionUrl()
CONTENT_LENGTH
DocTypeSelected
DocUrl
DownloadSuggestedName
fileUrl
FIRSTREV

HasLocalCopy
HasOriginal
HasUrl
IsCriteriaSubscription
IsEditRev
IsFailedConversion
IsFailedIndex
IsFilePresent
IsFullTextIndexed
IsLocalSearchCollectionID
isNew
IsNotLatestRev
IsNotSyncRev
MajorRevSeq
MinorRevSeq
NotificationQuery
SingleGroup
SourceID

4.2.4 Content Profiles

The following Idoc Script functions are used to manage content profiles:

dpGet()
dpPromote()
dpPromoteRs()
dpSet()
getFieldViewValue()
getRequiredMsg()
loadDocumentProfile()
utLoadDocumentProfiles()

4.2.5 Content Server

The following Idoc Script variables and functions are related to the Oracle WebCenter Content Server system:

DownloadApplet
getFieldConfigValue
HasExternalUsers
HttpAbsoluteCgiPath
HttpAdminCgiPath
HttpCgiPath
HttpCommonRoot
HttpEnterpriseCgiPath
HttpHelpRoot
HttpImagesRoot
HttpLayoutRoot
HttpRelativeAdminRoot
HttpSystemHelpRoot
InstanceDescription
isComponentEnabled
IsContributor
isLayoutEnabled()
IsPromptingForLogin
IsRequestError

isValidateFile()
LmDefaultLayout()
LmDefaultSkin()
lmGetLayout()
lmGetSkin()
MultiUpload
StatusCode
StatusMessage
SysAdminAddress
UploadApplet

4.2.6 Conversion

You can use some Idoc Script variables with these conversion products:

- Inbound Refinery
- Dynamic Converter

4.2.6.1 Inbound Refinery

The following Idoc Script variable is related to the Oracle WebCenter Content: Inbound Refinery system:

IsFailedConversion

4.2.6.2 Dynamic Converter

The following Idoc Script variables and function are related to the Dynamic Converter:

dcShowExportLink
ForcedConversionRules
incDynamicConversionByRule()
IsDynamicConverterEnabled

4.2.7 Database

The following Idoc Script function is related to databases:

formatDateDatabase()

4.2.8 Date and Time

The following Idoc Script variables and functions are related to formatting and manipulating dates and times:

CURRENT_DATE
dateCurrent()
formatDate()
formatDateDatabase()
formatDateDisplay()
formatDateOnly()
formatDateOnlyDisplay()
formatDateOnlyFull()
formatDateWithPattern()
formatTimeOnly()
formatTimeOnlyDisplay()
lastEntryTs
parseDataEntryDate()

parseDate
parseDateWithPattern()

4.2.9 Debugging

The following Idoc Script variables and functions are related to debugging:

getDebugTrace()
getErrorTrace()
getFreeMemory()
getTotalMemory()
isActiveTrace()
IsJava
IsPageDebug
IsRequestError
IsSoap
isVerboseTrace
IsXml
Json
trace()

For information about other Idoc Script configuration variables that are related to debugging, see "Debugging" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

4.2.10 Directories and Paths

The following Idoc Script variables and function are related to directories and file paths:

docRootFilename()
DownloadSuggestedName
HttpAbsoluteCgiPath
HttpAdminCgiPath
HttpBrowserFullCgiPath
HttpCgiPath
HttpCommonRoot
HttpEnterpriseCgiPath
HttpHelpRoot
HttpImagesRoot
HttpLayoutRoot
HttpRelativeAdminRoot
HttpRelativeWebRoot
HttpServerAddress
HttpSystemHelpRoot
HttpWebRoot
SafeDir

4.2.11 Dynamicdata

The following Idoc Script functions are related to `dynamicdata` tables:

ddAppendIndexedColumnResultSet()
ddAppendResultSet()
ddApplyTableSortToResultSet()
ddGetFieldList()
ddIncludePreserveValues()

ddLoadIndexedColumnResultSet()
ddLoadResultSet()
ddMergeIndexedColumnResultSet()
ddMergeResultSet()
ddMergeUsingIndexedKey()
ddSetLocal()
ddSetLocalByColumnsFromFirstRow()
ddSetLocalByColumnsFromFirstRowIndexed()
ddSetLocalEmpty()
ddSetLocalEmptyByColumns()

4.2.12 Field Display

The following Idoc Script variables and functions are related to the display of metadata fields on Oracle WebCenter Content Server web pages:

captionEntryWidth
captionFieldWidth
defaultFieldInclude
defaultOptionListScript
fieldCaption
fieldCaptionInclude
fieldCaptionStyle
fieldDefault
fieldEditWidth
fieldEntryInclude
fieldExtraScriptInclude
fieldInclude
fieldIsOptionList
fieldMaxLength
fieldName
fieldOptionListType
fieldType
fieldValue
fieldValueStyle
fieldWidth
hasOptionList
isExcluded
is Field Excluded
isFieldHidden
isFieldInfoOnly
isFieldMemo
isHidden
isInfoOnly
isMultiOption
IsOverrideFormat
isRelocated
isRequired
isStrictList
maxLength
optionListKey
optionListName
optionListResultSet
optionListScript
optionListValueInclude
optionsAllowPreselect

optList()
requiredMsg
valueStyle

4.2.13 Idoc Script

The following Idoc Script functions are not related to any specific functionality:

break()
cacheInclude()
docLoadResourceIncludes()
eval()
executeService()
forceExpire()
getTextFile()
getUserValue()
getValue()
inc()
incGlobal()
incTemplate()
sawflies()
isTrue()
setExpires()
setHttpHeader()
setMaxAge()
setResourceInclude()

4.2.13.1 Keywords

The following Idoc Script variables are related to keywords:

c
exec
include

4.2.14 Indexing

The following Idoc Script variables and function are related to indexing:

indexerSetCollectionValue()
IsFailedIndex
IsFullTextIndexed
MaxCollectionSize

4.2.15 Localization

The following Idoc Script variables and functions are related to localization:

lc()
lcCaption()
rptDisplayMapValue()
UserLanguageID
UserLocaleId
UserLanguageID

4.2.16 Page Display

The following Idoc Script variables and functions are related to the display of Oracle WebCenter Content Server web pages:

abortToErrorPage()
AfterLogin
coreContentOnly
docLoadResourceIncludes()
generateUniqueId
getHelpPage
getOptionListSize
htmlRefreshTimeout
htmlRefreshUrl
isCheckin
IsCurrentNav
isDocPage
IsDynamic
isEditMode
isFormSubmit
isInfo
IsMaxRows
IsMultiPage
isNew
isQuery
IsSavedQuery
isUpdate
isUploadFieldScript
localPageType
noMCPrefill
PageParent
pneNavigation()
ResultsTitle
TemplateClass
TemplateFilePath
TemplateName
TemplateType

4.2.17 Personalization

The following Idoc Script functions are related to user topic (personalization) files:

utGetValue()
utLoad()
utLoadDocumentProfiles()
utLoadResultSet()

4.2.18 ResultSets

The following Idoc Script variable and functions are related to ResultSets:

CURRENT_ROW
getValue()
loadDocMetaDefinition()
loadSearchOperatorTables()
loadUserMetaDefinition()
rs()

rsAddFields()
rsAddFieldsWithDefaults()
rsAddRowCountColumn()
rsAppend()
rsAppendNewRow()
rsAppendRowValues()
rsCopyFiltered()
rsCreateReference()
rsCreateResultSet()
rsDeleteRow()
rsDocInfoRowAllowDisclosure()
rsExists()
rsFieldByIndex()
rsFieldExists()
rsFindRowPrimary()
rsFirst()
rsInsertNewRow()
rsIsRowPresent()
rsLoopInclude()
rsLoopSingleRowInclude()
rsMakeFromList()
rsMakeFromString()
rsMerge()
rsMergeDelete()
rsMergeReplaceOnly()
rsNext()
rsNumFields()
rsNumRows()
rsRemove()
rsRename()
rsRenameField()
rsSetRow()
rsSort()
rsSortTree()
setContentType()
setValue()
utLoadResultSet()

4.2.19 Schemas

The following Idoc Script functions are related to schemas.

A schema is *republished* whenever a change occurs that might affect the relationship between the parts of the schema. These settings relate to publishing factors:

clearSchemaData()
getFieldViewDisplayValue()
getParentValue()
getViewValue()
getViewValueResultSet()
jsFilename()
loadSchemaData()

4.2.20 Searching

The following Idoc Script variables and functions are related to searching:

EnableDocumentHighlight
indexerSetCollectionValue()
IsCurrentNav
IsFullTextIndexed
IsLocalSearchCollectionID
IsMultiPage
IsSavedQuery
isZoneSearchField
loadCollectionInfo()
NoMatches
OneMatch
QUERY_STRING
regexMatches()
regexReplaceAll()
regexReplaceFirst()
ResultsTitle
UseHtmlOrTextHighlightInfo
UseXmlUrl

4.2.21 Security

The following sections list Idoc Script variables and functions that are related to security:

- Section 4.2.21.1, "Internal Security"
- Section 4.2.21.2, "External Security"

4.2.21.1 Internal Security

The following Idoc Script variables and functions are related to internal security:

AdminAtLeastOneGroup
AllowCheckin
AllowCheckout
AuthorDelete
DefaultAccounts
DelimitedUserRoles
docUrlAllowDisclosure()
EmptyAccountCheckinAllowed
encodeHtml()
ExclusiveCheckout
GetCopyAccess
hasAppRights()
HasPredefinedAccounts
IsCheckinPreAuthed
IsLoggedIn
IsSubAdmin
IsSysManager
isUserOverrideSet()
rsDocInfoRowAllowDisclosure()
SelfRegisteredAccounts
SelfRegisteredRoles
SelfRegisteredAccounts
stdSecurityCheck()
UserAccounts
UserAppRights

UserDefaultAccount
userHasAccessToAccount()
userHasGroupPrivilege()
userHasRole()
UserIsAdmin
UserRoles

4.2.21.2 External Security

The following Idoc Script variables are related to external security (web server and NTLM security).

HasExternalUsers
IsPromptingForLogin
IsUploadSockets
UseSSL

4.2.22 Strings

The following Idoc Script variables and functions are related to strings:

js()
strCenterPad()
strCommaAppendNoDuplicates()
strConfine()
StrConfineOverflowChars
strEquals()
strEqualsIgnoreCase()
strGenerateRandom()
strIndexOf()
strLeftFill()
strLeftPad()
strLength()
strLower()
strRemoveWs()
strReplace()
strReplaceIgnoreCase()
strRightFill()
strRightPad()
strSubstring()
strTrimWs()
strUpper()
toInteger()
url()
urlEscape7Bit()
xml()

4.2.23 Templates

The following Idoc Script variables are related to templates:

TemplateClass
TemplateFilePath
TemplateName
TemplateType

4.2.24 Users

The following Idoc Script variables and functions are related to users:

AdminAtLeastOneGroup
AllowCheckin
AllowCheckout
AuthorAddress
DefaultAccounts
DelimitedUserRoles
ExclusiveCheckout
ExternalUserAccounts
ExternalUserRoles
getUserValue()
getValueForSpecifiedUser()
HasPredefinedAccounts
IsCheckinPreAuthed
IsExternalUser
IsLoggedIn
IsSubAdmin
IsSysManager
IsUserEmailPresent
isUserOverrideSet()
SysAdminAddress
UserAccounts
UserAddress
UserAppRights
UserDefaultAccount
UserFullName
userHasAccessToAccount()
userHasGroupPrivilege()
userHasRole()
UserIsAdmin
UserName
UserRoles
wfAddUser()

4.2.25 Web Servers

The following Idoc Script variables are related to web servers.

GATEWAY_INTERFACE
getCookie
HTTP_ACCEPT
HTTP_ACCEPT_ENCODING
HTTP_ACCEPT_LANGUAGE
HTTP_COOKIE
HTTP_HOST
HTTP_INTERNETUSER
HTTP_REFERER
HTTP_USER_AGENT
IdcAuthExtraRequestParams
PATH_INFO
QUERY_STRING
REMOTE_ADDR
REMOTE_HOST
REQUEST_METHOD

SCRIPT_NAME
SERVER_NAME
SERVER_PORT
SERVER_PROTOCOL
SERVER_SOFTWARE
setCookie
UseSSL

4.2.26 Workflow

The following sections list Idoc Script variables and functions that are related to workflows:

- Section 4.2.26.1, "Global Function"
- Section 4.2.26.2, "Workflow Functions"
- Section 4.2.26.3, "Other Variables"

4.2.26.1 Global Function

The following Idoc Script global function is related to workflows:

getValueForSpecifiedUser()

4.2.26.2 Workflow Functions

The following Idoc Script functions are related to workflows:

wfAddActionHistoryEvent()
wfAddUser()
wfComputeStepUserList()
wfCurrentGet()
wfCurrentSet()
wfCurrentStep()
wfDisplayCondition()
wfExit()
wfGet()
wfGetStepTypeLabel
wfIsFinishedDocConversion()
wfIsNotificationSuppressed()
wfIsReleasable()
wfLoadDesign()
wfNotify()
wfReleaseDocument
wfSet()
wfSetSuppressNotification()
wfUpdateMetaData()

4.2.26.3 Other Variables

The following Idoc Script variables are related to workflows:

AllowReview
dWfName
dWfStepName
entryCount
IsEditRev
IsWorkflow
lastEntryTs

SingleGroup
wfAction
wfJumpEntryNotifyOff
wfJumpMessage
wfJumpName
wfJumpReturnStep
wfJumpTargetStep
wfMailSubject
wfMessage
wfParentList
WfStart

Part III

Changing the Look and Feel of the Content Server Interface

This part provides information about the several different methods that you can use to change the appearance and navigation of the Oracle WebCenter Content Server interface.

Part III contains the following chapters:

5

Customizing the Content Server Interface

This chapter provides information about the several different methods that you can use to customize the look and feel of the Oracle WebCenter Content Server interface. You can use skins and layouts to change the appearance of the user interface and dynamic server pages to change the navigation.

This chapter includes the following sections:

Tip: In addition to using the methods discussed in this chapter, you can alter the metadata fields that are presented to users and modify the types of presentations used for check-in pages, search pages, and other user interfaces. For information about creating and modifying metadata fields and creating content profiles, see "Customizing Repository Fields and Metadata" and "Managing Content Profiles" in *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

5.1 About Customizing the Content Server Interface

Skins and *layouts* provide alternate color schemes and alternate navigation designs.

5.1.1 Types of Skins and Layouts

Some skins and layouts are provided by default with Content Server. In addition, you can design custom skins and layouts. When you change the skin or layout, you change the look and feel of the interface. You can select a skin and layout from the options provided on the User Profile page.

The only skills required to create and modify skins or layouts is an understanding of HTML, Cascading Style Sheets, and JavaScript. After altering the appearance, the edited layouts and skins are published so that others in your environment can use them.

Note: Only administrators can make new or custom skins. For more information about setting the default look and feel of the user interface, see Section 5.3, "Configuring a Default Skin and Layout for New Users and Guests."

5.1.2 Skins

Skins define the color scheme and other aspects of appearance of the layout such as graphics, fonts, or font size. (the default skin is `Oracle`). You can design custom skins or modify the existing skins.

5.1.3 Layouts

Layouts define the navigation hierarchy display (the default layout is Trays) and custom layouts can be designed.

Custom layouts change behavior and the look-and-feel systemwide. If you want your changes to apply only in limited situations, you might want to consider dynamic server pages.These layouts are provided:

- **Trays**: This layout with the standard Oracle skin is the default interface. High-level navigation occurs through the navigation trays.
- **Top Menus:** This layout provides an alternate look with top menus providing navigation.

5.2 Choosing a Different Skin or Layout

You can choose a different skin to provide an alternate color scheme or a different layout to provide an alternate navigation design, or both.

5.2.1 How to Choose a Different Skin or Layout

The **User Personalization** settings available on the User Profile page enable users to change the layout of Content Server or the skin.

Important: This personalization functionality works with Internet Explorer 7+ or Mozilla Firefox 3+ and later versions.

To choose a different skin or layout:

1. On the Content Server Home page, click ***your_user_name*** in the top menu bar.

   The User Profile page appears.

2. Choose the desired skin and layout.
3. Click **Update**, and view the changes.

5.2.2 What Happens at Runtime

After you choose a different skin or layout, it becomes the user interface for Content Server whenever you log in.

5.3 Configuring a Default Skin and Layout for New Users and Guests

These values can be placed in the *IntradocDir*/config/config.cfg file to alter the default behavior for the Content Server instance:

- **LmDefaultSkin**: The name of the skin used by guests, and new users. The default is Oracle.
- **LmDefaultLayout**: The name of the layout used by guests, and new users. The default is Trays, but it can be set to Top Menus.

5.4 Modifying the Template for a Skin or Layout

The Top Menus and Trays layouts are included by default with the system. The two layouts have two skin options (Oracle and Oracle2). The layouts are written in JavaScript, and the *look* of the skins is created using Cascading Style Sheets.

You can modify skins and layouts by altering the template files provided with Content Server or design new skins and layouts by creating components that can be shared with other users.

5.4.1 About Dynamic Publishing

When Content Server starts, or when the PUBLISH_WEBLAYOUT_FILES service is run, the PublishedWeblayoutFiles table in the std_resource.htm file is used to publish files to the weblayout directory. To have your custom component use this publishing mechanism, create a template, and then merge a custom row that uses that template into the PublishedWeblayoutFiles table.

Other users who want to modify or customize your file can override your template or your row in the PublishedWeblayoutFiles table. If your template uses any resource includes, other users can override any of these includes or insert their own Idoc Script code using the standard super notation. When your component is disabled, the file is no longer published or modified and Content Server returns to its default state.

In addition to giving others an easy way to modify and add to your work, you can also construct these former static files using Idoc Script. For example, you can have the files change depending on the value of a custom configuration flag. You can use core Content Server objects and functionality by writing custom Idoc Script functions and referencing them from inside your template.

Because this Idoc Script is evaluated once during publishing, you cannot use Idoc Script as you would normally do from the *IdcHomeDir*/resources/core/idoc/std_page.idoc file. When a user requests that file, it has already been created, so the script that was used to create it did not have any access to the current service's DataBinder object or to any information about the current user.

This does limit the type of Idoc Script you can write in these files. If you are writing CSS or JavaScript that needs information that dynamically changes with users or services, consider having the pages that need this code include the code inline. This increases the size of pages delivered by your web server and so increases the amount of bandwidth used.

5.5 Altering the Anonymous User Interface

The ExtranetLook component can be used to change the interface for anonymous, random users. An example of this is when a website based on Content Server must be available to external customers without a login, but you want employees to be able to contribute content to that website.

When Content Server is running on Oracle WebLogic Server, the ExtranetLook component alters privileges for certain pages so that they require write privilege to access. The component also makes small alterations to the static portal page to remove links that anonymous, random users should not see.

Note: The ExtranetLook component does not provide form-based authentication for Oracle WebLogic Server or provide customizable error pages.

The ExtranetLook component is installed (disabled) with Content Server. To use the component, you must enable it with the Component Manager.

You can customize your web pages to make it easy for customers to search for content, and then give employees a login that permits them to see the interface on login. To do the customization, modify the `ExtranetLook.idoc` file, which provides dynamic resource includes that can be customized based on user login. The IDOC file is checked in to the Content Server repository so it can be referenced by the Content Server templates.

5.5.1 How to Alter the Anonymous User Interface

You can update the look and feel of the anonymous user interface for the Content Server website by altering the following files in the *`IntradocDir`*`/data/users/` directory:

- `prompt_login.htm`
- `access_denied.htm`
- `report_error.htm`

To alter the anonymous user interface:

1. Display the Web Layout Editor.
2. From the **Options** menu, choose **Update Portal**.
3. Modify the portal page as you want to. You can use dynamic resource includes to customize this page.
4. Click **OK**.
5. Customize the `ExtranetLook.idoc` file as you want to.
6. Check out the `ExtranetLook` content item from Content Server.
7. Check in the revised `ExtranetLook.idoc` file to Content Server.

5.5.2 What Happens at Runtime

After you modify the portal page and customize the`ExtranetLook.idoc` file, your design becomes the user interface for Content Server whenever a user goes to the website without logging in.

5.6 Changing the URL of the Login Page

You can change the URL of the Login page for Content Server by changing its context root, which is normally `/cs/`. You cannot change the URL by setting a relative context root with the `HttpRelativeWebRoot` property because the value of this property does not apply to the Login page. If you need to change the web location where users log in, you can redeploy the WebCenter Content application with a deployment plan.

To change the URL of the Login page:

1. Log in to the Oracle WebLogic Server Administration Console as the administrator of the domain where WebCenter Content is deployed.
2. Click **Deployments** under the name of your domain, in the Domain Structure area on the left.
3. Click **Oracle WebCenter Content - Content Server** in the **Deployments** table on the **Control** tab of the Summary of Deployments page.

   This application may be on the second or third page of the table.
4. Note the path to the deployment plan.

   If no plan is specified for your WebCenter Content instance, you can create one:

   a. Click **Configuration** on the Settings for Oracle WebCenter Content - Content Server page.

   b. Change the value of any parameter on the **Configuration** tab.

   c. Click **Save**.

   d. Confirm the path to the deployment plan on the Save Deployment Plan Assistant page, or change the path.

   e. Click **OK**.
5. In a text editor, add lines at two places in the deployment plan:

   a. Add the `original_loginpage_path` and `original_loginerror_path` variables, each in a `<variable>` element of a `<variable-definition>` element, as in this example:

```
<deployment-plan xmlns="http://xmlns.oracle.com/weblogic/deployment-plan"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.oracle.com/weblogic/deployment-plan
http://xmlns.oracle.com/weblogic/deployment-plan/1.0/deployment-plan.xsd">
    <application-name>ServletPlugin</application-name>
    <variable-definition>
     <variable>
       <name>original_loginpage_path</name>
       <value>/content/login/login.htm</value>
     </variable>
     <variable>
       <name>original_loginerror_path</name>
       <value>/content/login/error.htm</value>
     </variable>
      <variable>
        <name>SessionDescriptor_timeoutSecs_12996472139160</name>
        <value>3600</value>
     </variable>
```

   b. In the `<module-descriptor>` element of `web.xml`in the `cs.war` file, add two `<variable-assignment>` elements that assign the following values to

the `original_loginpage_path` and `original_loginerror_path` variables, respectively:

* `/web-app/login-config/form-login-config/form-login-page`
* `/web-app/login-config/form-login-config/form-error-page`

For example:

```
<module-override>
  <module-name>cs.war</module-name>
  <module-type>war</module-type>
  <module-descriptor external="false">
    <root-element>weblogic-web-app</root-element>
    <uri>WEB-INF/weblogic.xml</uri>
  </module-descriptor>
  <module-descriptor external="false">
    <root-element>web-app</root-element>
    <uri>WEB-INF/web.xml</uri>
    <variable-assignment>
      <name>original_loginpage_path</name>

<xpath>/web-app/login-config/form-login-config/form-login-page</xpath>
      </variable-assignment>
      <variable-assignment>
        <name>original_loginerror_path</name>

<xpath>/web-app/login-config/form-login-config/form-error-page</xpath>
     </variable-assignment>
    </module-descriptor>
  </module-override>
  <module-override>
```

6. Stop the WebCenter Content Managed Server (`UCM_server1` by default), with the `stopManagedWebLogic` script.
   - **UNIX script:** *DomainHome*`/bin/stopManagedWebLogic.sh UCM_server1`
   - **Windows script:** *DomainHome*`\bin\stopManagedWebLogic.cmd UCM_server1`
7. In the Administration Console, click **Deployments** under the name of your domain.
8. Select **Oracle WebCenter Content - Content Server** in the **Deployments** table, and click **Update**.
9. Select **Redeploy this application using the following deployment files**, make sure the path to the deployment plan is correct, and then click **Finish**.
10. After the redeployment completes successfully, click **Apply Changes**.
11. Start the WebCenter Content Managed Server with the `startManagedWebLogic` script.
    - **UNIX script:** *DomainHome*`/bin/startManagedWebLogic.sh UCM_server1`
    - **Windows script:** *DomainHome*`\bin\startManagedWebLogic.cmd UCM_server1`
12. In the Administration Console, click **Deployments**.

13. Select **Oracle WebCenter Content - Content Server** in the **Deployments** table, and from the **Start** menu, choose **Servicing all requests**.
14. After the WebCenter Content application is launched, verify that the URL of the login page has changed.

5.7 Creating and Publishing a New Layout

The following general steps are necessary to create and publish new layouts:

1. Merge a table into the `LmLayouts` table in *`IdcHomeDir`*`/resources/core/tables/std_publishing.htm` to define the new layout. Define the layout ID, label, and whether it is enabled (set to `1`) or not.
2. Merge a table into the `PublishedWeblayoutFiles` table in *`IdcHomeDir`*`/resources/core/tables/std_publishing.htm`. This new table describes the files that are created from Content Server templates and then pushed out to the `weblayout` directory. Specify the necessary `skin.css` files to push out to each skin directory.
3. Merge a table with the `PublishedStaticFiles` table in `std_publishing.htm`. This lists the directories that contain files, such as `images`, that should be published to the `weblayout` directory.

5.8 Optimizing the Use of Published Files

You can direct Content Server to bundle published files so that they can be delivered as one, minimizing the number of page requests to the server. In addition, you can optimize file use by referencing published pages using Idoc Script.

5.8.1 Bundling Files

Multiple resources may be packaged together into units called bundles. A bundle is a single file containing one or more published resources. Only JavaScript and `css` resources should be bundled and only with other resources of the same type. Bundling helps reduce the client overhead when pages are loaded but increases client parse, compile, and execute overhead. Generally, it is recommended to bundle resources that have some thematic similarity or are expected to be included at similar times. For example, if you know that resources A, B, and C are needed on every page, and resources D, E, and F are needed rarely but are all needed together, it is recommended to bundle A, B, and C together and to put D, E, and F into a separate bundle.

Almost all JavaScript resources for the Content Server core are bundled into one of two bundles: `yuiBundle.js`, which contains script provided by the third-party Yahoo User Interface library, and `bundle.js`, which contains the rest of the resources.

The `PublishedBundles` table is used for determining how resources are bundled. Essentially a bundle is identified by its target `bundlePath`, which is the path name to the bundle (relative to the `weblayout` directory), and a list of rules detailing which resource classes are included or excluded. A `loadOrder` value in this table applies only to the order in which the filtering rules are applied, not the order in which the resources appear in the bundle.

Note: The bundling has changed since Oracle Universal Content Management 10*g*, which used a different table and had a `loadOrder` value that determined the order of resources in each bundle.

Static `weblayout` file contents are cached on client machines and on web proxies, significantly lowering the amount of server bandwidth they use. Therefore, the best practice is to use these types of files wherever possible.

However, each static `weblayout` file requested by the client's browser requires a round-trip to the server just to verify that the client has the most up-to-date version of the file. This occurs even if the file is cached. As the number of these files grows, so does the number of downloads from the server for each page request.

To help minimize the number of round-trips, Content Server can bundle multiple published files so that they are delivered as one. You can disable this feature by setting the following configuration in the server's *IntradocDir*`/config/config.cfg` file:

```
BundlePublishedWeblayoutFiles=false
```

Bundling is accomplished by using the `PublishedBundles` table in the `std_publishing.htm` file, which Example 5–1 shows.

Example 5–1 PublishedBundles Table in std_publishing.htm File

```
<@table PublishedBundles@>
<table border=1><caption><strong>
    <tr>
        <td>bundlePath</td>
        <td>includeClass</td>
        <td>excludeClass</td>
        <td>loadOrder</td>
    </tr>
    <tr>
        <td>resources/bundle.js</td>
        <td>javascript:common</td>
        <td></td>
        <td>128</td>
    </tr>
. . .
</table>
<@end@>
```

The columns in this table are as follows:

- `bundlePath`: The eventual location where the bundle is published. This path is relative to the `weblayout` directory.
- `includeClass`: This is used to determine which resources to include in a bundle.
- `excludeClass`: This is used to determine which resources to exclude from a bundle.
- `loadOrder`: The order in which the `includeClass` and `excludeClass` filters are applied.

In the previous example, files of the `javascript:common` class are published to a single bundle located at `resources/layouts/commonBundle.js`. The contents of all bundled files that match this class are appended to form a single file to be stored at that location.

5.8.2 Referencing Published Files

Most published files (both bundled and unbundled) must be directly referenced from within HTML to be included in a page. It can therefore be difficult to know exactly which files to include for a given situation, especially when bundling can be enabled

or disabled by server administrators. A simple Idoc Script method can be used to easily and transparently include all of the files you need on a given page.

For example, if you write a page that includes all files associated with the `javascript:common` bundle (as described previously), then do not write HTML that includes all of the files mentioned in the first table in addition to the bundle mentioned in the second, the server is asked for each file. This negates the purpose of bundling because the server is pinged for each file whether it actually exists or not.

Example 5–2 shows Idoc Script code, within the `HEAD` section for a page, to correctly include these files on the page.

Example 5–2 Idoc Script to Reference a Bundle of Files

```
<$exec createPublishedResourcesList("javascript:common")$>
<$loop PublishedResources$>
<script language="JavaScript" src="<$HttpWebRoot$><$PublishedResources.path$>" />
</script>
<$endloop$>
```

This code fragment includes all `javascript:common` files even if bundling is switched off. If `javascript` instead of `javascript:common` is passed, all files whose class starts with `javascript` are included.

This `PublishedResources` ResultSet is sorted by `loadOrder`, so files and bundles with the lowest `loadOrder` are included first. Files with a greater `loadOrder` can override JavaScript methods or CSS styles that were declared earlier.

6

Creating Dynamic Server Pages

This chapter describes how to use the building blocks necessary for creating dynamic server pages to alter the appearance and navigation of web pages.

This chapter includes the following sections:

6.1 About Dynamic Server Pages

Dynamic server pages are files that are checked in to Content Server and then used to generate web pages dynamically. Dynamic server pages are typically used to alter the look-and-feel and the navigation of web pages. For example, dynamic server pages can be used to do these tasks:

- Create web pages to be published through Content Publisher
- Implement HTML forms
- Maintain a consistent look-and-feel throughout a website

Dynamic server pages include the following file formats:

- **IDOC**: A proprietary scripting language
- **HCST**: Hypertext Content Server Template, similar to a standard Content Server template page stored in the *IdcHomeDir*`/resources/core/templates/` directory
- **HCSP**: Hypertext Content Server Page, an HTML-compliant version of the HCST page, usually used for published content
- **HCSF**: Hypertext Content Server Form, similar to HCSP and HCST pages, but containing HTML form fields that can be filled out and submitted from a web browser

When you use dynamic server pages, Content Server assembles web pages dynamically using a custom template (HCST, HCSP, or HCSF file) that you have checked in to Content Server. The template calls HTML includes from a text file (IDOC file) that you have also checked in to Content Server.

To make changes to the look-and-feel or navigation on a web page, you modify the HCS* template page, or the IDOC file, or both, and then check in the revised files as new revisions. Your changes are available immediately.

Using dynamic server pages with Content Server gives you these advantages:

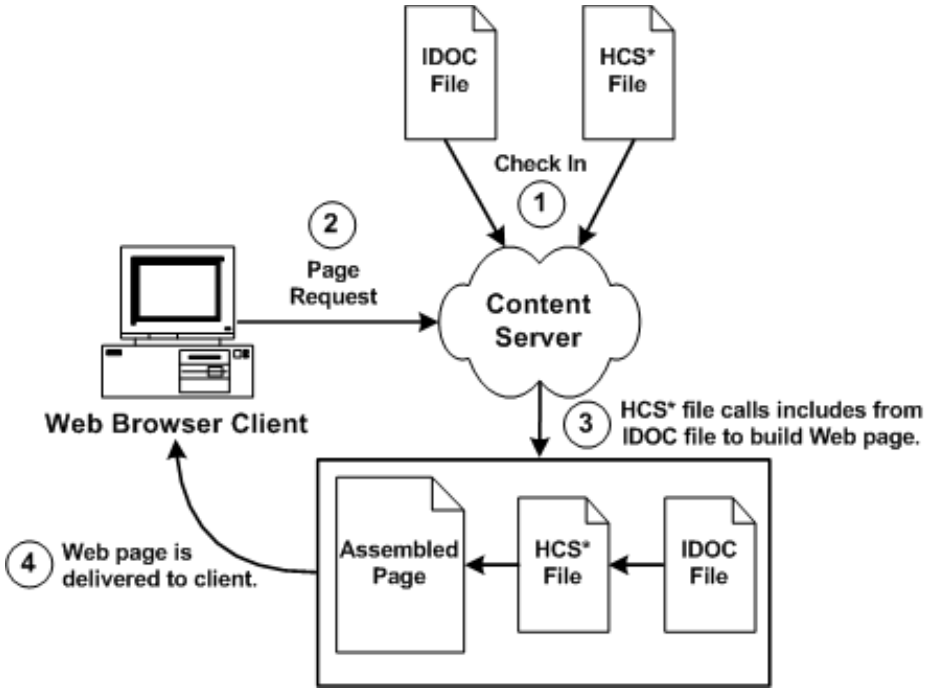
- **You can introduce and test customizations quickly and easily.** Simply checking in a revision of a dynamic server page implements the changes immediately—you do not have to restart Content Server.
- **Your web pages can make use of functionality not found in standard HTML.** For example, HTML forms can be submitted directly to Content Server without the need for CGI scripts. Also, Idoc Script enables you to work directly with environment and state information about Content Server.
- **You do not have to install or keep track of component files.** It can be difficult to maintain and troubleshoot components if they have a lot of files or your system is highly customized. Dynamic server pages are easier to work with because you can check in just a few content items that contain all of your customizations.
- **Customizations can be applied to individual pages.** Dynamic server pages enable you to apply customizations to a single page rather than globally, leaving the standard Content Server page coding intact.

Keep the following constraints in mind when deciding whether to use dynamic server pages:

- **Dynamic server pages cannot be used to modify core functionality of Content Server.** Dynamic server pages are most useful for customizing your web design and form pages.
- **Frequent revisions to dynamic server pages can result in a large number of obsolete content items.** You should do as much work on a development system as possible before deploying to a production instance, and you may need to delete out-of-date pages regularly.

Figure 6–1 shows the process for generating and using a Dynamic Server Page.

Figure 6–1 The Dynamic Server Page Process



6.1.1 Page Types

There are four types of dynamic server pages, which are identified in Content Server by their four-character file-name extensions:

- IDOC
- HCST
- HCSP
- HCSF

6.1.1.1 IDOC File

An IDOC file is a text file containing HTML includes that are called by HCST, HCSP, and HCSF pages.

For more information about includes, see Chapter 11, "Getting Started with Content Server Components."

6.1.1.2 HCST File

A Hypertext Content Server Template (HCST) file is a template page, similar to a standard Content Server template page, that is used as a framework for assembling a web page.

- HCST pages are typically used when the content of the page itself is dynamic or where Content Server functionality is needed, such as on a search page, search results page, or custom check-in page.
- Because this type of page consists mostly of dynamically assembled code, HCST files are not indexed in Content Server.

6.1.1.3 HCSP File

A Hypertext Content Server Page (HCSP) file is a published web page that displays actual website content.

- HCSP files are typically created either by generating the web page through Content Publisher using an HCST page as a template, or by submittal of a form in Content Server through an HCSF page.
- Because this type of page contains web-viewable content, HCSP files are indexed in Content Server.

6.1.1.4 HCSF File

A Hypertext Content Server Form (HCSF) file is similar to an HCSP file, except that it contains HTML form fields that can be filled out and submitted from a web browser.

- When a user fills out and submits a form from an HCSF page, an HCSP file is checked in as a separate content item with metadata defined by XML elements for the HCSF page.
- Because this type of page contains web-viewable content, HCSF files are indexed in Content Server.

  For more information about HCSF pages, see Section 6.1.1.4, "HCSF File."

6.2 Altering the Appearance and Navigation of Web Pages

Although dynamic server pages are implemented in Content Server differently than custom components, you must be familiar with WebCenter Content component architecture concepts, particularly Content Server templates and HTML includes. For more information, see Chapter 11, "Getting Started with Content Server Components."

Use the following basic procedure to customize your Content Server instance with dynamic server pages:

1. Create an IDOC file with custom includes.
2. Check in the IDOC file to Content Server.
3. Create an HCST, HCSP, or HCSF file that references the IDOC file.
4. Check in the HCS* file to Content Server.
5. Display the HCS* file in your web browser by searching for it in Content Server or linking to it from a published web page.

Tip: Using dynamic server pages with Content Publisher can be a powerful tool for web publishing. For more information, see the Content Publisher documentation.

6.2.1 Syntax

Because the different types of dynamic server pages are interpreted and displayed differently, the Idoc Script in the files must be coded differently. The following table summarizes these differences.

File Type	.idoc	.hcst	.hcsp	.hcsf
Full Text Indexed?	No	No	Yes	Yes
Idoc Script Expressions	<$ … $>	<$ … $>	<!--$ … --> [!--$ … --]	<!--$ … --> [!--$ … --]
Comparison Operators	Symbols (==)	Symbols (==)	Special operators (eq)	Special operators (eq)
Special Characters	Symbols **(&)**	Symbols **(&)**	Escape sequence (&)	Escape sequence (&)
Referencing Metadata	Required	Required	Required	Required

Notes: Idoc Script uses standard HTML include coding. For more information, see Section 17.2.1, "HTML Includes."

HCST uses standard Content Server template coding. For more information, see Section 17.2.8.1, "Template and Report Pages."

Special coding is used with HCSP and HCSF to allow the page to be rendered both statically and dynamically, and full-text indexed.

6.2.1.1 Idoc Script Expressions

For HCSP and HCSF pages, Idoc Script expressions are generally placed between HTML comment tags. When a page is viewed statically, this placement enables a web browser to present the page content while ignoring any dynamic code that is used to format the content. This also enables the full-text indexing engine to successfully index the contents of these pages.

Some examples follow.

- IDOC or HCST file: `<$include MyIdocExpression$>`
- HCSP or HCSF file: `<!--$include MyIdocExpression-->`

In some situations, you might want to control the opening and closing of the HTML comment. In HCSP and HCSF files, this can be done by substituting other characters for the dash (-) in the closing tag after an Idoc Script expression, as Example 6–1 shows.

Example 6–1 Pound Sign Delimiter for HTML Comment in HCSP or HCSF File

```
<!--$a="ab"##> HTML comment remains open
<a href="<!--$myUrlAsVariable##>">MyUrl</a> Static view does not see this
<!--$dummy=""--> <!—Ended the comment area-->.
```

In Example 6–1, the pound sign (#) is substituted for the dash (-).

Another option in HCSP and HCSF files is to substitute brackets ([]) for the opening and closing tags (< >) of the standard HTML comment, as Example 6–2 shows. This substitution enables an XHTML parser to properly identify all the script when viewed statically.

Example 6–2 Bracket Delimiters for HTML Comment in HCSP or HCSF File

```
<!--$a="ab"--] HTML comment remains open
<a href="[!--$myUrlAsVariable--]">MyUrl</a> Static view does not see this
[!--$dummy=""--> <!—Ended the comment area-->.
```

6.2.1.2 Comparison Operators

For HCSP and HCSF pages, the standard comparison operators (such as ==) cannot be used because of their special meaning to HTML parsers. Use the following comparison operators in dynamic server pages.

IDOC or HCST File	HCSP or HCSF File	Description
==	eq	Tests for equality.
!=	ne	Tests for inequality.
<	lt	Tests if less than.
>	gt	Test if greater than.
<=	le	Tests if less or equal than.
>=	ge	Tests if greater or equal than.

For example, the following code evaluates whether or not the value of the variable `count` is greater than 10.

IDOC or HCST File	HCSP or HCSF File

`<$if count > 10$>`
`  <$"Count is greater than"$>`
`<$endif$>`
	`<!--$if count gt 10-->`
	`  <!--$"Count is greater than"-->`
	`<!--$endif-->`

6.2.1.3 Special Characters

For HCSP and HCSF pages, special characters such as the ampersand (&) cannot used because of their special meaning to HTML parsers. You must use the standard HTML or XML escape format (such as `&` or `&#038;`).

> **Note:** It is especially important to use the `&` escape character when you call the *docLoadResourceIncludes* function from an HCSP or HCSF page. For more information, see Section 6.2.2.1, "docLoadResourceIncludes Function."

As the following examples show, in Idoc Script, a quotation mark can be included in a string by preceding it with a backslash escape character, but in an HCSP or HCSF page, the quotation mark character must be represented by an HTML escape character.

- IDOC or HCST file: `"Enter \"None\" in this field."`
- HCSP or HCSF file: `"Enter &quot;None&quot; in this field."`

In an HCST page, a line feed is inserted using `\n`. In an HCSP page, insert the line feed directly in the file or encode it in the XML using the numeric ASCII number for a line feed.

> **Note:** You can now substitute the word `join` for the & string join operator. For example, you can write **[!--$a join b--]** instead of **[!--$a & b--]**. The first is accepted by an XML parser inside an attribute of a element, but the second is not.

6.2.1.4 Referencing Metadata

For dynamic server pages, several metadata values are stored with a `ref:` prefix, which makes them available to the page but does not replace `ResultSet` values. (This prevents *pollution* of ResultSets by dynamic server pages.)

When you reference any of the following metadata values on a dynamic server page, you must include the `ref:` prefix:

- `hasDocInfo`
- `dDocName`
- `dExtension`
- `dSecurityGroup`
- `isLatestRevision`
- `dDocType`

For example, the following statement determines if the document type is `Page`:

```
<$if strEquals(ref:dDocType,"Page"))$>
```

6.2.2 Idoc Script Functions

Two special Idoc Script functions are required for dynamic server pages:

- `docLoadResourceIncludes`
- `executeService`

6.2.2.1 docLoadResourceIncludes Function

To be able to use the HTML includes in an IDOC file, an HCS* file must call the `docLoadResourceIncludes` function, as in the following examples. This function loads all the includes from the specified IDOC file for use in assembling the current page. Example 6–3 shows the format for calling this function from an HCST file.

Example 6–3 docLoadResourceIncludes Function Call in HCST File

```
<$docLoadResourceIncludes("dDocName=system_std_
page&RevisionSelectionMethod=Latest")$>
```

Example 6–4 shows the format for calling this function from an HCSP or HCSF file.

Example 6–4 docLoadResourceIncludes Function Call in HCSP or HCSF file

```
<!--$docLoadResourceIncludes("dDocName=system_std_
page&amp;RevisionSelectionMethod=Latest")-->
```

6.2.2.1.1 Requirements for Calling the docLoadResourceIncludes Function

- The native file for the specified content item must have the `.idoc` extension.
- The `docLoadResourceIncludes` call must be placed before the first include call in the HCS* file. It is recommended that you place this function within the `HEAD` section of the page.
- You must use the correct ampersand character when you call the `docLoadResourceIncludes` function from an HCS* page. For more information, see Section 6.2.1.3, "Special Characters."

6.2.2.1.2 Parameters Use the following parameters with the `docLoadResourceIncludes` function to specify which IDOC file to call.

- You must define either a `dDocName` or a `dID`; do not use both parameters together.
- If you define a `dDocName`, you must define `RevisionSelectionMethod` to be `Latest` or `LatestReleased`.
- If you define a `dID`, do not define a `RevisionSelectionMethod`, or define the `RevisionSelectionMethod` to be `Specific`.

Parameter	Description
`dDocName`	Specifies the **Content ID** value of the IDOC file. This parameter should always be present when the **Content ID** value is known. Error messages are based on the assumption that it is present, as are other features, such as forms.
`dID`	Specifies the unique ID number of a particular revision of the IDOC file.
`RevisionSelectionMethod`	Specifies which revision of the IDOC file to use: ■ **Latest**: The latest checked-in revision of the document is used (including revisions in a workflow). ■ **LatestReleased**: The latest released revision of the document is used. ■ **Specific**: Use only with `dID`.
`Rendition`	Specifies which rendition of the IDOC file to use: ■ **Primary**: The primary (native) file. This is the default value in effect if no `Rendition` value is specified. ■ **Web**: The web-viewable file. ■ **Alternate**: The alternate file.

6.2.2.2 executeService Function

The `executeService` function executes a Content Server service from within a dynamic server page. For example:

HCST file: `<$executeService("GET_SEARCH_RESULTS")$>`

HCSP or HCSF file: `<!--$executeService("GET_SEARCH_RESULTS")-->`

- Services that can be called with the `executeService` function must be *scriptable*, meaning that they do not require parameter input.
- Scriptable services have an access level of 32 or more. For more information, see Chapter 24, "Getting Started with Integrating WebCenter Content into Your Environment."
- For a list of standard Content Server services, see the *`IdcHomeDir`*`/resources/core/tables/std_services.htm` file.
- For more information about the `executeService` function, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.
- For more information about services, see Chapter 24, "Getting Started with Integrating WebCenter Content into Your Environment."

Performance Tip: Use services sparingly. Too many service calls on a page can affect performance and limit scalability.

6.2.3 Development Recommendations

The following recommendations to assist you in developing dynamic server pages include general guidelines and HCSF guidelines.

6.2.3.1 General Guidelines

The following recommendations apply to the development of all types of dynamic server pages:

- Keep templates as simple and free of code as possible. Strive to have only HTML includes in your templates, with all code and conditionals in an IDOC file. This is especially helpful for HCSF pages, where submitted forms also reflect changes made to the IDOC file.
- Whenever you are customizing an Oracle WebCenter Content Server instance, you should isolate your development efforts from your production system. Keep in mind that frequent revisions to dynamic server pages can result in a large number of obsolete content items. You should do as much work on a development system as possible before deploying to a production instance, and you may need to delete out-of-date pages regularly.
- When you develop a website using dynamic server pages, think of the development and contribution processes in terms of ownership:
  - **Structure**, including site design and navigation, is owned by the webmaster. When you use dynamic server pages, structure is contained in and controlled with includes that are defined in IDOC files.
  - **Content**, that is, the actual text of the web pages, is owned by the contributors. When you use dynamic server pages, content is contained primarily in HCSP files that make use of the includes in the IDOC files.
- Using dynamic server pages with Content Publisher can be a powerful tool for web publishing. You can create content using Word documents or HCSF pages, and then use Content Publisher to convert the documents to published HCSP files. You can also use the "include before" and the "include after" options in the SCP template to insert additional Idoc Script includes.
- If you publish dynamic server pages with Content Publisher, use the prefix option for easy identification of your documents.
- Use a consistent naming convention. For example, for "system" level includes, you could name your IDOC file `system_std_page`, and then name each include in that file with the prefix `system_`. This makes locating the includes easier.
- You may want to create a content type for each type of dynamic server page (such as `HCSF_templates` or `submitted_forms`).
- In accordance with good coding practices, you should always put comments in dynamic server pages to document your customizations.

6.2.3.2 HCSF Guidelines

The following recommendations apply specifically to the development of HCSF pages:

- When designing a form, consider how the template will be used:
  - Will this template change depending on the role of the user submitting the form?
  - Will the submitted content enter into a criteria workflow?

  - What default metadata values should be set?
  - Does the form contain ResultSets for multiple line entries?
- To see the form parameters as they are passed from the web browser to the web server, filtered through Oracle WebCenter Content Server, and then passed back to the web browser, change the `method` attribute in the include code from `POST` to `GET`:

```
<form name="<$formName$>" method="GET" action="<$HttpCgiPath$>">
```

- If you add a form field called DataScript to a form being submitted, then any Idoc Script for that value is evaluated by Oracle WebCenter Content Server when it processes the form.

6.2.4 HCSF Pages

In addition to following the standard formatting rules for Oracle WebCenter Content Server templates and HTML forms, HCSF pages require several special sections and tags that enable Oracle WebCenter Content Server to process them. These special sections appear in the following order in a typical HCSF file:

1. Load section
2. Data section
3. Form section

For an example of a complete HCSF page, see Section 6.1.1.4, "HCSF File."

6.2.4.1 Load Section

The load section at the beginning of an HCSF page declares the file as an HTML file, loads an IDOC file, and loads other information about the page. Example 6–5 shows a typical load section.

Example 6–5 Load Section for an HCSF Page

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<!--$docLoadResourceIncludes("dDocName=my_idoc_
page&amp;RevisionSelectionMethod=Latest")-->
<meta NAME="idctype" CONTENT="form; version=1.0">
<!--$defaultPageTitle="Department News Form"-->
<!--$include std_html_head_declarations-->
</head>
```

The load section has these items:

- HTML declaration
- `docLoadResourceIncludes` function
- `meta` element
- Variables and includes

6.2.4.1.1 HTML Declaration The HTML declaration identifies the file as an HTML file, with the following syntax:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
```

6.2.4.1.2 The docLoadResourceIncludes Function The docLoadResourceIncludes function loads all the includes from the specified IDOC file for use in assembling the current page. For more information, see Section 6.2.4.1.2, "The docLoadResourceIncludes Function."

6.2.4.1.3 Meta Element The `meta` element is used by Content Publisher to identify that this is a special type of page.

- This element is not required if the form is not being published through Content Publisher.
- The `meta` element must be placed inside the `HEAD` section of your HTML file.
- Use the following syntax for the `meta` tag:

```
<meta NAME="idctype" CONTENT="form; version=1.0">
```

6.2.4.1.4 Variables and Includes The `HEAD` section for an HCSF page can contain variable definitions and HTML includes as necessary. Example 6–6 shows lines in a `HEAD` section that define the default page title and load the `std_html_head_declarations` code.

Example 6–6 Variable Definition and Include in the HEAD Section for an HCSF Page

```
!--$defaultPageTitle="Department News Form"-->
<!--$include std_html_head_declarations-->
```

6.2.4.2 Data Section

The data section for an HCSF page contains rules and metadata information that is used to process the form. There is a close relationship between the information in the data section and the presentation of the page:

- Upon delivery of the HCSF page to the user, the information in the data section is parsed into a `DataBinder` object and merged into the Form Section.
- Upon form submittal, the information in the data section is merged with the request and written out again to the data section. For more information, see Chapter 11.1.3.3, "Data Binder," and Section 11.1.3.1.1, "Elements in HDA Files."

6.2.4.2.1 Data Section Structure The data section consists of XML elements that are placed between `idcbegindata` and `idcenddata` Idoc Script tags, as Example 6–7 shows.

Example 6–7 Data Section for an HCSF Page

```
<!--$idcbegindata-->
<idcformrules isFormFinished="0"/>
<model_number content="html">AB-123</model_number>
<revision>12</revision>
…
<!--$idcenddata-->
```

The following rules apply to the data section:

- The data section must be placed inside the <BODY> section of your HTML file, before the beginning of the form section.
- You can place Idoc Script variable definitions and includes before or after the data section, but not within it.

- Two types of XML elements are used in the data section:
  - The idcformrules Element
  - Metadata Elements
- You can also use the following types of formatting in the data section:
  - Nested Elements
  - Referencing XML Elements
  - Form Elements
  - ResultSets

6.2.4.2.2 The idcformrules Element The idcformrules element defines Content Server rules in the data section. This element requires one attribute, either `isFormFinished` or `resultsets`.

- **IsFormFinished Attribute**: The `isFormFinished` attribute indicates whether the form can be submitted again or not.
  - Use the following attribute value to specify that the form can be submitted again:

    ```
    <idcformrules isFormFinished="0"/>
    ```

  - Use the following attribute value to specify that the form cannot be submitted again:

    ```
    <idcformrules isFormFinished="1"/>
    ```

    This code results in a read-only form.
- **resultsets Attribute**: The `resultsets` attribute indicates which XML elements in the data section are interpreted as ResultSets.
  - This attribute specifies one or more XML tag names separated by commas. For example:

    ```
    <idcformrules resultsets="volume,chapter">
    ```

  - During delivery of an HCSF page to the user, the core Content Server reads the `resultsets` attribute and, if necessary, places empty ResultSets with the specified names into the `DataBinder` object so that they are available for merging.

    For more information about ResultSet formatting in the data section, see Section 6.2.4.2.7, "ResultSets."

6.2.4.2.3 Metadata Elements Metadata elements specify the metadata values that appear in form fields when a form is displayed in a browser. For example:

```
<model_number>AB-123</model_number>
```

Each metadata element can be assigned a `content` attribute that indicates which type of content the element contains. For example:

```
<model_number content="html">AB-123</model_number>
```

- The value of the `content` attribute can be either `html` or `text`: Text indicates that the content of the element should be interpreted strictly as text. HTML indicates that the content of the element should be interpreted as HTML code.

- If the `content` attribute is not specified for a metadata element, it defaults to `html`.
- Content Publisher ignores all other attributes except the `content` attribute.

6.2.4.2.4 Nested Elements If you are not publishing HCSF pages through Content Publisher, you can use nested XML elements (also called nodes) within the data section. Example 6–8 shows a `<section>` element nested in a `<chapter>` element.

Example 6–8 Nested XML Tags in a Data Section

```
<chapter title="Chapter 1">
This is the beginning of the chapter.
<section title="First Section">
This is the first section of the chapter.
</section>
</chapter>
```

Note: Nested XML elements are not allowed in Content Publisher.

6.2.4.2.5 Referencing XML Elements To refer to a nested XML element, start with the root-level element, and use an exclamation point (!) between element levels. For example:

```
chapter!section
```

To refer to the attribute of any element, use a colon (:) after the tag name. For example:

```
chapter!section:title
```

- If you reference an element in the data section, the element value can be merged back into the data section upon form submission only if one of the following statements are true:
  - The `root` element has already been referenced in the data area.
  - The `root` element is referenced in an `ExtraRootNodes` form element.
  - A prefix part of the tag is referenced as a ResultSet in the **`resultsets`** form element.
- Default values can be specified by applying the `:default` suffix to a tag path. Note that default elements might contain Idoc Script for further evaluation. Example 6–9 shows the format for specifying a default `dDocTitle` value.

Example 6–9 Specification of a Default Metadata Value

```
<input type=hidden name="dDocTitle:default" value="<$'MyTitle ' &
dateCurrent()$>">0
```

6.2.4.2.6 Form Elements ■The **ExtraRootNodes** form element enables you to add tags by creating an Idoc Script variable and then appending the tag names to it, rather than specifying the tags in the data section of the form. At the end of your form, you can substitute a string value in place of the ExtraRootNodes value to be merged back into the data section.

- The **resultsets** form element enables you to add a tag as a ResultSet, rather than specifying the ResultSet in the data section.

- Both the ExtraRootNodes and ResultSet form elements take a comma-delimited list of tags.

Example 6–10 shows form elements that add the `mychapters!chapter` element as a valid ResultSet if it is not already defined in the `idcformrules resultsets` attribute. They also add the root element `mychapters`, if necessary.

Example 6–10 Form Elements That Add Elements to a ResultSet

```
<input type=hidden name="resultsets" value="mychapters!chapter">
<input type=hidden name="ExtraRootNodes" value="mychapters">
```

6.2.4.2.7 ResultSets You can define a ResultSet using XML elements within the data section for an HCSF page, as follows:

- You must use the `resultsets` attribute of the `idcformrules` element to specify a ResultSet.
- The element names must be completely qualified, and the full reference path from the root node must be used.
- The columns in the ResultSet are the element content and the element attributes.

For information about limitations on repeating and nesting XML elements in a ResultSet, see Example 6–13 and Example 6–15.

Example 6–11 shows XML elements that define two ResultSets, named `volume` and `chapter`.

Example 6–11 XML Elements for Defining ResultSets in the Data Section for an HCSF Page

```
<idcformrules resultsets="volume,chapter">
<volume title="First Volume">
    Volume content here
</volume>
<chapter title="First Chapter">
    Chapter content here
</chapter>
```

This code evaluates to two ResultSets with two columns each. Example 6–12 shows these ResultSets.

Example 6–12 ResultSets Defined by XML Elements

```
@ResultSet volume
2
volume
volume:title
Volume content here
First Volume
@end
@ResultSet chapter
2
chapter
chapter:title
Chapter content here
First Chapter
@end
```

If you are not publishing HCSF pages through Content Publisher, you can use repeated elements for a ResultSet in the data section. Repeated elements are typically useful for looping over code to create the ResultSet.

The following rules apply to repeated elements for ResultSets:

- Repeated elements are not allowed unless they are part of a ResultSet.
- Repeated XML elements are not allowed in Content Publisher.

Example 6–13 shows how the `chapter` element is repeated for the `chapter` ResultSet.

Example 6–13 Repeated Elements for a ResultSet

```
<idcformrules resultsets="chapter">
<chapter title="First Chapter">
    Some content here
</chapter>
<chapter title="Second Chapter">
    More content here
</chapter>
```

This code evaluates to a ResultSet with two columns and two rows. Example 6–14 shows this ResultSet.

Example 6–14 ResultSet Created with Repeated Elements

```
@ResultSet chapter
2
chapter
chapter:title
Some content here
First Chapter
More content here
Second Chapter
@end
```

A ResultSet can have nested elements, but the nested elements cannot be repeated within a parent element, as Example 6–15 shows. In this example code, an additional `<section>` element would not be allowed within the first `<chapter>` element.

Example 6–15 Repeated Nested Elements for a ResultSet

```
<idcformrules resultsets="chapter">
<chapter title="First Chapter">
    Some content here
    <section title="First Section of First Chapter">
    Section content
    </section>
</chapter>
<chapter title="Second Chapter">
    More content here
</chapter>
```

This code evaluates into a ResultSet that has four columns and four rows, with the last two cells blank, as Example 6–16 shows.

Example 6–16 ResultSet Created with Repeated Nested Elements

```
@ResultSet chapter
4
chapter
chapter:title
chapter!section
chapter!section:title
Some content here
First Chapter
Section Content
First Section of First Chapter
More content here
Second Chapter


@end
```

The following guidelines apply to editing ResultSets:

- Updating a specific row in a ResultSet requires that you indicate the ResultSet row number in the request parameter. The pound sign character (#) is used by Content Server to indicate a specific row. If you do not specify a row with the # character, then a row is appended. If you specify a row # that does not yet exist, then enough empty rows are added to provide a row to be edited.

  Example 6–17 shows how to update the first row (row 0) of a ResultSet.

Example 6–17 Editing a Row in a ResultSet

```
<input type="text" name="comment#0"
    value="new comment">
<input type="text" name="comment!title#0"
    value="new title"
```

- Use the exclamation point character (!) to insert new fields into a ResultSet.

  For example, to insert author and title fields into the `comment` ResultSet, you could name the input fields `comment!author` and `comment!title`. If those fields are not already in the ResultSet, they would be added when the form is submitted.

- To delete a row from a ResultSet, empty all the field values so that they are blank, as Example 6–18 shows.

Example 6–18 Deleting the First Row from a ResultSet

```
<input type="hidden" name="comment#0" value="">
<input type="hidden" name="comment!title#0" value="">
<input type="hidden" name="comment!date#0" value="">
<input type="hidden" name="comment!author#0" value="">
```

  Another method for deleting rows from a ResultSet is to set the `DeleteRows` form element to a list of comma-delimited pairs of ResultSet names and row numbers. For example, to delete row 2 from the `comment` ResultSet and row 5 from the `book` ResultSet, the `DeleteRows` form element would be set to the following comma-delimited pairs:

  ```
  comment:2,book:5
  ```

6.2.4.3 Form Section

The form section contains the code for presentation of the HTML form elements and any other functionality that the page requires. The form properties, form fields, and form buttons are placed in an HTML table to control the formatting of the assembled web page.

For code examples, see Section 6.6.1, "Common Code for Forms."

6.2.4.3.1 Form Begin The form section begins two lines of Idoc Script, as Example 6–19 shows.

Example 6–19 Idoc Script to Begin Form Section

```
<!--$formName="HTMLForm"-->
<!--$include std_html_form_submit_start-->
```

The `std_html_form_submit_start` include in `the std_page.idoc` resource file contains code to create a standard HTML form using a `POST` method, set the `IdcService` to `SUBMIT_HTML_FORM`, and set the `dID` variable to the value of the current HCSF page. Example 6–20 shows this code.

Example 6–20 Standard HTML Form for an HCSF Page

```
<form name="<$formName$>" method="POST"action="<$HttpCgiPath$>">7
<input type=hidden name="IdcService"value="SUBMIT_HTML_FORM">
<input type=hidden name="dID" value="<$SourceID$>">
```

6.2.4.3.2 Form Properties The form table typically begins with property definitions that create the fields as form fields, allow the fields to be edited, and set the size of the field caption area. Example 6–21 shows these property definitions.

Example 6–21 Field Property Definitions for a Form Table

```
<!--$isFormSubmit=1,isEditMode=1-->
<!--$captionFieldWidth=200, captionEntryWidth=80-->
```

6.2.4.3.3 Form Fields A few lines of code are typically used to create each input field, as Example 6–22 shows.

Example 6–22 Code to Create an Input Field on a Form

```
<!--$eval("<$product_name:maxLength=250$>")-->
<!--$fieldName="model", fieldCaption="Model Number"-->
<!--$include std_display_field-->
```

> **Note:** Some fields might require additional code for proper display. For example, you might need to override the standard `std_memo_entry` include to increase the size of text areas. You can do this by defining a custom include in the IDOC file, as follows:
>
> ```
> <@dynamicalhtml std_memo_entry@>
> <textarea name="<$fieldName$>" rows=15 cols=50
> wrap=virtual><$fieldValue$></textarea>
> <@end@>
> ```

- `DataScript`: If you add a form field called DataScript to a form being submitted, then any Idoc Script for that value is evaluated by Content Server when it processes the form.

There are two tables (coming from the data island inside the HCSP form) with an entry in one table that references entries in the other table. Your goal is to change a value in a specific column and row in the second table when you update a row in the first table. To accomplish this value change, you can write JavaScript to set the DataScript value with Idoc Script, as Example 6–23 shows.

Example 6–23 Changing a Field Value in a Table When You Update a Row in Another Table

```
modifyRowAndColumn(row, column, value)
{
document.myform.DataScript = "<$setValue('#local', 'table2!'"+ column + "#'"+ row
+
"','" + value + "')$>";
}
```

Then, when you call the function with *column = "myColumn"* and `row="1"` and `value = "Test"` while submitting the update form, the resulting DataScript value before submit would be as follows:

```
DataScript.value = <$setValue('#local', 'table2!myColumn#1', 'Test')$>
```

The result would be the column `table2!myColumn` in row 1 of the table `table2` would be updated with the value `Test` after the form was submitted.

Another way of saying this is that the DataScript can allow arbitrary edits of other entries in the data island without having to actually create HTML form fields that reference their names.

6.2.4.3.4 Form Buttons Two lines of code are typically used to create the form submission and reset buttons. Example 6–24 shows these lines.

Example 6–24 Code for Creating Form Submission and Reset Buttons

```
<input type=submit name=Submit value=" Submit ">
<input type=reset name=Reset value="Reset">
```

6.2.4.3.5 Form End After all the form elements and default values have been defined, the form must end with a `</form>` tag.

6.3 Creating an IDOC File with Custom Includes for Dynamic Server Pages

Dynamic server pages can work together to modify Content Server behavior. Before you can create a dynamic server page, you need an IDOC file with custom includes for the page to reference.

To create an IDOC file with custom includes for dynamic server pages:

1. Create an IDOC file with a custom include, in the format that Example 6–25 shows.

Example 6–25 Custom Include

```
<@dynamichtml HelloWorld@>
<H1>Hello World/<H1>
<@end@>
```

In the example, the first include is named `HelloWorld`, and the second include defines one line of HTML code.

2. Save the file with the `.idoc` extension; for example, `helloworld.idoc`.
3. Check in the IDOC file to Content Server with a Content ID that you can reference from another file, such as `helloworld`.

The IDOC file is available to any HCS* pages that reference it.

6.4 Creating an HCST Page

You can create an HCST dynamic server page by referencing an IDOC file in an HCST file.

To create an HCST page:

1. Create an HCST file that references an include in an IDOC file, like the HCST file that Example 6–26 shows.

Example 6–26 HCST File Referencing Custom Include

```
<HTML>
<HEAD>
<$docLoadResourceIncludes("dDocName=helloworld&RevisionSelectionMethod=LatestReleased")$>
</HEAD>
<BODY>
You should see it:
<$include HelloWorld$>
</BODY>
</HTML>
```

In the example, the line after the `<HEAD>` tag loads the `helloworld.idoc` file so that the includes in the IDOC file are available to this HCST page. The second line after the `<BODY>` tag displays the code from the `HelloWorld` include from the `helloworld.idoc` file. Note the use of the standard Idoc Script tags, `<$...$>`.

2. Save the file with the `.hcst` extension; for example, `helloworld.hcst`.
3. Check in the HCST file to Content Server.

6.5 Creating an HCSP Page

You can create an HCSP dynamic server page by referencing an IDOC file in an HCSP file.

To create an HCSP Page

1. Create an HCSP file that references an include in an IDOC file, like the HCSP file that Example 6–27 shows.

Example 6–27 HCSP File Referencing Custom Include

```
<HTML>
<HEAD>
<!--$docLoadResourceIncludes("dDocName=helloworld&amp;RevisionSelectionMethod=LatestReleased")-->
</HEAD>
<BODY>
You should see it:

<!--$include HelloWorld-->
```

```
</BODY>
</HTML>
```

In the example, the line after the <HEAD> tag loads the helloworld.idoc file so that the includes in the IDOC file are available to this HCSP page. The second line after the <BODY> tag displays the code from the HelloWorld include from the helloworld.idoc file. Note the use of the HTML comment tags, <!--...-->.

2. Save the file with the .hcsp extension; for example, helloworld.hcsp.
3. Check in the HCSP file to Content Server.

6.6 Creating an HCSF Page

A typical HCSF page and its associated IDOC file are shown in Example 6–28. This example creates a form that users can fill out and submit to enter product descriptions as content items.

To create an HCSF page:

1. Create an HCSF file that references an IDOC file named form_std_page, as Example 6–28 shows.

Example 6–28 Product Description Form in HCSF File

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<!--$docLoadResourceIncludes("dDocName=form_std_page&amp;
      RevisionSelectionMethod=Latest")-->

<head>
<meta NAME="idctype' CONTENT="form; version=1.0">

<!--$include form_head_section-->

</head>

<!--$include form_pre_xml_section-->

<!--$idcbegindata-->
      <idcformrules isFormFinished="0"/>
      <product_name content="html">

      </product_name?
      <model_number content="html">
            SC-
      </model_number>
      <summary_description>
            Enter a summary here.
      </summary_description>
      <full_description>
             Enter a full description here.
      </full_description>
      <author>
      <division>
            Household Products
      </division>
      <revision>

      </revision>
```

```
<!--$idcenddata-->

<!--$include form_post_xml_section-->

</body>
</html>
```

In the example, the line after the `html` tag loads the IDOC file with the Content ID of `form_std_page` so that the includes in the IDOC file are available to this HCSF page.

The `meta` tag is required for Content Publisher to acknowledge that this is a Content Server HTML form.

The two includes after the `meta` tag, defined in the `form_std_page` IDOC file, generate the code at the beginning of the web page.

The `isFormFinished` attribute of the `idcformrules` tag tells Content Server that the form is not finished, so the fields can be edited, and the form can be submitted.

The `content` property does not have to be set for each tag; it defaults to `html`.

The `idcbegindata` and `idcenddata` tags define the XML tagged area, which specifies rules and initial metadata values for the form.

The text in each set of XML tags will populate the corresponding field on the form.

The last include in the example, defined in the `form_std_page` IDOC file, generates the code at the end of the web page.

2. Save the file as `product_form.hcsf`.
3. Check in the HCSF file to Content Server.
4. Create an IDOC file with custom includes, as Example 6–29 shows.

Example 6–29 IDOC File with Custom Includes

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<body>

<@dynamichtml form_head_section@>
<!--standard includes for a standard hcsp page-->
<$defaultPageTitle="Product Description Form"$>
$include std_html_head_declarations$>
<@end@>

<@dynamichtml form_pre_xml_section@>
<!--This code is here for static viewing.-->
<$if 0$>
      <body>
<$endif$>

<$include body_def$>
<@end@>

<@dynamichtml form_post_xml_section@>

<$include std_page_begin$>
<$include std_header$>
```

```
<$formName="HTMLForm"$>
<$include std_html_form_submit_start$>

<table>

<$if strEquals(ref:dExtension,"hcsf"))$>
      <$isHcsf=1$>
<$else$>
      <$isHcsp=1$>
<$endif$>

<$if isHcsf$>
      <$isFormSubmit=1,isEditMode=1$>
<$endif$>

<$captionFieldWidth=150, captionEntryWidth=200$>

<$eval("<$product_name:maxLength=250$>")$>
<$fieldName="product_name", fieldCaption="Product Name"$>
<$if isHcsp$><isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$model_number:maxLngth=250$>")$>
<$fieldName="model_number", fieldCaption="Model Number"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$fieldName="summary_description",
      fieldCaption="Summary Description",
      fieldType="Memo"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<fieldName="full_descripton",
      fieldCaption="Full Description",
      fieldType="Memo"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$author:maxLength=250$>")$>
<$fieldName="author", fieldCaption="Author"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$division:maxLength=250$>")$>
<$fieldName="division", fieldCaption="Division"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$revision:maxLength=250$>")$>
<$fieldName="revision", fieldCaption="Revision"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<tr>
      <td colspand=2><hr></td>
</tr>
<tr>align=center>
      <td colspan=2>
      <$if isHcsf$>
```

```
            <input type=submit name=Submit value=" Submit ">
            <input type=reset name=Reset Value="Reset">
      <$endif$>
            <input type=hidden name="dDocTitle:default"
            value="<$"Product Description  " & dateCurrent()$>">
      </td>
</tr>
</table>
</form>
<$include std_page_end$>

<@end@>

</body>
</html>
```

In the example, the `form_head_section` include defines the page title and the code for the standard HTML head section (referencing the `std_html_head_declarations` include in the `std_page.htm` resource file).

The `form_pre_xml_section` include allows the page to be viewed statically and defines code for a standard Content Server web page (referencing the `body_def` include in the `std_page.htm` resource file).

The `form_post_xml_section` include defines the form fields. The `std_page_begin` and `std_header` includes, which are defined in the `std_page.htm` resource file, define code for a standard Content Server web page. The two lines after these includes define the form name and the code for a standard HTML form (referencing the `std_htm_form_submit_start` include in the `std_page.htm` file).

The conditional after the `table` tag determines if this is an editable form or a page that has already been submitted, based on the file-name extension. If this is an editable page (`isHcsf=1`), the next conditional sets variables that create the fields as form fields and allow the fields to be edited. The line after the conditionals sets the width of the table cells for field captions to 150 pixels and sets the width of the table cells for input fields to 200 pixels.

The `eval` function sets the maximum length of a text field to 250 characters.

The `fieldName` tag defines the name, caption, and type of field. If `fieldType` is not defined, it defaults to `Text`.

If this is a form that has already been submitted (`isHcsp=1`), the `if isHcsp` conditional sets a variable that makes the form field read-only.

The `std_display_field` include, defined in the `std_page.htm` resource file, defines code that creates the form field.

If this is an editable form (`isHcsf=1`), the `if isHcsf` conditional creates the **Submit** and **Reset** buttons.

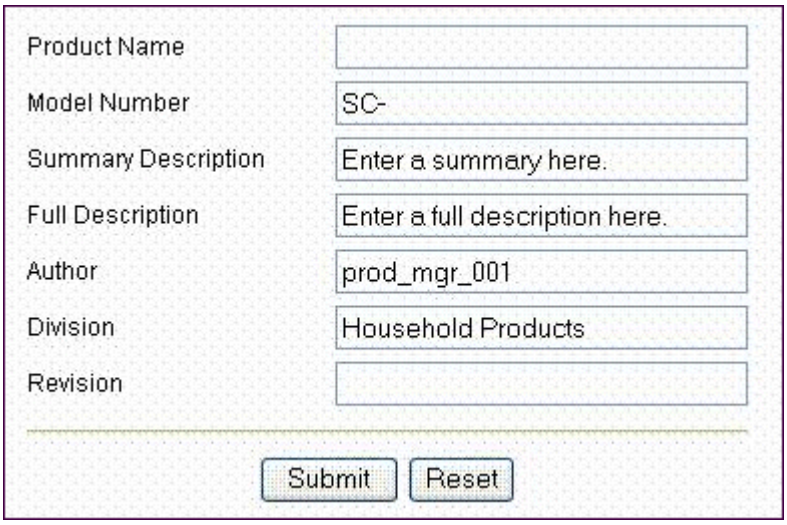
The line after the `if isHcsf` conditional generates the document title (`dDocTitle`) automatically.

The `std_page_end` include, defined in the `std_page.htm` resource file, generates the code at the end of the web page.

5. Save the file as `form_std_page.idoc`.
6. Check in the IDOC file to Content Server with a Content ID of `form_std_page`. (This is the name that is referenced by the HCSF page.)

7. Search for the HCSF content item in Content Server.
8. Click the link to display the form to create an HCSF page in your web browser. The form should look like the sample in Figure 6–2.

Figure 6–2 Form to Create HCSF Page Displayed in a Web Browser



9. Fill out the form with some sample values, and click **Submit**.

   A content item is created as an HCSP page.
10. Search for the HCSP page in Content Server.
11. Click the link to display the HCSP page in your web browser. Figure 6–3 shows how the page should look.

Figure 6–3 HCSP Page

Product Name:	Super Cleaner
Model Number:	SC-1
Summary Description:	This product cleans everything!
Full Description:	You can use Super Cleaner in the kitchen, bath, laundry, garage--anywhere there's dirt.
Author:	ProdMgr67
Division:	Household Products
Revision:	New

6.6.1 Common Code for Forms

The following features are commonly used in HCSF pages and associated IDOC files.

- Retrieving file information
- Referencing a file extension
- Defining form information
- Defining form fields

- Defining hidden fields
- Submitting a form

6.6.1.1 Retrieving File Information

Executing the service DOC_INFO_SIMPLE makes metadata from a specific file available to the page. Example 6–30 shows the Idoc Script to execute this service.

Example 6–30 Idoc Script to Retrieve Metadata

```
<$dID=SourceID$>
<$executeService("DOC_INFO_SIMPLE")$>
```

6.6.1.2 Referencing a File Extension

You can reference a file extension in an `if` statement for a form to determine whether the form has been submitted (`.hcsp` file) or unsubmitted (`.hcsf` file), as Example 6–31 shows.

Example 6–31 Statement to Reference a File Extension

```
<$if (strEquals(ref:dExtension,"hcsf"))$>
    <$isHcsf=1$>
<$else$>
    <$isHcsp=1$>
<$endif$>
```

For information about the *ref:* prefix, see Section 6.2.1.4, "Referencing Metadata."

6.6.1.3 Defining Form Information

Two lines of code define the form name and the standard include to start an HTML form, as Example 6–32 shows.

Example 6–32 Name and Standard Include for an HTML Form

```
<$formName="HTMLForm"$>
<$include std_html_form_submit_start$>
```

Example 6–33 shows typical code to define form properties.

Example 6–33 Form Properties

```
<table border=0 width=100%>
<$isEditMode=1,isFormSubmit=1$>
<$captionFieldWidth="25%", captionEntryWidth="75%"$>
```

6.6.1.4 Defining Form Fields

Use standard Idoc Script variables and the std_display_field include to display the form fields, as Example 6–34 shows.

Example 6–34 Standard Idoc Script to Display Form Fields

```
<$fieldName="news_
author",fieldDefault=dUser,fieldCaption="Author",isRequired=1,requiredMsg =
"Please specify the author."$>
<$include std_display_field$>
```

Some fields might require extra code to display the field correctly. For instance, the standard text area for a memo field is 3 rows by 40 columns, but you might need to override the standard include to increase the size of the text area. Example 6–35 shows the standard `std_memo_entry` include.

Example 6–35 Standard Include for a Memo Field

```
<@dynamichtml std_memo_entry@>
    <textarea name="<$fieldName$>" rows=3 cols=40 wrap=virtual> <$fieldValue$></textarea>
<@end@>
```

Example 6–36 shows how to use a custom `std_memo_entry` include to increase the text area to a specified size, in this case 15 rows by 50 columns.

Example 6–36 Custom Include for a Memo Field

```
<@dynamichtml std_memo_entry@>
    <textarea name=<$fieldName$> rows=15 cols=50 wrap=virtual><$fieldValue$></textarea>
<@end@>
```

6.6.1.5 Defining Hidden Fields

You can specify metadata for a submitted form (HCSP) by defining a hidden field, which contributors cannot change. For example, the following code assigns the document type `News_Forms` to each submitted form:

```
<input type=hidden name="dDocType" value="News_Forms">
```

This code specifies the security group of the submitted forms:

```
<input type=hidden name="dSecurityGroup" value="Public">
```

6.6.1.6 Submitting a Form

When a form is submitted, you may want to call a Java function to perform additional validation or processing. For example:

```
<input type=button name=Submit value="Save" onClick="postCheckIn(this.form)">
```

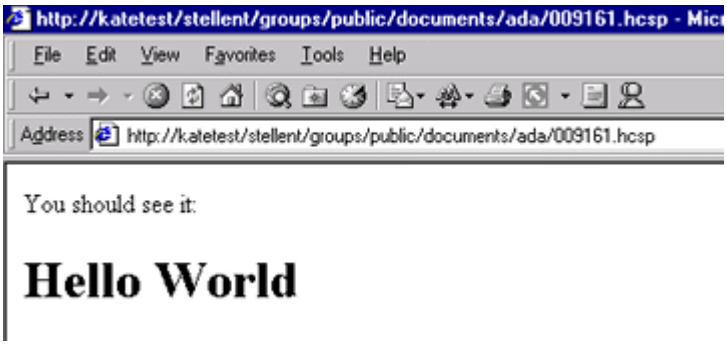
6.7 Verifying the Display of an HCST, HCSP, or HCSF Page in a Web Browser

After you save an HCST, HCSP, or HCSF file, you can verify the page display in a Web Browser, as Example 6–37 shows.

Example 6–37 Displaying an HCST Page and HCSP Page

1. Search for the `helloworld` content items in Content Server.
2. Display the HCST file and HCSP files in your web browser. They should both look like the example in Figure 6–4.

Figure 6–4 HelloWorld Content Item Displayed in a Web Browser



Part IV

Modifying the Functionality of Content Server

This part describes how to change the basic functionality of Oracle WebCenter Content Server.

Part IV contains the following chapters:

7

Changing System Settings

This chapter describes how to change the basic functionality of Oracle WebCenter Content Server.

This chapter includes the following sections:

The instructions in this chapter are for changing system settings on Oracle WebLogic Server. Oracle WebCentr Content can also be deployed to an IBM WebSphere Application Server. For more information, see the *Oracle Fusion Middleware Third-Party Application Server Guide*.

7.1 About Changing System Settings

Content Server has a number of features that you can set up to change features systemwide according to your needs. For example, you can use the following administration tools within Content Server to customize your content management system settings:

- Admin Server
- System Properties utility
- Web Layout Editor
- User Admin application
- Other administration customizations

  In addition to changing system setting with these tools, you can change other settings in different ways to meet the needs of your site:

  – Workflows can be designed, customized, and implemented using the Workflow Admin tool available from the Admin Applets menu

- New custom metadata fields can be created and default values set using the Configuration Manager
- Customized action screens (such as check-in, search, and check-out) can be created using Content Profiles

7.2 Changing System Settings Through the Admin Server

The Admin Server is a collection of web pages that you can use to configure systemwide settings for Oracle WebCenter Content Server. To access these web pages, choose **Admin Server** from the **Administration** tray in the portal navigation bar, which displays the Admin Server main page. From this page, you can check the status of each server that is running, and you can check console output.

For more information about changing system settings through the Admin Server, see "Configuring System Properties" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

7.3 Changing System Settings Through the System Properties Application

The System Properties administration application is used to configure systemwide Oracle WebCenter Content settings for content security, Internet settings, localization, and other types of settings. In the System Properties application, you can set these options:

- Optional functionality for Content Server
- Options related to content item security
- Options related to the Internet and web interaction
- JDBC connectivity options
- Functionality such as time zones and IP filters
- Localization features
- Directory paths

The application server is the primary tool for setting system properties for Oracle WebCenter Content; however, for some purposes you must use the System Properties application. You do not need administrative-level permissions to set these options; just access to the directory where the instance is installed.

For more information about changing system settings through the System Properties application, see "Running Administration Applications in Standalone Mode" in the *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

7.4 Customizing the Library and System Home Page with the Web Layout Editor

The Web Layout Editor is used to customize the Library and system home (portal) page. To access this editor, click **Web Layout Editor** on the Admin Applets page. With the Web Layout Editor, you can change the organization of local web pages in the Library and build new portal pages for your site. You can create links to websites outside your local site.

For more information, see Appendix B, "Building a Website."

7.5 Defining Security and Accounts for Users with the User Admin Application

You can define security groups, aliases, roles, and accounts for the users at your site using the User Admin function. To access this screen, choose **Admin Applets** from the **Administration** tray or menu, then click **User Admin** on the Administration Applets for *user* page. Options on this screen are used to create aliases, set permissions for security groups, establish roles and permissions associated with those roles, and customize information that is stored about users.

For more information, see "Managing Logins and Aliases" in the *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

8 Changing Configuration Information

This chapter describes how to change Oracle WebCenter Content Server configurations with the Idoc Script Custom Scripting Language and with other development tools and technologies.

This chapter includes the following sections:

8.1 About Changing Configuration Information

You can change configuration information with Idoc Script, a proprietary, server-side custom scripting language for Content Server. With Idoc Script, you can reference variables, conditionally include content in HTML pages, and loop over results returned from queries. Because Idoc Script is evaluated on the server side (rather than the client side), page elements are processed after the browser has made a request, but before the requested page is returned to the client.

For advanced customizations and integration with other business systems, you can use other development tools and technologies that Content Server supports.

8.2 Changing Configurations with the Idoc Script Custom Scripting Language

Idoc Script is primarily used in the following situations:

- For **include** code, an include defines pieces of code used to build Oracle WebCenter Content Server web pages. They are defined once in a resource file then referenced by template files as necessary. Includes are used on almost every page of the Oracle WebCenter Content Server website.

  A **super** tag can also be used, which defines exceptions to an existing include. The super tag tells the include to start with an existing include and add to it or modify it using the specified code.

- For **variables**, you can use variables to customize the Oracle WebCenter Content Server behavior. Variable values can be stored in an environment resource, such as the config.cfg file and many are predefined in Oracle WebCenter Content Server. You can also define your own custom variables.

- For **functions**, many built-in global functions are used in Oracle WebCenter Content Server. These perform actions such as date formatting or string comparisons. Some functions return results and some are used for personalization functions, such as those found on the My Profile page.
- For **conditionals**, you can use conditionals to test code and include or exclude the code from an assembled web page.
- For **looping**, two types of looping are available using Idoc Script: **ResultSet looping**, in which a set of code is repeated for each row in a ResultSet that is returned from a query and **while looping**, which is a conditional loop.
- In **Administration** areas, such as Workflow customization, web layouts, archiver and search expressions.

For information about usage, syntax, and configuration variables, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

8.3 Changing Configurations with Development Tools and Technologies

For advanced customizations and integration with other business systems, Content Server supports the following development tools and technologies:

- VBScript
- ASP
- J++
- JavaScript
- ASP+
- J2EE
- Java
- JSP
- COM
- Visual Basic
- DreamWeaver
- .Net
- C++
- Visual InterDev

9 Customizing Services

This chapter describes how to customize Oracle WebCenter Content Server services.

This chapter includes the following sections:

9.1 About Customizing Services

Content Server services are functions or procedures performed by Content Server. Calling a Content Server service (making a service request) is the only way to communicate with Content Server or to access the database.

Any service can be called externally (from outside Content Server) or internally (from within Content Server). Client services are usually called externally while administrative services are called internally. The service uses its own attributes and actions to execute the request, based on any parameters passed to the service.

The standard Content Server services are defined in the StandardServices table in *DomainHome*`/resources/core/tables/std_services.htm`. A service definition contains three main elements:

- The service **name**.
- The service **attributes**, which define the following aspects of the service:
  - **service class**, which specifies which Java class the service has access to. This determines what actions can be performed by the service.
  - **access level**, which assigns a user permission level to the service.
  - **template page**, which specifies the template that displays the results of the service
  - **service type**, which specifies if the service is to be executed as a subservice inside another service
  - **subjects notified**, which specifies the subsystems to be notified by the service.
  - **error message**, which is returned by the service if no action error message overrides it

- The service **action**, which is a colon-separated list that defines the following aspects of the action:
  - Action type
  - Action name
  - Action parameters
  - Action control mask
  - Action error message

Understanding and using services is an integral part of creating components and customizing Content Server. For more information, see Chapter 24, "Getting Started with Integrating WebCenter Content into Your Environment."

9.2 Customizing Services for Communicating with Content Server

Clients use services to communicate with Content Server. A service call can be performed from either the client or server side, so services can be performed on behalf of the web browser client or within the system itself.

For more information about services, see Section 17.2.7, "Services."

For more information about customizing services, see "Customizing Services" in the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

9.3 Customizing Services for Accessing the Database

Clients use services to access the Oracle WebCenter Content database. Any program or HTML page can use services to request information from Content Server.

For more information about services, see Section 17.2.7, "Services."

For more information about customizing services, see "Customizing Services" in the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

10

Generating Actions Menus

This chapter describes how to generate **Actions** menus for Oracle WebCenter Content Server.

This chapter includes the following sections:

10.1 About Generating Actions Menus

In previous versions of Content Server, when a component writer wanted to create an HTML table like those used on the search results page, HTML code had to be copied and pasted. The information in the tables was mixed with the HTML, with no separation between data and display.

The same issue was true for action menus. Data and display for the tables and menus were tightly coupled, making it impossible to perform global changes to all tables in Content Server except for those changes done with CSS modifications. It was also difficult for components to target and modify specific aspects of both the tables and the menus.

To customize a page's action menu, a developer can override one of the following include files then modify the `PageMenusData` ResultSet. These includes are all defined in the *`DomainHome`*`/resources/core/resources/std_page.idoc` file:

- `custom_searchapi_result_menus_setup`
- `custom_docinfo_menus_setup`
- `custom_query_page_menus_setup`
- `custom_audit_info_menus_setup`

In addition, tables like the one used on the search results page can be created by setting up ResultSets of data then calling specific resource includes which use that data to display the page. ResultSets can also be used to create action menus like those found on the Workflow In Queue and Search Results pages.

The action menu and HTML table display frameworks allow developers to create quick and flexible web pages that match the look and feel of the rest of the system. They also allow component writers to easily extend, add to, and override any or all of the Headline View or Thumbnail View tables on the server, and any of the action menus.

10.2 Creating Display Tables

Different display tables are used for the search results page for each display type:

- Headline view
- Thumbnail view

One of the first steps in any table setup is to retrieve documents to display, as Example 10–1 shows.

Example 10–1 Code to Retrieve Documents

```
<$QueryText = "dDocAuthor <matches> `sysadmin`"$>
<$executeService("GET_SEARCH_RESULTS")$>
```

10.2.1 Headline View Tables

The following example shows how to create a Headline View table. The concepts discussed here are also used to create the other table types.

The initial step in this process is to create a ResultSet that describes the columns of the table, as Example 10–2 shows.

Example 10–2 ResultSet to Describe Table Columns

```
<$exec rsCreateResultSet("ColumnProperties",
  "id,width,headerLabel,rowAlign")$>

<$exec rsAppendNewRow("ColumnProperties")$>
<$ColumnProperties.id = "dDocName"$>
<$ColumnProperties.width = "150px"$>
<$ColumnProperties.headerLabel = lc("wwDocNameTag")$>
<$ColumnProperties.rowAlign = "center"$>

<$exec rsAppendNewRow("ColumnProperties")$>
<$ColumnProperties.id = "dDocTitle"$>
<$ColumnProperties.width = "auto"$>
<$ColumnProperties.headerLabel = lc("wwTitle")$>
<$ColumnProperties.rowAlign = "left"$>

<$exec rsAppendNewRow("ColumnProperties")$>
<$ColumnProperties.id = "actions"$>
<$ColumnProperties.width = "75px"$>
<$ColumnProperties.headerLabel = lc("wwActions")$>
<$ColumnProperties.rowAlign = "center"$>
```

A ResultSet called `ColumnProperties` is created. Each row in the table corresponds to a column on the table to be created. Each column can have several attributes associated with it. Some of the more common attributes are:

- `id`: This is a mandatory attribute. Each column in the table being created must have an ID associated with it. The ID is used later to determine what will be displayed in every row.
- `width`: The width of the column. This can be any CSS width declaration such as `100px`, `15em`, or `auto`, which causes the column to auto-size, filling as much of the table as possible.
- `headerLabel`: The text to be displayed in the header of this column.

- `rowAlign`: An indication of whether the contents should be left, right, or center aligned.
- `headerURL`: Used to link the column header text to a URL.

The next step is to specify what data will be displayed in each row of the table, as Example 10–3 shows.

Example 10–3 Data to Display in a ResultSet

```
<$exec rsCreateResultSet("RowData","dDocName,dDocTitle,actions")$>
<$exec rsAppendNewRow("RowData")$>
<$RowData.dDocName = "<$dDocName$>"$>
<$RowData.dDocTitle = "<$dDocTitle$>"$>
<$RowData.actions = "<$include doc_info_action_image$>"$>
```

The `ColumnProperties` ResultSet technically has a row for each column in the table, while in `RowData`, there is only one row. Data entered into this ResultSet is of the following form:

```
<$RowData.%COLUMN_ID% = "%IDOCSCRIPT%"$>
```

Each column in the `RowData` ResultSet refers to an actual column that will appear in the final table. Each column in this ResultSet has a corresponding "ID" in the `ColumnProperties` ResultSet declared earlier. An Idoc Script expression is assigned to each cell in this ResultSet. It will then be evaluated during the display of each row as it is written to the HTML document.

Next the resource include must be created to display each row in the table.

```
<$include create_slim_table_row_include$>
```

Calling this resource include creates the `slim_table_row_include` resource include. Instead of parsing and evaluating the `RowData` ResultSet for each row in the table, it is done once.

Use the following steps to set multiple row includes (for example, for a single table which displays different rows for different types of items):

1. Delete and re-create the `RowData` ResultSet.
2. Set `rowIncludeName` to the name of the resource include to create.
3. Include `create_slim_table_row_include` again.

Example 10–4 shows code that displays the table.

Example 10–4 Code to Display a Table

```
<$include slim_table_header$>
<$loop SearchResults$>
  <$include slim_table_row_include$>
<$endloop$>
<$include slim_table_footer$>
```

To make the table look like the table on the search results page, set the following value in the script:

```
<$UseRowHighlighting = true$>
```

One special customization with the Headline View table allows any component writer or administrator to easily override how the data in any column is presented. Example 10–5 shows a custom include that can be declared from within a component.

Example 10–5 Custom Include Declaration in a Component

```
<@dynamichtml slim_table_title@>
  <b><$dDocTitle$></b>
<@end@>
```

If `dDocTitle:slimTableCellInclude=slim_table_title` is added to the *`IntradocDir`*`/config/config.cfg` file or set from within a script, all Headline View tables with a column ID of dDocTitle are displayed using the defined custom include. This overrides the `RowData` for these columns.

10.2.2 Thumbnail View Tables

The table for the Thumbnail View is created differently. The `ColumnProperties` and `RowData` ResultSets are not constructed. Instead, the number of columns are set, and an Idoc Script include name is used to *paint* each cell, as Example 10–6 shows. This is less easy to customize and less data-driven than the other methods, but this type of table is also much less structured.

Example 10–6 Code for Cells in a Thumbnail View Table

```
<$numDamColumns = 4$>
<$damCellIncludeName = "my_sample_dam_cell"$>
<$include dam_table_header$>
<$loop SearchResults$>
  <$include dam_table_item$>
<$endloop$>
<$include dam_table_footer$>
```

10.3 Customizing Actions Menus

The first step in customization is to add the **Actions** menu icon to the `Actions` column. Example 10–7 incorporates an action menu into each row of the Headline View sample table used previously.

Example 10–7 Code to Incorporate Actions Menus in Rows

```
<$RowData.actions = "<$include action_popup_image$>" &
  " <$include doc_info_action_image$>"$>
```

This inserts the action image into the appropriate column. However, clicking it does nothing because the actual menu is not written to the HTML page. Example 10–8 shows code that creates the data to be used to construct this menu.

Example 10–8 Data to Construct an Actions Menu

```
<$exec rsCreateResultSet("PopupProps",
  "label,onClick,function,class,id,ifClause")$>

<$exec rsAppendNewRow("PopupProps")$>
<$PopupProps.label = lc("wwCheckOut")$>
<$PopupProps.function = "<$HttpCgiPath$>?IdcService=CHECKOUT" &
  "&dID=<$dID$>&dDocName=<$url(dDocName)$>" &
  "&dDocTitle=<$url(dDocTitle)$>"$>
<$PopupProps.class = "document"$>
<$PopupProps.id = "checkout"$>

<$exec rsAppendNewRow("PopupProps")$>
<$PopupProps.label = lc("wwGetNativeFile")$>
```

```
<$PopupProps.function = "<$HttpCgiPath$>?IdcService=GET_FILE" &
  "&dID=<$dID$>&dDocName=<$url(dDocName)$>" &
  "&allowInterrupt=1"$>
<$PopupProps.ifClause = "showNativeFileLink"$>
<$PopupProps.class = "document"$>
<$PopupProps.id = "getNativeFile"$>

<$exec rsAppendNewRow("PopupProps")$>
<$PopupProps.label = lc("wwTest")$>
<$PopupProps.function = "javascript:alert('<$js(dDocName)$>');"$>
<$PopupProps.ifClause = "showTestAction"$>
<$PopupProps.class = "debug"$>
<$PopupProps.id = "alertDocName"$>
```

This code creates a ResultSet called `PopupProps`, where each row corresponds to an action in the menu being created. Each action can have several attributes associated with it. Some of the more common attributes follow:

- `label`: A string displayed as the label for the action.
- `function`: The URL or JavaScript method to be associated with this action.
- `class`: A classification for this action. It can be something as simple as "search", "document", "workflow", or even the name of your component. It places the action into a group so that it can be quickly enabled or disabled with the rest of the actions within that same group.
- `id`: Another method of classification, much more specific than "class". This method should be unique to the application, and you can use it to hide certain actions from appearing within the menus.
- `ifClause`: An optional attribute evaluated every time that action is about to be written to the HTML document. If the clause evaluates to FALSE, the action is not displayed.
- `isDisabled`: If set to 1, the action is never displayed.
- `linkTarget`: Used to make this link open a page in a different window. This attribute points to any anchor tag target.

After the data is set, it can be used to create an Idoc Script resource that writes this **Actions** menu, as Example 10–9 shows.

Example 10–9 Resource to Write an Actions Menu

```
<$include create_action_popup_container_include$>
```

This resource works like `create_slim_table_row_include`. It constructs a new Idoc Script resource called `action_popup_container_include`. To rename it, you could set `<$actionPopupContainerIncludeName = new_include_name$>` in the script.

Example 10–10 shows code to have this include called for each row of the Headline View table.

Example 10–10 Code to Call an Include for Each Row of a Table

```
<$exec rsCreateResultSet("PopupData", "actions")$>
<$exec rsAppendNewRow("PopupData")$>
<$PopupData.actions="<$include action_popup_container_include$>"$>
```

This code creates a `PopupData` ResultSet similar to the `RowData` ResultSet. It is structured in the same way, and is used as a location to print the action menu containers which are hidden until a user clicks on the action image.

The table created now has **Acitons** menus, similar to those normally seen on the search results page whenever the appropriate image is clicked.

Editing these actions is done by adding and deleting rows from the `PopupProps` ResultSet or editing rows that already exist. In addition to this type of customization, actions can be hidden by setting the `disabledActionPopupClasses` and `disabledActionPopupIds` variables. These can be set in the `config/config.cfg` file or in the Idoc Script itself, as Example 10–11 shows.

Example 10–11 Code to Hide Items in an Actions Menu

```
<$disabledActionPopupClasses = "workflow,folders"$>
<$disabledActionPopupIds = "getNativeFile,alertDocName"$>
```

Setting these variables causes any actions whose class is either `workflow` or `folders`, or whose ID is `getNativeFile` or `alertDocName`, to always be hidden. Using these variables enable Oracle WebCenter Content Server administrators and component writers to hide specific actions either globally or for specific pages.

Component writers also can override a number of Idoc Script resource includes to modify functionality in this area on either a global or targeted scale. The following includes are just a few of the available resource includes:

- `custom_add_to_action_popup_data`
- `custom_modify_action_popup_data`
- `classic_table_row_pre_display`
- `slim_table_row_pre_display`
- `custom_row_pre_display`

Part V

Customizing Content Server with Components

This part describes how to work with Oracle WebCenter Content Server components, which are programs that modify Content Server functionality.

Part V contains the following chapters:

11

Getting Started with Content Server Components

This chapter describes how to work with Oracle WebCenter Content Server components, which are programs used to modify Content Server functionality.

This chapter includes the following sections:

11.1 About Standard, System, and Custom Components

Components are modular programs designed to interact with Content Server at runtime. **Standard components**, **system components**, and **custom components** are included with Content Server to add or change the standard core functionality of Content Server.

11.1.1 Component Files Overview

When you define a custom component, you create or make changes to the following files:

- The `idc`*`short-product-id`*`_components.hda` file, which tells Content Server what components are enabled and where to find the definition file for each component.
- The component definition (or *glue*) file, which tells Content Server where to find the resources for the custom component.
- Different custom resource files, which define your customization to standard Content Server resources.
- Template files, which define custom template pages.
- Other files which contain customization to Content Server graphics, Java code, help files, and so on.

For more detailed information about these files, see Section 11.1.3, "About Directories and Files."

Any type of file can be included in a component, but the following file formats are used most often:

- HDA
- HTM
- CFG
- Java CLASS

If you build or unpackage components in the Component Wizard, or upload and download components in the Component Manager, you work with the following files:

- A compressed ZIP file used to deploy a component on other Content Server instances.
- A `manifest.hda` file that tells Content Server where to place the files that are unpackaged or uploaded from a component ZIP file.

11.1.2 Using Components

Components are modular programs that are designed to interact with Oracle WebCenter Content Server at runtime. The **component architecture** model is derived from object-oriented technologies, and encourages the use of small modules to customize Oracle WebCenter Content Server as necessary, rather than creation of a huge, all-inclusive (but cumbersome) application.

Note: You can create custom components by manually creating the necessary files and resources. However, the Component Wizard has no limitations compared to the manual method, and using it prevents many common mistakes.

Any type of file can be included in a component, but the following file formats are used most often:

- HDA
- HTM
- CFG
- Java CLASS

Components are typically used to alter the core functionality of Oracle WebCenter Content Server. For example, you could use a component to perform any of these tasks:

- Modify the standard security features
- Change the way search results are requested and returned
- Enable Oracle WebCenter Content Server to work with a particular system (such as a Macintosh client or a proprietary CAD program)

Using component architecture with Oracle WebCenter Content Server gives you these advantages:

- You can modify source code without compromising the integrity of the product.

  Oracle WebCenter Content Server loads many of its resources from external text files, so you can view the files to analyze how the system works, and then copy and modify the files to your requirements.

- You can use a custom component on multiple instances across multiple platforms.

  When you have created a custom component, you can package it as a ZIP file and load it on other Oracle WebCenter Content Server instances. Many custom components can work on Oracle WebCenter Content Server platforms other than the original development platform.

- You can turn individual components on and off for troubleshooting purposes.

  You can group customizations so that each component customizes a specific Oracle WebCenter Content Server function or area. If you have problems, disabling components one at a time can help you quickly isolate the trouble.

- You can reinstall or upgrade an Oracle WebCenter Content Server instance without compromising customizations.

  Custom components override existing product resources rather than replace them. Replacing the standard Oracle WebCenter Content Server files might not affect your customizations.

Keep the following constraints in mind when deciding whether to use custom components:

- **Custom components change behavior and look-and-feel systemwide.** If you want your changes to apply only in limited situations, you might want to consider dynamic server pages.
- **Custom components can be affected by changes to the Oracle WebCenter Content Server core functionality.** Because new functionality may change the way your components behave, customizations are not guaranteed to work for future Oracle WebCenter Content Server releases. Whenever you upgrade, you should review and test your custom components.
- **A component may not be necessary for simple customizations.** A large number of simple components could become difficult to manage.

Components must be installed and enabled to be used by Oracle WebCenter Content Server. Components provided with Oracle WebCenter Content Server are automatically installed, and they are enabled or disabled by default. Custom components must be installed and enabled to be usable. Several tools are available for working with components:

- The Component Wizard automates the process of creating custom components. You can use the Component Wizard to create new components, modify existing components, and package components for use on other Oracle WebCenter Content Server instances. For more information, see Section 11.2.1, "Component Wizard."
- The Advanced Component Manager provides a way to manage custom components in Oracle WebCenter Content Server. By using the Advanced Component Manager, you can add new components and enable or disable components for Oracle WebCenter Content Server. For more information, see Section 11.2.2, "Advanced Component Manager."
- The ComponentTool is a command-line utility for installing, enabling, and disabling components for Oracle WebCenter Content Server.

For information about component architecture and creation, see Chapter 11, "Getting Started with Content Server Components."

11.1.3 About Directories and Files

The following files are used in component creation:

- HDA files
- Custom resource files
- Manifest file
- Other files, such as customized site files, the component ZIP file, and custom installation parameter files

In the typical directory structure for WebCenter Content, the files for a component are stored in a `component` directory, in the *`DomainHome`*`/ucm/`*`short-product-id`*`/custom/` directory.

Content Server uses a data binder to cache data, such as variable values and lookup keys.

11.1.3.1 HDA Files

A HyperData File (HDA) is used to define properties and tabular data in a simple, structured ASCII file format. It is a text file that is used by Content Server to determine which components are enabled and disabled and where to find the definition files for that component.

The HDA file format is useful for data that changes frequently because the compact size and simple format make data communication faster and easier for Content Server.

The HDA file type is used to define the following component files:

- Components file (`idc`*`short-product-id`*`_components.hda`)
- Component definition file
- Manifest file
- Dynamic table resource file
- Template resource file

Example 11–1 shows an `idccs_components.hda` file that points to a component called `customhelp`.

Example 11–1 idccs_components.hda File for a Component

```
<?hda charset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
@end
@ResultSet Components
2
name
location
customhelp
custom/customhelp/customhelp.hda
@end
```

11.1.3.1.1 Elements in HDA Files Each HDA file contains a *header line* and one or more *sections*. The header line identifies the Content Server version, character set, and Java encoding for the HDA file. If an HDA file contains double-byte (Asian language) characters, the correct character set and encoding must be specified so that Content

Server can read the file properly. The header line is not required for single-byte characters, but it is a good practice to include it in your HDA files.

Two types of sections, `Properties` and `ResultSet`, are relevant to component development. These sections are used to define the properties of the file (name, location, and so on) and the ResultSet, which defines a table or columns and rows of data. A ResultSet often represents the results of a query. All other sections tags are for internal application use only.

Comments are not allowed within a section of an HDA file. However, you can place comments in the HDA file before the first section, between sections, or after the last section. Blank lines within a section of an HDA file are interpreted as a NULL value. Blank lines before the first section, between sections, or after the last section are ignored. None of the section types are mandatory in an HDA file, so unused sections can be deleted.

- The `Properties` section contains a group of name/value pairs. For a custom component, the most common name for a `Properties` section is `LocalData`, which means that the name/value pairs are valid only for the current HDA file.

  You can also define global name/value pairs in a `Properties` section called `Environment`, but this section is rarely used. The recommended practice is to define global environment variables in a configuration file, such as `config.cfg`.

  Example 11–2 shows a `Properties` section from an HDA file.

Example 11–2 Properties Section of an HDA File

```
@Properties LocalData
PageLastChanged=952094472723
LocationInfo=Directory,Public,
IsJava=1
refreshSubMonikers=
PageUrl=/intradoc/groups/public/pages/index.htm
LastChanged=-1
TemplatePage=DIRECTORY_PAGE
IdcService=PAGE_HANDLER
LinkSelectedIndex=0
PageName=index
HeaderText=This is a sample page.  The Page Name must remain index.  The Page
Properties for this index page should be customized.
PageFunction=SavePage
dSecurityGroup=Public
restrictByGroup=1
PageType=Directory
PageTitle=Content Server Index Page
@end
```

- Each `ResultSet` section of an HDA file defines a table or columns and rows of data. A ResultSet can be used to pass information to a database or to represent the result of a database query. A `ResultSet` section has the following structure:
  - The first line defines the name of the ResultSet table, using the format `@ResultSet` *resultset_name*.
  - The second line defines the number of columns.
  - The next *n* lines define the column names.
  - The remaining lines define the values in each cell of the table.
  - The last line of the section ends the table, using the format `@end`.

Example 11–3 shows a ResultSet called `Scores` that has 4 columns and 3 rows.

Example 11–3 ResultSet Section of an HDA File

```
@ResultSet Scores
4
name
match1
match2
match3
Margaret
68
67
72
Sylvia
70
66
70
Barb
72
71
69
@end
```

The following table shows the ResultSet data in a columnar form. A ResultSet can be given any name.

name	match1	match2	match3
Margaret	68	67	72
Sylvia	70	66	70
Barb	72	71	69

Content Server uses some predefined ResultSets with the following names, which should not be used for the custom component table.

ResultSet Name	Location	Purpose
`Components`	*`IntradocDir`*`/data/components/idc`*`short-product-id`*`_components.hda`	Defines the name and location of any custom components you have created. You must specify the short product ID (`cs`, `ibr`, `urm`) for *`short-product-id`*.
`IntradocReports`	*`IdcHomeDir`*`/resources/core/reports/reports.hda`	Specifies the default report templates for Content Server.
`IntradocTemplates`	*`IdcHomeDir`*`/resources/core/templates/templates.hda`	Specifies all of the default templates for Content Server (except for search results and report templates).
`ResourceDefinition`	*`DomainHome`*`/ucm/`*`short-product-id`*`/custom/`*`component_name`*`/`*`component_name`*`.hda`	Defines resources for a custom component.

ResultSet Name	Location	Purpose
SearchResultTemplates	*IdcHomeDir*/resources/core/templates/templates.hda	Specifies the default search results templates for Content Server.

11.1.3.1.2 The idccs_components.hda, idcibr_components.hda, or idcurm_components.hda File The idc*short-product-id*_components.hda file is a text file that tells Content Server which components are enabled and where to find the definition file for each component.

The idc*short-product-id*_components.hda file is always stored in the *IntradocDir*/data/components/ directory. You can use Component Wizard, Component Manager, or ComponentTool to make changes to this file if needed.

> **Note:** As of release 11*g*R1, the components.hda file and edit_components.hda file have been combined into one file called idc*short-product-id*_components.hda. If the Admin Server does not find the idc*short-product-id*_components.hda file but does find the legacy files, then it will migrate the data from the legacy file and create an idc*short-product-id*_components.hda file containing the appropriate data.

Example 11–4 shows an idccs_components.hda file that lists several enabled components, such as schema, configuration migration, and SOAP components.

Example 11–4 idccs_components.hda File for Multiple Enabled Components

```
@properties LocalData
blDateFormat=M/d/yy
@end
@ResultSet Components
2
name
location
SchemaDCL
custom/SchemaDCL/SchemaDCL.hda
ConfigMigrationUtility
custom/ConfigMigrationUtility/Cmu.hda
Soap
custom/Soap/Soap.hda
@end
```

11.1.3.1.3 Component Definition Files A **component definition file** points to the custom resources that you have defined. This file specifies information about custom resources, ResultSets, and merge rules. Because it serves as the "glue" that holds a component together, the component definition file is sometimes called the **glue** file.

The definition file for a component is typically named *component_name*.hda, and it is located in the *DomainHome*/ucm/*short-product-id*/custom/*component_name*/ directory.

Note: Do not confuse the `idc`*`short-product-id`*`_components.hda` file with the *`component_name`*`.hda` file. The `idc`*`short-product-id`*`_components.hda` file is used to track all installed components. The *`component_name`*`.hda` file contains information that is specific to a single component.

11.1.3.2 Custom Resource Files

Custom resource files define your Content Server customization. They are usually HDA files but some are HTM files.

The custom resource files for a component are typically located in the *`DomainHome`*`/ucm/`*`short-product-id`*`/custom/`*`component_name`*`/` directory. Some resource files may be placed in subdirectories, such as `resources/core/templates/`.

Table 11–1 describes these resources.

Table 11–1 Custom Resource Files

Resource Type	File Type	Contents
HTML include	HTM	Definitions of includes
String	HTM	Localized string definitions
Dynamic table	HDA	Tables for data that changes often
Static table	HTM	Tables for data that seldom changes
Query	HTM	Tables that define queries
Service	HTM	Tables that define service scripts
Template	HDA	Tables that specify location and file name for template pages
Environment	CFG	Configuration variable name/value pairs

For more detailed information about these files, see Section 11.4, "Resources for Assembling Web Pages."

In addition, a `template.htm` page is used by Content Server to assemble web pages. For more detailed information about the `template.hdm` file, see Section 17.2.8, "Templates."

A ResultSet HTM table file is used by other resources. A ResultSet table in an HTM file is similar to the ResultSet of an HDA file, except that it uses HTML table tags to lay out the data. Static table resources, service resources, and query resources all use this table format.

A ResultSet table in an HTM file begins with `<@table` *`table_name`*`@>` and ends with `<@end@>`. The markup between the start and end tags is an HTML table. Unlike a ResultSet in an HDA file, the number of columns is implied by the table tags.

Any HTML syntax that does not define the data structure is ignored when the table is loaded. Therefore, HTML comments are allowed within tables in an HTM file, and HTML style attributes can be used to improve the presentation of the data in a web browser.

11.1.3.3 Data Binder

Content Server caches data (such as variable values and lookup keys) internally in a data binder. All data in the data binder is categorized according to where it came from

and how it was created. When a value is required to fulfill a service request, the data in the data binder is evaluated in the following default order:

1. LocalData
2. ResultSets
3. Environment

This precedence can be changed using Idoc Script functions. For more information, see Appendix A, "Idoc Script Functions and Variables."

11.1.3.3.1 LocalData The `@Properties LocalData` section in an HDA file maps to the `LocalData` category of the data binder. The `LocalData` information consists of name/value pairs.

`LocalData` information is maintained only during the lifetime of the Content Server request and response. Unlike information about the server environment, which rarely changes, the `LocalData` information for each request is dynamic.

From the point of view of an HTTP request, the initial `LocalData` information is collected from the `REQUEST_METHOD`, `CONTENT_LENGTH`, and `QUERY_STRING` HTTP environment variables. As the service request is processed, the `LocalData` name/value pairs can be added and changed.

11.1.3.3.2 ResultSets Each `@ResultSet` section of an HDA file maps to a named result in the `DataBinder` object. Some ResultSet can be made *active*, taking precedence over other ResultSets during a value search. A ResultSet becomes active when the ResultSet is looped on during page assembly. An active ResultSet take precedence over any other ResultSets during a value search of the `DataBinder` object. When a service request requires data and the value is not found in the LocalData or an active ResultSet, the remaining ResultSets (those that are not active) are searched next.

11.1.3.3.3 Environment Environment values are placed in the `DataBinder` object as name/value pairs, which are defined in configuration files such as *`IntradocDir`*`/config/config.cfg, intradoc.cfg`, and environment-type resource files.

11.1.3.4 Manifest File

Manifest files are used to upload or unpackage a component ZIP file on Content Server. This file tells Content Server where to place the individual files that are included in the component ZIP file. A manifest file is created automatically when you build a component in the Component Wizard, or when you download a component using the Admin Server Advanced Component Manager.

All manifest files must be called manifest.hda. The manifest.hda file is included in the component ZIP file along with the other component files. It must be at the top level of the ZIP file directory structure.

The `manifest.hda` file contains a `ResultSet` table called `Manifest`, which consists of two columns:

- The `entryType` column defines the type of entry in the manifest file.

Entry Type	Description	Default Path
`Classes`	Java class files	*`DomainHome`*`/ucm/`*`short-product-id`*`/classes/`
`Common`	Common files	*`DomainHome`*`/ucm/`*`short-product-id`*`/weblayout/common/`

Entry Type	Description	Default Path
`Component`	Component resource files	*`DomainHome`*`/ucm/`*`short-product-id`*`/custom/`
`ComponentExtra`	Associated files, such as a readme	*`DomainHome`*`/ucm/`*`short-product-id`*`/custom/`
`Help`	Online help files	*`DomainHome`*`/ucm/`*`short-product-id`*`/weblayout/help/`
`Images`	Graphics files	*`DomainHome`*`/ucm/`*`short-product-id`*`/weblayout/images/`
`Jsp`	JavaServer Pages	*`DomainHome`*`/ucm/`*`short-product-id`*`/weblayout/jsp/`

Caution: Avoid using the entry types `Common`, `Help`, `Images`, and `Jsp` because they are deprecated in WebCenter Content 11*g*. WebCenter Content has a publishing engine that pushes files into the `weblayout` directory from components. If you want the same behavior as in a previous release, use the publishing engine; otherwise, the publishing engine may place files directly into the `weblayout` directory from a custom component, overwriting existing files. The overwritten files could be permanently lost.

- The `location` column defines the directory where the files associated with the entry are installed and specifies the file name for some entry types.
  - For a `Component` entry type, the location is the path and file name for the definition file. The definition file then tells Content Server which resource files are included in the component.
  - For other entry types, the location can be a path without a file name (to specify all files in a particular subdirectory) or a path with a file name (to specify an individual file).
  - The location should be a path relative to the *`DomainHome`*`/ucm/`*`short-product-id`*`/custom/` directory. You can use an absolute path, but then the component can be installed only on Content Server instances with the same installation directory path.

Example 11–5 shows a `manifest.hda` file.

Example 11–5 manifest.hda File

```
@ResultSet Manifest
2
entryType
location
component
MyComponent/MyComponent.hda
componentExtra
MyComponent/readme.txt
images
MyComponent/
@end
```

11.1.3.5 Other Files

Your custom components can include any type of file that Content Server uses for functionality or to generate its look and feel.

11.1.3.5.1 Customized Site Files You can add customized files for your site to change the look or actions of Content Server. For example, the following types of files are often referenced in custom resources:

- Graphics

  Replace the icons, backgrounds, and logos that constitute the standard Content Server interface.

- Help

  With the assistance of Consulting Services, you can customize help files for your content management system.

- Classes

  Java code can change or extend the functionality of Content Server. Java class files must be packaged into directories for placement in the *DomainHome*/ucm/*short-product-id*/classes/ directory.

> **Caution:** Avoid placing Graphics and Help files in the `weblayout` directory manually because your files may be overwritten by the WebCenter Content 11*g* publishing engine, which pushes files into the `weblayout` directory from components. If you want the same behavior as in a previous release, use the publishing engine; otherwise, the publishing engine may place files in this directory directly from a custom component, overwriting existing files. The overwritten files could be permanently lost. If you need to place these files in the `weblayout` directory manually, contact Oracle Consulting Services.

11.1.3.5.2 Component ZIP File A component ZIP file contains all files that define a Content Server component. It can be unpackaged to deploy the component on other Content Server instances.

11.1.3.5.3 Custom Installation Parameter Files When you define one or more custom installation parameters, several additional files are created in addition to the files that compose the basic component file structure.

If installation parameters are created for the component, then during the component installation process the component installer automatically places two files in the directory for the component within the `data/components/` directory. These files hold the preference data as follows:

- The `config.cfg` file: Contains the parameters that can be reconfigured after installation.
- The `install.cfg` file: Contains the preference data definitions and prompt answers.
- Backup ZIP file: A backup file that is created if the component is currently installed and is being reinstalled.

11.1.3.6 Typical Directory Structure

If you use the Component Wizard to create custom components, your files are stored in the appropriate directory.

Different component directories are established for each custom component in the *DomainHome*/ucm/*short-product-id*/custom/ directory. Within each component directory, separate subdirectories are established for reports, templates, and resources, all named appropriately (for example, *component_name*/resources/). The *component_name*.hda file (the definition file) is stored in the *component_name* directory.

11.1.4 Development Recommendations

The following sections provide some guidelines to assist you in developing custom components:

- Section 11.1.4.1, "Creating a Component"
- Section 11.1.4.2, "Working with Component Files"
- Section 11.1.4.3, "Using a Development Content Server"
- Section 11.1.4.4, "Component File Organization"
- Section 11.1.4.5, "Naming Conventions"

For more detailed information about creating or modifying components, see *Oracle Fusion Middleware Administering Oracle WebCenter Content* or online help.

11.1.4.1 Creating a Component

If your site needs some functionality in Content Server that the existing components do not provide, you can create a custom component for your Content Server instance.

11.1.4.1.1 How to Create a Custom Component You can create a custom component in a definition file, then enable the component and apply it to Content Server.

To create and enable a custom component:

1. Create a definition file.
2. Add a reference to the definition file in the idc*short-product-id*_components.hda file to enable the component.
3. Restart Content Server to apply the component.
4. Create resources and other files to define your customization. A good approach is to copy, rename, and modify standard Content Server files to create your custom resource files.
5. Test and revise your customization as necessary. You may need to restart Content Server to apply your changes.
6. If you want to package the component for later use or for deployment on other Content Servers instances, build the component and create a component ZIP file.

11.1.4.2 Working with Component Files

Two tools are available for working with component files:

- Component Wizard

  The Component Wizard is a Content Server utility that can help you create and edit component files. You can also use the Component Wizard to package,

unpackage, enable, and disable components. For more information about using this utility, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

- Text editor

  Because most component files are plain text files, you can create and edit the files in your favorite text editor.

You should use the Component Wizard as much as possible when working with custom components.

The Component Wizard does several tasks for you and minimizes the amount of work you need to do in a text editor. Using the Component Wizard helps you follow the recommended file structure and naming conventions. The Component Wizard automatically adds a *readme* text file when you build a component, which helps you document your customization. You should also include comments within your component files.

For information about using the Component Wizard to create components, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

11.1.4.3 Using a Development Content Server

Whenever you are customizing Content Server, you should isolate your development efforts from your production system. Remember to include the same custom metadata fields on your development Content Server as you have defined for your production Content Server.

When you have successfully tested your modifications on a development Content Server, use the Component Wizard to build a component ZIP file and then unpackage the component on your production system.

Remember to restart Content Server after enabling or disabling a component.

If you are having problems with Content Server after you have installed a custom component, disable the component and restart Content Server. If this fixes the problem, you probably need to troubleshoot your component. If the problem is not fixed, you may need to remove the component completely, using the Component Wizard, to determine whether there is a problem with the component or with Content Server.

11.1.4.4 Component File Organization

To keep your custom components organized, follow these file structure guidelines. For more information, see Section 11.1.3.6, "Typical Directory Structure."

Note: If you use the Component Wizard, it creates component directories for you and places the component files in the correct directories.

Place each custom component in its own directory within a directory called *DomainHome*`/ucm/`*short-product-id*`/custom/`. If your custom component includes resource-type or template-type resources, or both, the `component` directory should have subdirectories that follow the structure of the *IdcHomeDir*`/data/resources/core/` directory:

- `resources/` to hold HTML include and table resource files
- `resources/lang/` to hold string resource files

- `templates/` to hold template files
- `reports/` to hold report files

When considering files and their organization, keep the following points in mind:

- Place the definition file for each custom component at the top level of the component's directory.
- When referring to other files within a component, use relative path names instead of absolute path names. Using relative path names enables you to move the component to a different location without having to edit all of the files in the component.
- Content Server is a Java-based application, so forward slashes must be used in all path names.
- Custom components do not have to be stored on the same computer as Content Server, but all component files must be accessible to your Content Server instance.
- Images and other objects that are referenced by Content Server web pages must reside somewhere in the *DomainHome*`/ucm/`*short-product-id*`/weblayout/` directory (so that the web server can access the objects).

11.1.4.5 Naming Conventions

To keep your component files organized and ensure that the files work properly in Content Server, follow these naming conventions for directories, individual files, and file contents:

- You should give all of your component directories and files unique and meaningful names. Keep in mind that as each component is loaded into Content Server, it overrides any resources with the same file names, so you should use duplicate file names only if you want certain components to take precedence.
- If you are copying a standard Content Server file, a common practice is to place the prefix `custom_` in front of the original file name. This ensures that you do not overwrite any default templates, and your customization is easy to identify.
- HTM file types should have the `.htm` extension, and HDA file types should have the `.hda` extension.
- If you are creating a new component file with a text editor, like WordPad, place the file name within quotation marks in the Save dialog box so that the proper file extension is assigned to it (for example, `myfile.hda`). Failure to use quotation marks to define the file name may result in a file name such as myfile.hda.txt.
- Content Server is case sensitive even if your file system is not. For example, if a file is named My_Template, Content Server does not recognize case variations such as my_template or MY_TEMPLATE.
- For localized string resources, you must follow the standard file naming conventions for Content Server to recognize the strings. You should also use the standard two-character prefix (cs, sy, ap, or ww) when naming your custom strings. For more information, see Section 1.5.5, "Localized String Resolution."

11.2 Tools for Managing Components

You can use the following tools to manage components:

- Component Wizard
- Component Manager
- Advanced Component Manager
- ComponentTool

11.2.1 Component Wizard

The Component Wizard utility automates the process of creating custom components, including creating and editing all the files necessary for custom components. You can also use the Component Wizard to modify existing components and to package and unpackage components for use on Content Server instances.

Figure 11–1 shows the interface to the Component Wizard. For more information, see "Creating Components Using the Component Wizard" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

Figure 11–1 Component Wizard Interface

To access the Component Wizard

- **UNIX operating system:** Run ComponentWizard, stored in *DomainHome*/ucm/*short-product-id*/bin/.

  The Component Wizard main page is displayed.

- **Windows operating system:** From the **Start** menu, choose the instance name, then **Utilities**, and then **Component Wizard**.

  The Component Wizard main page is displayed.

11.2.2 Advanced Component Manager

The Advanced Component Manager provides a way to manage custom components in Content Server. With the Advanced Component Manager, you can easily enable or disable components or add new components to Content Server.

To use the Advanced Component Manager:

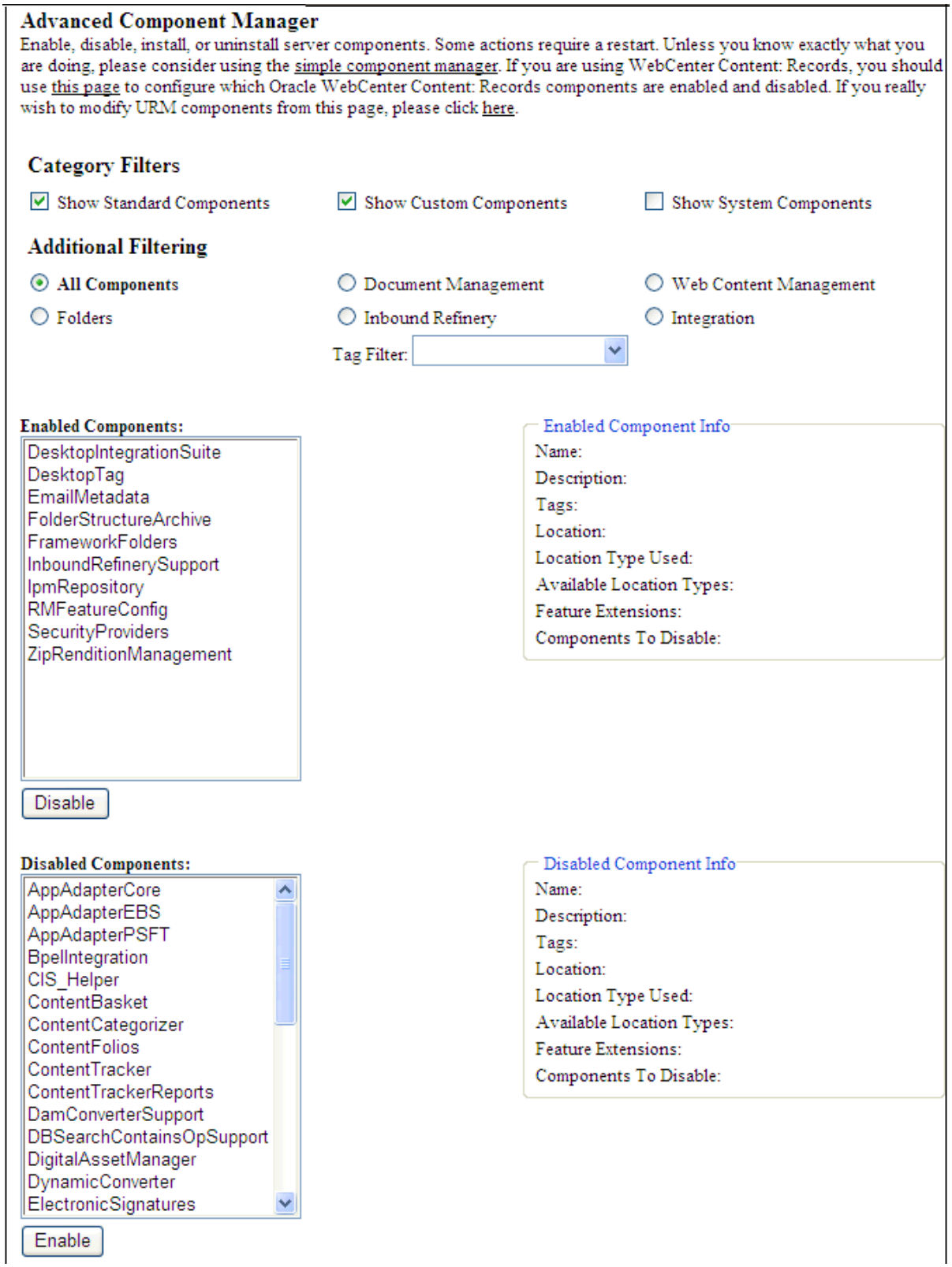
1. In the **Administration** tray or menu, choose **Admin Server**.

   The Admin Server displays the Component Manager page.

2. In the first paragraph on the Component Manager page, click **advanced component manager**.

   The Admin Server displays the Advanced Component Manager page, which Figure 11–2 shows. This page has lists of enabled and disabled components.

Figure 11–2 Advanced Component Manager Page



3. On the Advanced Component Manager page, you can do these tasks:
   - View lists of enabled and disabled components by categories and other filters
   - View details about a selected component
   - Enable components
   - Disable components
   - Install custom components
   - Uninstall custom components

For more information, see "Managing Components" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

11.2.3 ComponentTool

ComponentTool is a command-line utility for installing, enabling, and disabling components in Oracle WebCenter Content Server. After installing a component, ComponentTool automatically enables it. ComponentTool is located in the *DomainHome*`/ucm/cs/bin/` directory.

11.3 Component Files

The `idc`*short-product-id*`_components.hda` file tells Oracle WebCenter Content which components are enabled and where to find the component definition (glue) file for each component. In 11*g* Release 1 (11.1.1), this file has three forms, one for each of the WebCenter Content applications: `idccs_components.hda` (for Content Server), `idcibr_components.hda` (for Inbound Refinery), and `idcurm_components.hda` (for Records). The file is always stored in the *IntradocDir*`/data/components/` directory.

You should not create these files manually. Always use the Component Wizard to create your component files.

11.3.1 The idc Product _components.hda File

The `idc`*short-product-id*`_components.hda` file always includes a ResultSet called `Components` that defines the name and file path of each definition file. You can use the Component Wizard or the Component Manager to make changes to the components HDA file. For more information, see Chapter 12, "Enabling and Disabling Components for Content Server."

Example 11–6 shows an `idccs_components.hda` file that specifies two enabled components, `MyComponent` and `CustomHelp`, for Content Server.

Example 11–6 idccs_components.hda File

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
blFieldTypes=
@end
@ResultSet Components
2
name
location
MyComponent
custom/MultiCheckin/my_component.hda
```

```
CustomHelp
custom/customhelp/customhelp.hda
@end
```

11.3.2 Components ResultSet

The order that components are listed in the `Components` ResultSet determines the order that components are loaded when you start Content Server. If a component listed later in the ResultSet has a resource with the same name as an earlier component, the resource in the later component takes precedence.

A `Components` ResultSet has two columns:

- The `name` column provides a descriptive name for each component, which is used in the Component Wizard, Component Manager, and Content Server error messages.
- The `location` column defines the location of the definition file for each component. The location can be an absolute path or can be a path relative to the Content Server installation directory.

> **Note:** Always use forward slashes in the `location` path.

11.3.3 Component Definition (Glue) File

A component definition file, or **glue file**, points to the custom resources that you have defined. The definition file for a component is named *`component_name`*`.hda`, and it is typically located in the *`DomainHome`*`/ucm/`*`short-product-id`*`/custom/`*`component_name`* directory. The Component Wizard can be used to create and make changes to a definition file.

A definition file contains a ResourceDefinition ResultSet and may contain a MergeRules ResultSet, a Filters ResultSet, a ClassAliases ResultSet, or any combination of these ResultSets. Example 11–7 shows a typical component definition file.

Example 11–7 Component Definition File

```
<?hda jcharset=UTF8 encoding=utf-8?>
@Properties LocalData
classpath=$COMPONENT_DIR/classes.jar
ComponentName=Custom DCL Component
serverVersion=7.3
version=2010_10_22
@end
@ResultSet ResourceDefinition
4
type
filename
tables
loadOrder
template
dcl_templates.hda
DCLCustomTemplates
1
resource
dcl_resource.htm
null
1
resource
```

```
dcl_upper_clmns_map.htm
DCLColumnTranslationTable
1
resource
dcl_data_sources.htm
dclDataSources
1
service
dcl_services.htm
CustomServices
1
query
dcl_query.htm
CustomQueryTable
1
resource
dcl_checkin_tables.hda
null
1
@end

@ResultSet MergeRules
3
fromTable
toTable
column
DCLCustomTemplates
IntradocTemplates
name
DCLColumnTranslationTable
ColumnTranslation
alias
DCLDataSources
DataSources
name
CustomDCLServiceQueries
ListBoxServiceQueries
dataSource
@end

@ResultSet Filters
4
type
location
parameter
loadOrder
loadMetaOptionsLists
intradoc.server.ExecuteSubServiceFilter
GET_CHOICE_LIST_SUB
1
@end
```

11.3.3.1 ResourceDefinition ResultSet

The `ResourceDefinition` ResultSet table defines the type, file name, table names, and load order of custom resources. Example 11–8 shows the structure of a `ResourceDefinition` ResultSet:

Example 11–8 ResourceDefinition ResultSet

```
@ResultSet ResourceDefinition
4
type
filename
tables
loadOrder
template
dcl_templates.hda
DCLCustomTemplates
1
resource
dcl_resource.htm
null
1
resource
dcl_upper_clmns_map.htm
DCLColumnTranslationTable
1
resource
dcl_data_sources.htm
dclDataSources
1
service
dcl_services.htm
CustomServices
1
query
dcl_query.htm
CustomQueryTable
1
resource
dcl_checkin_tables.hda
null
1
@end
```

11.3.3.1.1 ResourceDefinition ResultSet Columns A `ResourceDefinition` ResultSet consists of four columns:

- The `type` column defines the resource type, which must be one of the following values:
  - `resource`, which points to an HTML include (HTM), string (HTM), dynamic table (HDA), or static table (HTM) resource file.
  - `environment`, which points to an environment resource (CFG) file.
  - `template`, which points to a template resource (HDA) file.
  - `query`, which points to a query resource (HTM) file.
  - `service`, which points to a service resource (HTM) file.

- The `filename` column defines the path and file name of the custom resource file. This can be an absolute path or a relative path. Relative paths are relative to the *DomainHome*`/ucm/`*short-product-id*`/custom/`*component_name* directory.
- The `tables` column defines the ResultSet tables to be loaded from the resource file. ResultSet names are separated with a comma. If the resource file does not include ResultSets, this value is null. For example, HTML include resources do not include table definitions, so the value for the `tables` column is always null for an HTML include file.
- The `loadOrder` column defines the order in which the resource is loaded. Resources are loaded in ascending order, starting with resources that have a loadOrder of 1. If multiple resources have the same `loadOrder`, the resources are loaded in the order they are listed in the ResourceDefinition ResultSet. If there are multiple resources with the same name, the last resource loaded is the one used by the system. Normally, you should set the `loadOrder` to 1, unless there is a particular reason to always load one resource after the others.

11.3.3.2 MergeRules ResultSet

The `MergeRules` ResultSet table identifies new tables that are defined in a custom component, and specifies which existing tables the new data is loaded into. MergeRules are **required** for custom template resources but are optional for custom dynamic table and static table resources. MergeRules are **not required** for custom service, query, HTML include, string, and environment resources.

Example 11–9 shows a `MergeRules` ResultSet.

Example 11–9 MergeRules ResultSet

```
@ResultSet MergeRules
4
fromTable
toTable
column
loadOrder
DCLCustomTemplates
IntradocTemplates
name
1
DCLColumnTranslationTable
ColumnTranslation
alias
1
DCLDataSources
DataSources
name
1
CustomDCLServiceQueries
ListBoxServiceQueries
dataSource
1
@end
```

11.3.3.2.1 MergeRules ResultSet Columns A `MergeRules` ResultSet consists of three columns:

- The `fromTable` column specifies a table that was loaded by a custom resource and contains new data to be merged with the existing data. To properly perform a merge, the `fromTable` table must have the same number of columns and the same column names as the `toTable` table.
- The `toTable` column specifies the name of the existing table into which the new data is merged. Usually, the `toTable` value is one of the standard Content Server tables, such as `IntradocTemplates` or `QueryTable`.
- The `column` column specifies the name of the table column that Content Server uses to compare and update data.
  - Content Server compares the values of `column` in `fromTable` and `toTable`. For each `fromTable` value that is identical to a value currently in `toTable`, the row in `toTable` is replaced by the row in `fromTable`. For each `fromTable` value that is not identical to a value currently in `toTable`, a new row is added to `toTable` and populated with the data from the row of `fromTable`.
  - The `column` value is usually `name`. Setting this value to null defaults to the first column, which is generally a `name` column.

11.3.3.3 Filters ResultSet

The `Filters` ResultSet table defines filters, which are used to execute custom Java code when certain Content Server events are triggered, such as when new content is checked in or when the server first starts. Example 11–10 shows a typical `Filters` ResultSet.

Example 11–10 Filters ResultSet

```
@ResultSet Filters
4
type
location
parameter
loadOrder
loadMetaOptionsLists
intradoc.server.ExecuteSubServiceFilter
GET_CHOICE_LIST_SUB
1
@end
```

11.3.3.4 ClassAliases ResultSet

The `ClassAliases` ResultSet table points to custom Java class files, which are used to extend the functionality of an entire Content Server Java class. Example 11–11 shows a typical `ClassAliases` ResultSet.

Example 11–11 ClassAliases ResultSet

```
@ResultSet ClassAliases
2
classname
location
WorkflowDocImplementor
WorkflowCheck.CriteriaWorkflowImplementor
@end
```

11.4 Resources for Assembling Web Pages

Resources are the files that define and implement the actual customization you make to Content Server. Resources can be snippets of HTML code, dynamic page elements, queries that gather data from the database, services that perform Content Server actions, or special code to conditionally format information.

The custom resource files for a component are typically located in the *DomainHome*/ucm/*short-product-id*/custom/*component_name* directory. If your component has more than a few resources, it is easier to maintain the files if you place them in subdirectories (such as *component_name*/resources or *component_name*/templates) within the component directory.

Always use the Component Wizard to create your resource files.You should not create a resource file manually. There are two ways to edit a resource file after it is created:

- Component Wizard

  You can add, edit, or remove a resource file from a component using the Component Wizard. The Component Wizard provides code for predefined resources that you can use as a starting point for creating custom resources. You can also open resource files in a text editor from within the Component Wizard.

- Manual editing in a text editor

  After creating a resource file with the Component Wizard, you can open the resource file in a text editor and edit the code manually, if necessary.

For more information, see Section 17.2, "Creating Resources for a Component."

Note: You must restart Content Server after changing a resource file.

12

Enabling and Disabling Components for Content Server

This chapter describes how to enable components that have been installed in Oracle WebCenter Content Server and how to disable components.

This chapter includes the following sections:

12.1 About Enabling and Disabling Components

By definition, a component is **enabled** when it is properly defined in the Components ResultSet in the `idc`*`short-product-id`*`_components.hda` file. A component is **disabled** if there is no entry or the entry is not formatted correctly.

12.2 Enabling a Component

There are several ways to enable a component:

- ComponentTool: Run *`DomainHome`*`/ucm/`*`short-product-id`*`/bin/ComponentTool` to enable a component. For example:

  ```
  ComponentTool -enable component_name
  ```

- Component Wizard: Choose **Enable** from the **Options** menu. For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.
- Component Manager: Select the checkbox next to a component name to enable a server component specified on the Component Manager screen. For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.
- Advanced Component Manager: On the Advanced Component Manager page, select a component name, and then click **Enable** to enable the component.

12.3 Disabling a Component

There are several ways to disable a component:

- ComponentTool: Run *DomainHome*/ucm/*short-product-id*/bin/ComponentTool to disable a component. For example:

  ```
  ComponentTool -disable component_name
  ```

- Component Wizard: Choose **Disable** from the **Options** menu. For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.
- Component Manager: Clear the checkbox next to a component name to disable a server component on the Component Manager screen. For more information, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.
- Advanced Component Manager: On the Advanced Component Manager page, select a component name, and then click **Disable** to disable the component.

13

Updating Component Configurations

This chapter provides information about updating the configuration of components in Oracle WebCenter Content Server.

This chapter includes the following sections:

13.1 About Updating Component Configurations

You can update, or modify, the configuration of some Content Server components with the Advanced Component Manager or the Configure for Instance screen, whether the component is enabled or disabled. The Advanced Component Manager has a list of the components whose configuration you can modify in the Update component configuration field. From the Configure for Instance screen, the Update Component Configuration screen is displayed for the specified component if you can modify its configuration, or if you cannot modify it, a message is displayed.

Content Server has Update Component Configuration screens for these components:

- `Folders_g`
- `PDF Watermark`
- `Content Tracker`
- `Content Tracker Reports`
- `Site Studio`
- `DesktopIntegrationSuite`
- `DesktopTag`
- `EmailMetadata`

13.2 Updating a Component Configuration with the Advanced Component Manager

For information about updating a component configuration with the Advanced Component Manager, see "Modifying a Component Configuration Using the Admin Server" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

13.3 Updating a Component Configuration Through the Configuration for instance Screen

For information about updating a component configuration through the Configuration for `instance` screen, see "Modifying a Component Configuration Using the Admin Server" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

14

Customizing Content Tracker

Content Tracker and Content Tracker Reports, both optional components of Oracle WebCenter Content Server, are installed with Oracle WebCenter Content. When enabled, these components provide information about system usage, such as which content items are most frequently accessed and what content is most valuable to users or specific groups. You can customize these components to provide specific information about the consumption patterns of your organization's content.

This chapter includes the following sections:

For information about using Content Tracker with the default settings, see *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

14.1 About Content Tracker

Content Tracker monitors activity on your Content Server instance and records selected details of those activities. It then generates reports that illustrate the way the system is being used. This section includes an overview about Content Tracker and Content Tracker Reports functionality.

Content Tracker incorporates several optimization functions which are controlled by configuration variables. The default values for the variables set Content Tracker to function as efficiently as possible for use in high volume production environments. For more information about Content Tracker configuration variable, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

14.1.1 Content Tracker Reports

Content Tracker monitors a system and records information about various activities which is collected from various sources and then merged and written to a set of tables in your Content Server database. Content Tracker can monitor activity based on these accesses and services:

- Content item accesses: Information about content item usage

  Data is obtained from Web filter log files, the Content Server database, and other external applications, such as portals and websites. Content item access data includes dates, times, content IDs, and current metadata.

- **Content Server services:** All services that return content, as well as services that handle search requests. By default, Content Tracker logs only the services that have content access event types but by changing configuration, Content Tracker can monitor any Content Server service, even custom services.

- **User accesses:** Information about other non-content access events such as the collection and synthesis of user profile summaries. This data includes user names and user profile information.

After Content Tracker extracts data and populates database tables, Content Tracker Reports can be used to:

- **Generate reports:** Content Tracker Reports queries the tables and generates summary reports of activities and usage history of content items. The reports help analyze specific groups of content or users based on metadata, file extensions, or user profiles. Use the predefined reports, customize them, or use a compatible third-party reporting package.

- **Optimize content management practices:** The reported data can be used for content retention management. Depending on the access frequency, some items could be archived or deleted. Applications can also use the data to provide portlets with the top content for particular types of users.

14.1.2 Content Tracker Components and Functions

Content Tracker provides a debugging configuration variable that, if enabled, configures the service handler filter to write out the service DataBinder objects into dump files (SctDebugServiceBinderDumpEnable). These can be used as diagnostic tools when developing field map screens. The dump files enable you to see what data is available at the time the particular service events are recorded.

14.1.2.1 DataBinder Dump Facility

When Content Tracker records a specific service in the log file, the contents of that service's DataBinder object are written to a serialized dump file. The contents of these files are useful for debugging when creating field maps to use the extended service call tracking function. These dump files allow you to see the available `LocalData` fields for the recorded service.

The Content Tracker service handler filter only creates dump files for DataBinder objects if the associated services are defined in the SctServiceFilter.hda file.

Caution: The dump files for DataBinder objects continue to accumulate until manually deleted. Use the `SctDebugServiceBinderDumpEnabled` configuration variable only as necessary.

14.1.2.1.1 Values for the DataBinder Dump Facility The values for this configuration variable include:

- **SctDebugServiceBinderDumpEnabled=False** prevents the Content Tracker service handler filter from writing out the DataBinder objects into dump files. This is the default value.
- **SctDebugServiceBinderDumpEnabled=True** configures the Content Tracker service handler filter to write out the DataBinder objects into dump files. Use a dump file as a diagnostic aid when you are developing field maps for extended service logging. If creating field maps for services, the dump files enable you to see what data is available at the time the service events are recorded.

14.1.2.1.2 Location of the DataBinder Object Dump Files The serialized DataBinder objects are written to:

IntradocDir/data/ContentTracker/DEBUG_BINDERDUMP/*dump_file_name*

14.1.2.1.3 Names of the DataBinder Object Dump Files The dump file of DataBinder Objects are text files and their names consist of three parts as follows:

service_name_filter_function_serial_number.hda

Where:

- *service_name* is the name of the logged service (such as, GET_FORM_FILE).
- *filter_function* is one of the following:
  - End: Filter Event 'on EndServiceRequestActions' - Normal end-of-service event.
  - EndSub: FilterEvent 'on EndScriptSubServiceActions' - Normal end-of-service for service called as SubService.
  - Error: Filter Event 'on ServiceRequestError' - End of service where an error occurred. May happen in addition to End.
- *serial_number* is the unique identification number assigned to the file. This enables Content Tracker to create more than one DataBinder object dump file for a given service.

Example:

```
GET_SEARCH_RESULTS_End_1845170235.hda
```

14.1.2.2 Performance Optimization

Content Tracker incorporates several optimization functions which are controlled by configuration variables. The default values for the variables set Content Tracker to function as efficiently as possible for use in high volume production environments.

Content Tracker collects and records only content access event data. This excludes information gathering on non-content access events like searches or the collection and synthesis of user profile summaries. Alternate values can be set during installation or changed later.

Performance variables include:

- `SctTrackContentAccessOnly`. **Content Access Only:** This determines what types of information is collected. When enabled (the default), only content access events are recorded.

- SctDoNotPopulateAccessLogColumns. **Exclude Columns:** This is a list of columns that Content Tracker does not populate in the SctAccessLog table. By default, bulky and rarely used information is not collected which reduces the size of the output table.
- SctSimplifyUserAgent. **Simplify User Agent:** This minimizes the information that is stored in the cs_userAgent column of the SctAccessLog table.
- SctDoNotArchive. **Do Not Archive:** This ensures that the database tables contain the most current data and expired table rows are discarded rather than archived. This is applicable to all Content Tracker database tables. By default, only the SctAccessLog table is populated but expired rows are not archived. However, if both SctTrackContentAccessOnly and SctDoNotArchive are disabled, all tables are populated and their expired data archived.

14.1.2.3 Installation Considerations

Set the SctUseGMT configuration parameter to true to use Greenwich Mean Time (GMT). It is set to false by default, to use local time. When upgrading from an earlier version of Content Tracker there is a one-time retreat (or advance, depending on location) in access times. To accommodate the biannual daylight savings time changes, discontinuities in recorded user access times are used (contingent on the use of local time and the location).

14.2 Customizing Content Tracker with Configuration Variables

You can use configuration variables to customize Content Tracker.

14.2.1 About Configuration Variables

The following table lists the default values of the configuration settings used in the current version of Content Tracker. These configuration variables are contained in the Content Tracker configuration file:

cs_root/data/contenttracker/config/sct.cfg

Config. Setting	Default Value	Remarks
SctAutoTruncateDataStrings	FALSE	Used by: JAVA Determines if the reduction process truncates data strings to fit into the corresponding table column.
SctComponentDir	*cs_root*/data/contenttracker/	Used by: JAVA Path to the directory where Content Tracker is installed.
SctDebugLogEnabled	FALSE	Used by: JAVA Set TRUE to enable Java code execution trace. Used with SctDebugLogFilePath.
SctDebugLogFilePath	*cs_root*/data/contenttracker/log/SCT_DEBUG_TRACE.log	Used by: JAVA Directory for Java code execution trace. Used with SctDebugLogEnabled.

Config. Setting	Default Value	Remarks
SctDebugServiceBinderDumpEnabled	FALSE	Used by: JAVA Set TRUE to enable diagnostic output of Service DataBinder objects during Service logging.
SctExternalUserLogEnabled	TRUE	Used by: JAVA Set TRUE to enable replication of External user account and role information to UserSecurityAttributes table.
SctFilterPluginLogDir	*cs_root*/data/contenttracker/data/	Used by: filter plugin Path to the directory where filter plug-in stores the event logs.
SctIdcAuthExtraConfigParams		List of Content Tracker configuration parameters passed to the filter plugin, merged programmatically into idcAuthExtraConfigParams by the Content Tracker startup filter.
SctIgnoreDirectories	*DomainHome*/ucm/cs/resources/; *DomainHome*/ucm/cs/common/	Used by: filter plugin Directs filter plugin to disregard URLs contained within the listed directory roots.
SctIgnoreFileTypes	gif,jpg,js,css	Used by: filter plugin Directs filter plugin to disregard URLs with the listed file types.
SctLogDir	*cs_root*/data/contenttracker/data/	Used by: JAVA Path to one or more directories where Content Tracker looks for the raw event logs - sctLog, and so on. This parameter can be multivalued, as *dir1;dir2;…;dirn*.
SctLogEnabled	TRUE	Used by: filter plugin, JAVA If False, directs service handler filters and web server filter plugin to ignore all events and create no logs. This is the Content Tracker Master On/Off switch.
SctLogSecurity	TRUE	Used by: filter plugin, JAVA If true, directs filter plugin to record IMMEDIATE_RESPONSE_PAGE events in the sctSecurityLog event log, and the reduction process to read the event log.
SctMaxRecentCount	5	Used by: JAVA Maximum number of days worth of reduced data kept in the "Recent" state. Overflow from Recent is moved to Archive state.

Config. Setting	Default Value	Remarks
SctMaxRereadTime	3600	Used by: JAVA Maximum number of seconds that can occur between consecutive references by a particular user to a particular content item; for example, a PDF file, and have the adjacent references be considered a single sustained access. Consecutive references which occur further apart in time count as separate accesses.
SctReductionAvailableDatesLookback	0	Used by: JAVA Used with SctReductionRequireEventLogs to limit Available Dates range. Unit = Days. Zero = unlimited.
SctReductionLogDir	*cs_root*/data/contenttracker/log/	Used by: JAVA Path to the directory where the Content Tracker reduction logs are stored.
SctReductionRequireEventLogs	TRUE	Used by: JAVA Used in Detached configurations. FALSE means proceed with Reduction even if no event logs are found.
SctScheduledReductionEnable	TRUE	Used by: JAVA Used in multi-JVM configurations to select which Content Server instance performs the reduction.
SctSnapshotEnable	FALSE	Used by: JAVA Set TRUE to enable Snapshot functions. Set from Data Engine Control Center.
SctSnapshotLastAccessEnable	FALSE	Used by: JAVA Set TRUE to enable Last Access Date Snapshot function. Set from Data Engine Control Center.
SctSnapshotLastAccessField	[none]	Used by: JAVA Metadata field name for Last Access Date, e.g. xLastAccessDate. Set from Data Engine Control Center.
SctSnapshotLongCountEnable	FALSE	Used by: JAVA Set TRUE to enable "Long" interval access count Snapshot function. Set from Data Engine Control Center.
SctSnapshotLongCountField	[none]	Used by: JAVA Metadata field name for Long Interval Count; for example, xAccessesInLast90Days. Set from Data Engine Control Center.

Config. Setting	Default Value	Remarks
SctSnapshotLongCountInterval	[none]	Used by: JAVA Number of days for "Long" Interval. Set from Data Engine Control Center.
SctSnapshotShortCountEnable	FALSE	Used by: JAVA Set TRUE to enable "Short" interval access count Snapshot function. Set from Data Engine Control Center.
SctSnapshotShortCountField	[none]	Used by: JAVA Metadata field name for Short Interval Count; for example, xAccessesInLast10Days. Set from Data Engine Control Center.
SctSnapshotShortCountInterval	[none]	Used by: JAVA Number of days for "Short" Interval. Set from Data Engine Control Center.
SctUseGMT	FALSE	Used by: filter plugin, JAVA Set TRUE for logged event times to be converted to Universal Coordinated Time. FALSE uses local time.

The following variables are not available in the `sct.cfg` file and are accessible only through the Component Manager.

Config. Setting	Default Value	Remarks
SctPostReductionExec	[none]	Used by: JAVA Path to Post Reduction Executable (assumed to be in *IntradocDir*/custom/ContentTracker/bin/)
SctProxyNameMaxLength	50	Used by: JAVA Maximum number of characters in the name of any Content Server proxy server in the configuration. Used to increase the size of user name fields in Content Tracker table creation.
SctUrlMaxLength	3000	Used by: JAVA Maximum expected length (characters) for URL fields. Used to determine column widths when creating tables. There may be several such columns in a given table.
SctWebBeaconIDList	[none]	Used by: filter plugin List of zero or more web beacon objects. Required to add the ability to feed data to Content Tracker using client-side tags. Enables Content Tracker to gather data from cached pages and pages generated from cached services.

For more information about the Content Tracker configuration variables, see *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

14.2.1.1 Access Control Lists and Content Tracker Reports Secure Mode

The security checks preference variable (`SctrEnableSecurityChecks`) is set when the Content Tracker Reports component is installed. This preference variable enables selection of one of two security modes: secure and non-secure. The security checks

preference provides the option to employ individual user role and account information to restrict the visibility of content item information in report results.

This means you control what content items (and, subsequently, the metadata) that users can see in their generated reports. Ideally, users should not see anything through Content Tracker Reports that they could not find through a Content Server search. If secure mode is used, the information in any generated report is filtered based on the user's role and account privileges.

However, if Access Control Lists (ACLs) are enabled on your Content Server instance, the secure mode option in Content Tracker Reports does not work. During installation, leave the security checks preference checkbox blank. This means that on an ACL-based system, the secure mode must be disabled. In this case, it is possible for users other than a system administrator to see information about content items that they would not otherwise be authorized to access and view.

14.2.1.2 Values for the Security Checks Preference Variable

The values for the security checks preference variable include:

- `SctrEnableSecurityChecks=True` enables the security checks installation preference and configures Content Tracker Reports to operate in secure mode.

  In secure mode, the same security criteria (role and account qualifications) used to limit Content Server search results are also applied to the Content Tracker Report Generator's queries and the generated reports. Thus, it is possible that two different users running the Top Content Items report may see different results.

- `SctrEnableSecurityChecks=False` disables the security checks installation preference and configures Content Tracker Reports to operate in non-secure mode. This is the default setting.

  In non-secure mode, the additional role and account criteria used to restrict Content Server search results are not applied to Content Tracker Report Generator's queries and the generated reports. Thus, it is possible for a user other than a system administrator to see information about content items that they would not be authorized to access and view.

14.2.1.3 File Types for Entries in the SctAccessLog

By default, Content Tracker does not log accesses to GIF, JPG, JS, CSS, CAB, and CLASS file types. Therefore, entries for these file types are not included in the combined output table after data reduction.

To log these file types, enable the file type in the sct.cfg file located in the *IntradocDir*/custom/ContentTracker/resources/ directory. Change the default setting for the `SctIgnoreFileTypes` configuration variable (`gif,jpg,js,css`). The default setting excludes these file types. To include one or more of these file types, delete each desired file type from the list. To ensure that these changes take effect, it is necessary to restart the web server and Content Server.

14.2.2 Setting Content Tracker Configuration Variables

To set or edit any of the Content Tracker configuration variables:

1. In a text editor, open the `sct.cfg` file:

   cs_root`/data/contenttracker/config/sct.cfg`

2. Locate the configuration variable to be edited.
3. Enter the applicable value.

4. Save and close the `sct.cfg` file.
5. Restart Content Server to apply the changes.

Add or edit the configuration variables for the activity metrics metadata fields with the user interface included in the Data Engine Control Center. These include the following variables:

- `SctSnapshotEnable`
- `SctSnapshotLastAccessEnable`
- `SctSnapshotLastAccessField`
- `SctSnapshotLongCountEnable`
- `SctSnapshotLongCountField`
- `SctSnapshotLongCountInterval`
- `SctSnapshotShortCountEnable`
- `SctSnapshotShortCountField`
- `SctSnapshotShortCountInterval`

For more information about the user interface and the activity metrics functions, see "Data Tracking Functions" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

14.2.3 Tracking External Users and Content Items

The option exists to control if Content Tracker includes data about external user accesses in the applicable reports. These authenticated users are qualified based on their user roles and accounts. By default, the configuration parameter `SctExternalUserLogEnabled` is set to true (enabled). This allows Content Tracker to monitor external user logins and automatically propagate their role and account information to the UserSecurityAttributes table.

Regardless of whether the `SctExternalUserLogEnabled` configuration variable is enabled or disabled, all of the content item access information for external users is tracked and recorded. But when it is enabled, this variable ensures that this data is included in reports that explicitly correlate externally authenticated user names with their associated user roles and accounts. Specifically, the Top Content Items by User Role report and the Users by User Role report include all of the content item access activity by external users. For more information, see "Content Tracker Reports" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

Note: To manually disable the `SctExternalUserLogEnabled` configuration variable, see Section 14.2.2, "Setting Content Tracker Configuration Variables."

14.3 Configuring Service Calls

You can configure service calls in the service call configuration file, configure the Content Tracker logging service to log events, and manage service call information.

14.3.1 About the Service Call Configuration File

The Content Tracker service handler filter makes it possible to gather information about Content Server activity other than content requests. Service request details are

collected by the service handler filter and stored in the SctAccessLog table in real time. The details are obtained from the DataBinder that accompanies the service call. For a Content Server service call to be logged, it must have an entry in the service call configuration file (SctServiceFilter.hda).

The `SctServiceFilter.hda` file is a user-modifiable configuration file that is used to limit the number of logged service calls. This enables you to selectively control which services are logged. The data logging function for any service call included in the SctServiceFilter.hda file can also be expanded, to log and track data values of specific DataBinder fields relevant to a particular service. For more information, see Section 14.3.1.2, "Extended Service Call Tracking Function."

Service tracking is limited to top-level services called through the server socket port. Sub-services, or services called internally, cannot be tracked.

The purpose of the SctServiceFilter.hda file is to define which parts of Content Server are of particular interest to users. If a Content Server service is not listed in the SctServiceFilter.hda file, it is ignored by Content Tracker. Additionally, if a service is not listed in this file, it can only be logged by the Content Tracker logging service. For more information, see Section 14.3.2, "About the Content Tracker Logging Service."

You can make changes to the `SctServiceFilter.hda` file in two ways:

- Add new services and edit the existing service call parameters in the file from the Data Engine Control Center.
- Manually edit the `SctServiceFilter.hda` file.

  For more information, see Section 14.3.3.1, "Manually Editing the SctServiceFilter.hda File."

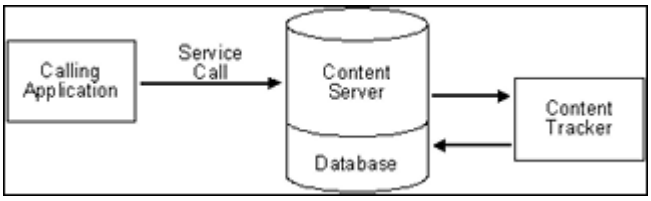
> **Tip:** Control the services to log by including or excluding them from the `SctServiceFilter.hda` file. This is an effective method to control logging for particular services or for all services. Also, the extended service call tracking function enables customization of the type of data that is logged for a specific service.

14.3.1.1 General Service Call Logging

Services listed in the `SctServiceFilter.hda` file are detected by the Content Tracker service handler filter and the values of selected data fields are captured. Content Tracker then logs the named service calls. The information with the timestamps, and so on, are written dynamically into the `SctAccessLog` table.

For each enabled service, Content Tracker automatically logs certain standard DataBinder fields, such as `dUser` and `dDocName`. Also, DataBinder fields associated with the extended service call tracking function are logged to the general purpose columns in the `SctAccessLog` table.

Data is inserted into the `SctAccessLog` table in real time using Content Tracker-specific services sequence numbers and a type designation of "S" for service. ("W" designations indicate static URL event types). Manual and/or scheduled reductions are only required to process the static URL access information gathered by the web server filter.



14.3.1.2 Extended Service Call Tracking Function

The extended service call tracking function enables the logging of Content Server service calls and supplement this information by also logging relevant data values from one or more additional DataBinder fields other than the standard DataBinder fields logged by each configured service call.

14.3.1.2.1 Service Call ResultSet Combinations Each service that Content Tracker logs must have an entry in the `ServiceExtraInfo` ResultSet that is contained in the `SctServiceFilter.hda` file. Content Tracker automatically logs various standard DataBinder fields, such as **dUser** and **dDocName**. However, the service-related data logged by Content Tracker can be expanded by logging and tracking relevant data values from supplementary DataBinder fields.

The extended service call tracking function is implemented by linking the entries in the `ServicesExtraInfo` ResultSet to field map ResultSets. Each field map ResultSet contains one or more sets of data field names, the source location, and the destination table column name in the **SctAccessLog** table. This grouping allows you to select data fields relevant to the associated service call and have the data values logged into the specified column in the **SctAccessLog** table.

Since more than one expanded service can be logged using the extended tracking function, the contents of the general purpose columns in the SctAccessLog table cannot be properly interpreted without knowing which service is being logged. The service name is always logged in the **sc_scs_idcService** column. Your queries should match this column with the desired service name.

Caution: In field map ResultSets, you can map data fields to existing, standard SctAccessLog table columns. The extended service mapping occurs after the standard field data values are collected. You can override any of the standard table column fields.

For example, the service you are logging might carry a specific user name (such as, MyUserName=john) in a data field. You could use the extended tracking function to override the contents of the sc_scs_dUser column. In this case, you simply combine MyUserName and sc_scs_dUser and use them as the data field, location, and table column set in the field map ResultSet.

It is your responsibility to ensure that the data being logged is a reasonable fit with the SctAccessLog column type.

For examples of linked service entries and ResultSets, see Section 14.3.1.4.2, "Linked Service Entries and Field Map ResultSets." For more information about the contents of the `SctAccessLog` table and the general purpose columns intended to be mapped to data fields, see "Combined Output Table" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

14.3.1.2.2 General Purpose Columns in the Output Table In the field map ResultSets for extended service tracking, map the DataBinder fields to columns in the SctAccessLog table. The general purpose columns (extField_1 through extField_10) are available for mapping. These columns may be filled with any data values you consider appropriate for logging and tracking for a particular service. It is recommended and expected that you use these columns to avoid overwriting the standard table columns.

Tip: The name of the service is always logged to the `sc_scs_idcService` column. Include it as a qualifier in any query that uses the contents of the extended fields. For more information about custom reports that include specific SQL queries involving `SctAccessLog` table columns, see "Report Creation Types" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

14.3.1.3 Service Call Configuration File Contents

The initial contents of the service call configuration file (SctServiceFilter.hda) are the commonly used content access, search, and user authentication services native to Content Server. This file contains a ResultSet structure with one entry for each service to be logged. To support the extended service call tracking function, this file may also include field map ResultSets linked to the service entries contained in the ServiceExtraInfo ResultSet.

Add new entries or edit existing entries, or both, in the SctServiceFilter.hda file with the Services user interface accessed through the Data Engine Control Center or change entries in the file manually. For more information, see Section 14.3.3.1, "Manually Editing the SctServiceFilter.hda File."

Note: To review the set of initial services that Content Tracker logs into the SctAccessLog table see the `SctServiceFilter.hda` file:

cs_root`/data/contenttracker/config/SctServiceFilter.hda`

The following tables provide details of the service call configuration file ResultSet schema. The values are copied directly to the corresponding columns in the `SctAccessLog` table.

ServiceExtraInfo ResultSet Contents

Feature	Description
Service Name (sctServiceName)	The name of the service to be logged. For example, GET_FILE. If no row is present in the ResultSet for a given service, the service is not logged.
Calling Product (sctCallingProduct)	An arbitrary string. It is generally set to "Core Server" for all standard Content Server entries.
Event Type (sctEventType)	An arbitrary string. It is generally set to "Content Access" for all standard Content Server entries.
Reference (sctReference)	Used to set the sc_scs_reference field in the SctAccessLog table. If blank, the internal getReference logic is used.

Feature	Description
Field Map (sctFieldMap)	The name of the field map ResultSet that is added to the SctServiceFilter.hda file. This field is only required if the extended service call tracking function is used. This function enables the logging of DataBinder field information to one or more of the general purpose columns in the SctAccessLog table.

Field Map ResultSet Contents:

Feature	Description
Field Map Link	The name of the field map ResultSet. A configuration variable can be set that writes out the service DataBinder object. This enables you to see the data available at the time the event is recorded.
DataBinder Field (dataFieldName)	The name of the DataBinder field name whose data values are logged to a general purpose column in the SctAccessLog table. See also the Field Name field on the Field Map Screen.
Data Location (dataLocation)	The section in the Content Server service DataBinder where the field to be logged is located.See also the Field Location field on the Field Map Screen.
Access Log Column (accessLogColumnName)	The specific general purpose column in the SctAccessLog table where data values from a specified DataBinder field are logged. See also the Column Name field on the Field Map Screen.

The fields copied from the DataBinder and inserted into the SctAccessLog table include: `dID`, `dDocName`, `IdcService`, `dUser`, `SctCallingProduct`, `SctEventType`, and `SctReference`. If the values for the latter three fields are included in a service's entry in the `SctServiceFilter.hda` file, they override the corresponding values in the data field.

There should be no duplication or conflicts between services logged through the service handler filter and those logged through the Content Tracker logging service. If a service is named in the Content Tracker service handler filter file then such services are automatically logged so there is no need for the Content Tracker logging service to do it.

> **Tip:** Adding desired service calls to the SctServiceFilter.hda file and using this method to log specific activity allows you the advantage of providing values for the CallingProduct, EventType, and Reference fields. The assigned values are copied directly to the corresponding columns in the in the SctAccessLog table.

14.3.1.4 ResultSet Examples

The default SctServiceFilter.hda file includes various common service calls.

Note: To review the initial set of services that Content Tracker logs into the SctAccessLog table and the service entries and field map ResultSets see the `SctServiceFilter.hda` file:

cs_root`/data/contenttracker/config/SctServiceFilter.hda`

For more detailed information about these services, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*

14.3.1.4.1 ServiceExtraInfo ResultSet Entries The following list provides examples of several service entries contained in the SctServiceFilter.hda file's ServiceExtraInfo ResultSet.

- `GET_FILE_BY_NAME`

  `Core Server`

  `Content Access`

- `GET_DYNAMIC_URL`

  `Core Server`

  `Content Access`

- `GET_DYNAMIC_CONVERSION`

  `Core Server`

  `Content Access`

- `GET_EXTERNAL_DYNAMIC_CONVERSION`

  `Core Server`

  `Content Access`

- `GET_ARCHIVED_FILE`

  `Core Server`

  `Content Access`

- `COLLECTION_GET_FILE`

  `Folders`

  `Content Access`

14.3.1.4.2 Linked Service Entries and Field Map ResultSets The following table lists several examples of service entries linked to field map ResultSets. These examples, or other similar ones, are included in the initial SctServiceFilter.hda file.

Service Entries	Field Map ResultSets
GET_SEARCH_RESULTS Core Server Search SearchFieldMap	@ResultSet SearchFieldMap 3 dataFieldName 6 255 dataLocation 6 255 accessLogColumnName 6 255 MiniSearchText LocalData extField_1 TranslatedQueryText LocalData extField_2 IsSavedQuery LocalData extField_7 @end
PNE_GET_SEARCH_RESULTS Core Server Search SearchFieldMap	@ResultSet SearchFieldMap 3 dataFieldName 6 255 dataLocation 6 255 accessLogColumnName 6 255 MiniSearchText LocalData extField_1 TranslatedQueryText LocalData extField_2 IsSavedQuery LocalData extField_7 @end
GET_FILE Core Server Content Access GetFileFieldMap	@ResultSet GetFileFieldMap 3 dataFieldName 6 255 dataLocation 6 255 accessLogColumnName 6 255 RevisionSelectionMethod LocalData extField_1 Rendition LocalData extField_2 @end

14.3.2 About the Content Tracker Logging Service

The Content Tracker logging service is a single service call (SCT_LOG_EVENT) that allows an application to log a single event to the SctAccessLog table. The service may be called directly through a URL or as an action in a service script. It may also be called from Idoc Script using the executeService() function. The calling application is responsible for setting any and all fields in the service DataBinder to be recorded, including the descriptive fields listed in the Content Tracker SctServiceFilter.hda configuration file.

The SCT_LOG_EVENT service copies information out of the service DataBinder. This data is inserted into the `SctAccessLog` table in real time using the Content Tracker specific services sequence numbers and a type designation of "S" for service. Manual

or scheduled reductions, or both, are required only to process the static URL access information gathered by the web server filter. For more information, see "Data Reduction" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

> **Note:** There should be no duplication or conflicts between services logged through the service handler filter and those logged through the Content Tracker logging service. If a service is named in the Content Tracker service handler filter file then such services are automatically logged so there is no need for the Content Tracker logging service to do it. However, Content Tracker makes no attempt to prevent such duplication.

14.3.3 Managing Service Call Information

This section provides information and task procedures for mapping and logging data from Content Server services to the combined output database table (SctAccessLog).

14.3.3.1 Manually Editing the SctServiceFilter.hda File

To add or change entries in the SctServiceFilter.hda file:

1. In a text editor, open the SctServiceFilter.hda file:

   cs_root/data/contenttracker/config/.../SctServiceFilter.hd

2. Edit an existing entry or add a new service entry. For example, to add the GET_FILE_FORM service, enter the following service entry to the ServiceExtraInfo ResultSet in the file:

   ```
   GET_FORM_FILE
   Threaded Discussion
   Content Access
   optional_reference_value
   optional_field_map_link_value
   ```

   where the *optional_field_map_link_value* is used when implementing the extended service call tracking function. In this case, add or edit the corresponding field map ResultSet. If implementing extended service tracking, skip Step 3.

3. When using extended service tracking, add or edit the corresponding field map ResultSet. For example, to add the SS_GET_PAGE service and track additional data field values, enter the following service entry and corresponding field map ResultSets to the file:

Service Entry	Field Map ResultSet
SS_GET_PAGE Site Studio Web Hierarchy Access web SSGetPageFieldMap	@ResultSet SSGetPageFieldMap 3 dataFieldName 6 255 dataLocation 6 255 accessLogColumnName 6 255 *DataBinder_field_name* *data_field_location_name* *access_log_column_name* @end

Note: Include as many sets of DataBinder field, location, and table column names as necessary.

4. Save and close the file.
5. Restart the Content Server to apply the new definitions.

Note: Search request events are logged into the `SctAccessLog` table in real time and do not need to be reduced. Add or edit services with the user interface included in the Data Engine Control Center.

14.3.3.2 Setting Required DataBinder Fields to Call the Content Tracker Logging Service

The following table provides the `SctAccessLog` column names and the corresponding DataBinder fields that Content Tracker looks for when the Content Tracker logging service (SCT_LOG_EVENT) is called. When an application calls the Content Tracker logging service, the application is responsible for setting the necessary fields in the service DataBinder for Content Tracker to find. For more detailed information about the `SctAccessLog` fields, see "Combined Output Table" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

SctAccessLog Column Name	Service DataBinder LocalData Field
SctDateStamp	[computed]
SctSequence	SctSequence
SctEntryType	"S"
eventDate	[computed]
SctParentSequence	SctParentSequence
c_ip	REMOTE_HOST
cs_username	HTTP_INTERNETUSER
cs_method	REQUEST_METHOD
cs_uriStem	HTTP_CGIPATHROOT
cs_uriQuery	QUERY_STRING
cs_host	SERVER_NAME
cs_userAgent	HTTP_USER_AGENT
cs_cookie	HTTP_COOKIE
cs_referer	HTTP_REFERER
sc_scs_dID	dID
sc_scs_dUser	dUser
sc_scs_idcService	IdcService (or SctIdcService)
sc_scs_dDocName	dDocName
sc_scs_callingProduct	sctCallingProduct
sc_scs_eventType	sctEventType
sc_scs_status	StatusCode

SctAccessLog Column Name	Service DataBinder LocalData Field
sc_scs_reference	sctReference (also ...)
comp_username	[computed - HTTP_INTERNETUSER or ...]
sc_scs_isPrompt	n/a
sc_scs_isAccessDenied	n/a
sc_scs_inetUser	n/a
sc_scs_authUser	n/a
sc_scs_inetPassword	n/a
sc_scs_serviceMsg	StatusMessage

14.3.3.3 Calling the Content Tracker Logging Service from an Application

You can call the SCT_LOG_EVENT service from an application. This can be done by the application developer, or by a user willing to modify the application service scripts. The application can call SCT_LOG_EVENT from Java. Or, the application can include calls to SCT_LOG_EVENT in the service script.

14.3.3.4 Calling the Content Tracker Logging Service from Idoc Script

You can call the SCT_LOG_EVENT service indirectly from Idoc Script, using the executeService() function. This is the same as calling the SCT_LOG_EVENT service from an application except that it occurs from Idoc Script instead of the application Java code. Content Tracker cannot distinguish if the SCT_LOG_EVENT service is called from Java or from Idoc Script.

14.3.4 Service Call Management and the User Interface

Content Tracker enables the logging of service calls with data values relevant to the associated services. Every service to be logged must have a service entry in the service call configuration file (SctServiceFilter.hda). In addition to the logged services, their corresponding field map ResultSets can be included in the SctServiceFilter.hda.

Content Tracker only logs services that have event types for content access or services that cause an entry to be made in the DocHistory table. This ensures maximum performance, but some service events are not logged.

The enabled services automatically log general DataBinder fields, such as dUser and dDocName. Linking a field map ResultSet to a service entry enables the use of the extended service call tracking function.

The SctAccessLog database table provides additional columns for use with the extended service call tracking function which can be filled with any data values appropriate for the associated service call. When listing the data field names in the field map ResultSet, also list the location name for the source of the data field, and the table column name where the data is logged.

Caution: In field map ResultSets, you can map data fields to existing, standard SctAccessLog table columns. The extended service mapping occurs after the standard field data values are collected. Therefore, any of the standard table column fields can be overwritten.

For example, the service you log might carry a specific user name (`MyUserName=john`) in a data field. You could use the extended tracking function to overwrite the contents of the sc_scs_dUser column. In this case, combine MyUserName and sc_scs_dUser and use them as the data field, location, and table column set in the field map ResultSet.

It is your responsibility to ensure that the data being logged is a reasonable fit with the SctAccessLog column type.

14.3.4.1 Adding, Editing, or Deleting Service Entries

Follow these steps to add or edit a service:

1. Choose **Administration** then **Content Tracker Administration** from the Main menu. Choose **Data Engine Control Center**.

   The Data Engine Control Center opens.

2. Click the **Services** tab.

3. Click **Add** to create a new service entry, or choose an existing service entry from the **Service Name** list and click **Edit**.

   The Extended Services Tracking screen opens. The fields are empty when adding a new service entry. When editing an existing service entry, the fields are populated with values that can be edited.

4. Enter or modify the applicable field values (except in the Field Map field).

   To link this service entry to a field map ResultSet, enter the applicable name in the `Field Map` field, and then link the field. For more information, see "Linking Activity Metrics to Metadata Fields" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

5. Click **OK**.

   A confirmation dialog box is displayed.

6. Click **OK**.

   The **Services** tab is redisplayed with the new service or newly edited service in the Services list. The services state and Content Tracker's SctServiceFilter.hda are updated.

   Content Tracker does not perform error checking (such as field type or spelling verification) for the extended services tracking function in the Data Engine Control Center. Errors are not generated until a reduction is done. These fields are case-sensitive. When adding new services or editing existing services, be careful to enter the proper service call names. Ensure that all field values are spelled and capitalized correctly.

To delete an entry, follow the previous steps, highlight an entry, and select **Delete**.

14.3.4.2 Adding, Editing, or Deleting Field Map ResultSets

To implement the extended service call tracking function, service entries must be linked to field map ResultSets in the SctServiceFilter.hda file.

Follow these steps to add a field map and link it:

1. Choose **Administration** then **Content Tracker Administration** from the Main menu. Choose **Data Engine Control Center**.

   The Data Engine Control Center opens.

2. Click the **Services** tab.

3. To add a new entry, follow the procedure in Section 14.3.4.1, "Adding, Editing, or Deleting Service Entries." Choose the service entry from the **Service Name** list.

4. Click **Edit**.

   The Extended Services Tracking screen opens. If necessary, edit this service entry's values now in addition to adding the field map ResultSet.

   If the service is already linked to a field map ResultSet, the name is listed in the **Field Map** field and one or more data field, location, and table column set are listed in the Field area.

5. If the selected service is not linked to a secondary ResultSet, the **Field Map** field is empty. Enter the name of the field map ResultSet. If the selected service is already linked, skip this step.

6. Click **Add**.

   The Field Map screen opens.

7. Enter the appropriate values in the fields:

   - **Field Name**: The name of the data field in the service DataBinder whose data values are logged to a general purpose column in the **SctAccessLog** table.
   - **Field Location**: The section in the Oracle WebCenter Content Server service DataBinder where the data field to be logged is located. You can use the following values:
     - `LocalData` (the default value)
     - `Environment`
     - `BinderResultSet`. This returns a comma-delimited string containing all values in the ResultSet. Size is restricted to 255 characters, allowing for commas and so on, so this value is useful only for small ResultSets.

       To accommodate more characters, enlarge or redefine the **SctAccessLog** table columns using standard database tools. For example, if you open up `extField_3` to `2047`, then it holds the equivalent amount of data. However, most databases have page-size limitations. In addition, SQL does not parse strings efficiently.
   - **Column Name**: The column in the **SctAccessLog** table where data values from a specified DataBinder field are logged.

8. Click **OK**.

   The Field Map screen closes, and the values are added to the **Field Name** and **Column Name** fields.

9. Click **OK** again.

   A confirmation dialog box opens.

   The **Services** tab is redisplayed with the updated information.

10. Click **OK**.

Content Tracker does not perform error checking (such as field type or spelling verification) for the extended services tracking function in the Data Engine Control Center. Errors are not generated until a reduction is done. These fields are case-sensitive. When adding new field map ResultSets or editing existing field map ResultSets, be sure to enter the proper names and ensure that all field values are spelled and capitalized correctly.

To edit a field map, perform the previous steps, and edit the entries as needed.

To delete an entry, perform the previous steps, highlight a service entry, and select **Delete**.

14.4 Customizing the Activity Metrics SQL Queries

The snapshot feature enables you to log and track search relevance custom metadata fields. Content Tracker fills these fields with content item usage and access information that reflects the popularity of particular content items. The information includes the date of the most recent access and the number of accesses in two distinct time intervals. For more information about the snapshot feature, see "Activity Snapshots" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

If the snapshot feature and activity metrics are enabled, the values in the custom metadata fields are updated following the reduction processing phase. When users access content items, the values of the applicable search relevance metadata fields change accordingly. Subsequently, Content Tracker runs three SQL queries as a post-reduction processing step to determine which content items were accessed during the reporting period. For more information about the post-processing reduction step, see "Data Reduction Process with Activity Metrics" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

The SQL queries are available as a resource and can be customized to filter information from the final tracking data. For example, you might want to exclude accesses by certain users in the tabulated results.

The SQL queries are included in the sctQuery.htm file:

IntradocDir/custom/ContentTracker/resources/SctQuery.htm

> **Note:** In general, the WHERE clause can be modified in any of the SQL queries. It is recommended that nothing else be modified.

The following SQL queries are used for the search relevance custom metadata fields:

- `qSctLastAccessDate`: For the last access function, this query uses the SctAccessLog table. It checks for all content item accesses on the reduction date and collects the latest timestamp for each `dID`. The parameter for the query is the reduction date. In this case, dates may be reduced in random order because the comparison test for the last access date only signals a change if the existing `DocMeta` value is older than the proposed new value.
- `qSctAccessCountShort` and `qSctAccessCountLong`: For the short and long access count functions, the `qSctAccessCountShort` and `qSctAccessCountLong` SQL

queries are identical except for the "column name" for the count. They use the **SctAccessLog** table to calculate totals for all accesses for each `dID` across the time intervals specified (in days) for each. The parameters are the beginning and ending dates for the applicable rollups.

14.4.1 Tracking Access to Content Items by External Users

The option exists to control if Content Tracker includes data about external user accesses in the applicable reports. These authenticated users are qualified based on their user roles and accounts. By default, the configuration parameter SctExternalUserLogEnabled is set to true (enabled). This allows Content Tracker to monitor external user logins and automatically propagate their role and account information to the UserSecurityAttributes table.

Regardless of whether the SctExternalUserLogEnabled configuration variable is enabled or disabled, all of the content item access information for external users is tracked and recorded. But when it is enabled, this variable ensures that this data is included in reports that explicitly correlate externally authenticated user names with their associated user roles and accounts. Specifically, the Top Content Items by User Role report and the Users by User Role report include all of the content item access activity by external users. For more information, see "Creating Custom Report Queries" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

Note: To manually disable the SctExternalUserLogEnabled configuration variable, see Section 14.2.2, "Setting Content Tracker Configuration Variables."

14.5 Tracking Indirect Access to Content with Web Beacons

Important: The implementation requirements for the web beacon feature are contingent on the system configurations involved. All of the factors cannot be addressed in this documentation. Information about the access records collected and processed by Content Tracker are an indication of general user activity and not exact counts.

A web beacon is a managed object that facilitates specialized tracking support for indirect user accesses to web pages or other managed content. In earlier releases, Content Tracker was unable to gather data from cached pages and pages generated from cached services. When users accessed cached web pages and content items, Oracle WebCenter Content Server and Content Tracker were unaware that these requests ever happened. Without using web beacon referencing, Content Tracker does not record and count such requests.

The web beacon involves the use of client side embedded references that are invisible references to the managed beacon objects within Oracle WebCenter Content Server. This enables Content Tracker to record and count user access requests for managed content items that have been copied by an external entity for redistribution without obtaining content directly from Oracle WebCenter Content Server. For details about circumstances when this might be used, see Section 14.5.1, "Web Beacon Use Cases."

When cached content is served to consumers, users perceive that the requested object was served by Oracle WebCenter Content Server. The managed content is actually provided using non-dynamic content delivery methods. In these situations, the

managed content is served by a static website, a reverse proxy server, or out of a file system. The web beacon feature ensures that this type of activity can be tracked.

14.5.1 Web Beacon Use Cases

Two situations in particular may merit the use of the web beacon functionality: reverse proxy activity and when using Site Studio.

In a reverse proxy scenario, the reverse proxy server is positioned between the users and Oracle WebCenter Content Server. The reverse proxy server caches managed content items by making a copy of requested objects. The next time another user asks for the document, it displays its copy from the private cache. If the reverse proxy server does not already have the object in its cache, it requests a copy.

Because it is delivering cached content, the reverse proxy server does not directly interact with Oracle WebCenter Content Server. Therefore, Content Tracker cannot detect these requests and does not track this type of user access activity.

A reverse proxy server is often used to improve web performance by caching or by providing controlled web access to applications and sites behind a firewall. Such a configuration provides load balancing by moving copies of frequently accessed content to a web server where it is updated on a scheduled basis.

For the web beacon feature to work, each user access includes an additional request to the managed beacon object in Oracle WebCenter Content Server. This adds overhead to normal requests, but the web beacon object is very small and does not significantly interfere with the reverse proxy server's performance. Note that it is only necessary to embed the web beacon references in objects you specifically want to track.

Another usage scenario involves Site Studio, a product that is used to create websites which are stored and managed in Oracle WebCenter Content Server. When Site Studio and Oracle WebCenter Content Server are located on the same server, Content Tracker is configured to automatically track the applicable user accesses. The gathered Site Studio activity data is then used in predefined reports. For more information, see "Site Studio Website Activity Reporting" in the *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

Note: Two modes of Site Studio integration are available with Content Tracker. One type is the existing built-in integration that automatically occurs when Site Studio is installed. This is typically used when a website is under construction and the web pages are managed in Oracle WebCenter Content Server.

The other form uses the web beacon feature and Content Tracker regards Site Studio the same as any other website generator. This is typically used when a website is in production mode and content is no longer managed in Oracle WebCenter Content Server.

If your website is intended for an external audience, you may decide to create a copy of the site and transfer it to another server. In addition to being viewed publicly, this solution also ensures that site development remains separate from the production site. In this arrangement, however, implement the web beacon feature to make sure that Content Tracker can collect and process user activity.

14.5.2 Web Beacon Overview

Content Tracker records and counts requests for objects managed by Oracle WebCenter Content Server. The web beacon feature counts requests for managed objects copied by an external entity (such as a reverse proxy server or other functions not involving Oracle WebCenter Content Server).

The following list provides a brief overview of the web beacon feature's functionality and implementation requirements.

- To begin implementing the web beacon feature, create a Web Beacon object. This is usually a small object such as a 1x1 pixel transparent image. The object is then checked in and added to Content Tracker's list of web beacon object names.
- Next create the Web Beacon references to the checked-in web beacon object and embedding them into cached HTML pages or managed content items. The first part of the reference is a URL reference to the web beacon object and the second part is identification information encoded as pseudo query parameters.
- Content Tracker logs the web beacon reference to the beacon object and performs Reduction Processing for Web Beacon references. During data reduction, Content Tracker checks the dDocName of each referenced object against the list of registered web beacons. If the dDocName is on the list, the query parameters are processed in such a manner to ensure that the URL request is logged as a request for the tagged object (web page or managed content item) rather than the web beacon object.

14.5.3 Web Beacon Object

One or more content items must be created to use as the web beacon object(s). These are usually a 1x1 pixel transparent image or anything with low overhead that won't disrupt the page being rendered. The ideal web beacon object has zero content. Multiple web beacon objects can be created but only one is required. Make sure the object is not a file type included in the `SctIgnoreFileType` configuration variable.

Check in the completed object then update Content Tracker's `SctWebBeaconIDList` configuration variable. During data reduction, Content Tracker checks the `SctWebBeaconIDList` settings to determine how the web beacon reference listings in the event logs should be processed. If the applicable web beacon object is listed, Content Tracker processes the data appropriately. See the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content* for details about configuration variables.

During installation the dDocNames of web beacon objects can be entered into the SctWebBeaconIDList preference variable or it can be added or edited later. Follow these steps to add or edit object names in the ID list:

1. Choose **Administration** then **Admin Server** from the Main menu.

   The Content Admin Server page opens.

2. Click the name of the instance whose web beacon preference setting will be changed.

   The Content Admin Server *instance_name* page opens.

3. Click **Component Manager**.

   The Component Manager page opens.

4. In the **Update Component** configuration field, choose Content Tracker from the list.

5. Click **Update.**

   The Update Component Configuration page opens.

6. In the SctWebBeaconIDList preference field, enter the applicable web beacon object dDocNames separated by commas.

7. Click **Update**.

8. Restart Oracle WebCenter Content Server to apply the changes.

14.5.4 Web Beacon References

After creating and checking in the web beacon object(s), create their corresponding reference(s). A single web beacon object works in most systems because different query strings appended to the web beacon static URL make each reference unique. Each query parameter set also consists of distinct combinations of variables that identify specific cached web pages or managed content items.

14.5.4.1 Format Structure for URL References

Web beacon URL references consist of the web beacon static URL used to access the web beacon object managed by Oracle WebCenter Content Server and a pseudo query string with content item variables.

When creating the references, make sure the web beacon static URL in Oracle WebCenter Content Server does not use a directory root that is included in the `SctIgnoreDirectories` configuration variable. If the URL is one of the listed values, Content Tracker does not collect the activity data. For more information about the `SctIgnoreDirectories` configuration variable, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

The query parameter set functions as a code that informs Content Tracker what the actual managed content item is that the user accessed. One of the query parameters is the item's `dID`. Including a unique set of query parameter values allows monitoring of indirect user access activity for managed objects that have been copied and cached. The query string is never actually executed but the query parameter values provide information for Content Tracker to be able to identify the associated managed object.

The following examples illustrate general format structures associated with the web beacon feature. The examples demonstrate how to use one web beacon object while creating an unbounded number of different query strings. The same web beacon object (dDocName = `bcn_2.txt`) is used in all of the examples. By varying the query parameters, the requests for this web beacon object can convey to Content Tracker a 'request' for any managed object in the repository.

These examples have the following assumptions:

- The web beacon object (`bcn_2.txt`) is checked in and is included in the web beacon list (SctWebBeaconIDList).
- The applicable web beacon references are embedded into the associated managed content items (doc1, doc2, and doc3).
- To resolve the web beacon reference, the browser must request a copy of the web beacon object from Oracle WebCenter Content Server.
- The web beacon requests occur because users are indirectly requesting the related content items.

Example 14–1 Web Beacon Request Without Query Parameters

```
http://myhost.somewhere.else/idc/groups/public/documents/adacct/bcn_2.txt
```

This begins with a static web reference to the web beacon object. Although it is a legitimate direct access to the web beacon object, there are no appended query parameters. Content Tracker processes this access event as a request for the web beacon object itself.

Example 14–2 Web Beacon Request for Tracking doc1

```
http://myhost.somewhere.else/idc/groups/public/documents/adacct/bcn_2.txt?sct_
dDocName=doc1&sct_dID=1234&...
```

This also begins with the usual static web reference to this beacon object. It has a pseudo query string appended to it that contains an arbitrary number of query parameters. The values contained in these query parameters convey the information about the specific managed object (`doc1`) the user has requested.

Example 14–3 Web Beacon Request for Tracking doc2

```
http://myhost.somewhere.else/idc/groups/public/documents/adacct/bcn_2.txt?sct_
dDocName=doc2&sct_Ext_2=WebSite4
```

This is similar to Example 14–4. The parameter values provide information about the user requested content item (`doc2`). In this example the query string includes another parameter to convey additional information about the tagged item. The added parameter uses an extField column name. The value `WebSite4` is copied into the extField_2 column of the SctAccessLog table. The extField column substitution is optional and application dependent.

Example 14–4 Web Beacon Request for Tracking doc3

```
http://myhost.somewhere.else/idc/groups/public/documents/adacct/bcn_2.txt?sct_
dDocName=doc2&sct_Ext_2=WebSite4&sct_Ext_8=SubscriptionClass6
```

This example modifies Example 14–3 by adding a second (although non-sequential) extField column name. In this case, `WebSite4` is copied into the extField_2 column of the SctAccessLog table, and `SubscriptionClass6` is copied into the extField_8 column. The extField column substitutions are optional and application dependent.

14.5.4.2 Placement and Retrieval Scheme

The specially constructed web beacon references must be embedded in the managed object to track. Web beacon references can be embedded in any HTML page. Users indirectly request access to the modified *managed* content items through an external Site Studio website or a reverse proxy server.

The browser encounters the web beacon reference while rendering the page. Each display of the managed object, regardless of how the object was obtained, causes the browser to request a copy of the web beacon object directly from Oracle WebCenter Content Server. When the browser resolves the web beacon reference, Content Tracker captures the data that includes the web beacon reference with the set of pseudo query parameters that identify the managed content item.

14.5.4.3 Data Capture and Storage

Ordinarily, query parameters in static URLs serve no function for the web browser. But when resolving the web beacon static URL, the browser ignores the appended query parameters long enough for Content Tracker's web server filter plugin to record them.

Although the pseudo query string is never executed, Content Tracker captures the query parameter values with other data such as the client IP address and date/time stamp. Content Tracker records the data in web access event logs.

14.5.5 Reduction Processing for Web Beacon References

When these specially constructed web beacon references are processed during data reduction, Content Tracker compares the web beacon's dDoc Name to the list of dDocNames in the SctWebBeaconID list to determine if the request was for a web beacon object rather than a regular content item.

If there is no match or if no query parameters are appended to the web beacon reference, Content Tracker processes the access event normally. If the web beacon's dDocName is identified, Content Tracker continues to process and interpret the associated URL query parameters with the data reduction process treating the web beacon access request as a request for the web page or content item.

During data reduction, Content Tracker completes the processing by parsing the query parameters and performing various value substitutions for fields ultimately written to the SctAccessLog. The query parameter values are mapped as follows:

- `sct_dID` replaces the web beacon object's dID
- `sct_dDocName` replaces the web beacon object's dDocName
- `sct_uriStem` replaces the web beacon object's URI stem (everything preceding the '?')
- `sct_uriQuery` replaces the web beacon object's URI query portion (everything following the '?')
- `sct_Ext_n` is copied directly into the SctAccessLog Extended Field n

Example 14–5 Data Reduction Processing for Query Parameter Values

```
/idc/groups/public/documents/adacct/bcn_2.txt?sct_dDocName=WW_1_21&sct_dID=42&sct_
Ext_1=WillowStreetServer&sct_Ext_2=SubscriptionTypeA
```

After data reduction, Content Tracker records this web beacon type request in the SctAccessLog table as an access to `WW_1_21` rather than to `bcn_2.txt`. Other data, such as the user name, time of access, and client IP, is derived from the HTTP request. Additionally, `WillowStreetServer` is copied into the extField_1 column of the SctAccessLog table, and `SubscriptionTypeA` is copied into the extField_2 column. (These last two field substitutions are optional and application dependent.)

14.5.6 Limitations and Guidelines

Perform the following tasks to implement Content Tracker's web beacon feature:

1. Create the web beacon object.
2. Check it in.
3. Update the SctWebBeaconIDList.
4. Define the web beacon references.
5. Embed them into the cached content items, websites, or both to track.

14.5.6.1 Limitations

The following limitations should be considered:

- One difficulty is determining the means by which the web beacon reference is attached to a tagged object. There are situations where the requested object does not allow embedded references (for example, a PDF or Word document). In this case, the web beacon object must be requested directly from Oracle WebCenter Content Server before the actual content item is requested.
- The web beacon feature does not work in many situations, such as with certain browser configurations. If the user has disabled cross-domain references in their browser, and both the web page and Oracle WebCenter Content Server instance are in different domains, the web beacon object is never requested from Oracle WebCenter Content Server and the user access is not counted.
- The first time a managed content item is accessed through a reverse proxy server, it is counted twice: once when the Oracle WebCenter Content Server provides the item to the reverse proxy server, and a second time when the browser requests the web beacon object.
- Depending on the specific configuration, it might be necessary to devise a method to prevent the reverse proxy server and external Site Studio from caching the web beacon object itself. Browsers also do caching. This situation would prevent Content Tracker from counting any relevant content accesses. To avoid this append a single-use query parameter to the web beacon reference that contains a random number as in this example:

  ```
  dDocName=vvvv_1_21&FoolTheProxyServer=12345654321
  ```

  By changing the number on each request, the cache, web server and the browser view the request as new.

14.5.6.2 Guidelines

The following guidelines should be considered:

- The sct_dDocName and sct_dID parameter values in the web beacon reference must resolve to an actual managed content item in the same Oracle WebCenter Content Server instance that provides the requested web beacon object.
- Using the ExtField columns in the SctAccssLog table is optional and application dependent.
- Use of ExtField_10 is reserved for the web beacon object's dDocName. This allows report writers a way to determine which web beacon object was used to signal the access to the actual managed content item.
- Spelling and capitalization of the query parameter names must be exact.
- Embedded commas or spaces in the query parameter values are not allowed.
- The dDocName and dID of a managed object are usually included in the web beacon reference although to be considered a legitimate access request, it is not necessary to provide both. If any of the standard fields are missing, Content Tracker resolves the identification parameters as follows:
  - Given a dID, Content Tracker can determine the content item's dDocName.
  - Given a dDocName, Content Tracker can determine the content item's dID. The dID is the content item's most current revision. If the revision changes after the content item is cached, then the user sees the older version. However,

Content Tracker counts this access request as a view of the most recent revision of the content item.

  - Given a proper URI Stem, Content Tracker can determine the content item's dDocName but assumes the dID of the most recent revision.

- Restart Oracle WebCenter Content Server after making changes to the web beacon list (SctWebBeaconIDList).
- Do not create a web beacon object that uses a file type or is located in a directory that Content Tracker is configured to disregard.
- Content Tracker is unable to verify if the cached content item was delivered.
- Content Tracker performs normal folding of static URL accesses. If a user repeatedly requests the same content item and makes no intervening requests for another document, then Content Tracker assumes that the consecutive requests are the same document. In this case, these access requests are considered to be all one access request.
- The query parameters can represent any managed object and need not necessarily be what the user is actually viewing.

14.5.7 Examples of Web Beacon Embedding

Several embedding methods can be used to implement the web beacon feature. Each technique has advantages and disadvantages and one may be more appropriate for a particular situation than another. Because of differences in system configurations, there is no optimal single technique.

All of the examples below use the following information:

- WebBeacon.bmp web beacon object
- Oracle WebCenter Content Server instance IFHE.comcast.net/idc/
- dDocName of wb_bmp

Code fragment files for all of the examples are included in Content Tracker's documentation directory. These examples are intended to demonstrate general approaches and are provided as a starting point. They will need to be adapted to work with your specific application and network topology.

14.5.7.1 Embedded HTML Example

The simplest, most direct use of a web beacon for tracking managed content access is to embed a reference to the beacon directly into the HTLM source for the containing web page. When the requesting user's browser attempts to render the page, it sends a request to the instance where the web beacon object resides.

In this example, the technique places an image tag in the web page to be tracked. The `src attribute` of the image refers to the web beacon object (`wb_bmp`) which was checked into an instance. When the user's browser loads the image the instance, the additional query information is recorded and ultimately interpreted as a reference to dDocName `BOPR`.

This approach is simple but has the disadvantage that the user's browser or a reverse proxy server, might cache a copy of the web beacon object. As such, no additional requests are posted directly to the instance and no additional accesses to any content tagged with this web beacon are counted.

The HTML fragment for this method might be written as follows:

```
<!-- WebBeaconEmbeddedHtml.htm - Adjust the Web Beacon web location and managed
object identfiers in the img src attribute, then paste into your web page -->
<img src="http://IFHE.comcast.net/idc/groups/public/documents/adacct/wb_
bmp.bmp?sct_dID=1&sct_dDocName=BOPR&sct_
uriStem=http://IFHE.comcast.net/idc/groups/public/documents/adacct/bopr.pdf&sct_
Ext_1=Sample_Html_Beacon_Access" width="21" height="21" />
```

14.5.7.2 Embedded JavaScript Example

The cached web beacon problem can be overcome by using JavaScript instead of HTML Using the embedded JavaScript method requires two script tags:

- The `cs_callWebBeacon` function that issues the actual web beacon request.
- An unnamed block that assigns context values to certain JavaScript variables, then calls the `cs_callWebBeacon` function.

The identifying information for the managed content object is defined in a list of variables which improves readability. The web beacon request is also made effectively unique by adding a random number to the pseudo query parameters.

Disadvantages include more code to manage and the URL of the web beacon server is hard coded in each web page. In addition, the user's browser might not have JavaScript enabled.

The JavaScript fragment for this method might be written as follows:

```
// WebBeaconEmbeddedJavascript.js - Adjust the managed object and Web Beacon
descriptors,
then paste this into your web page.
//

<script type="text/javascript" >

    var cs_obj_dID = "" ;
    var cs_obj_dDocName = "" ;
    var cs_obj_uriStem = "" ;
    var cs_extField_1 = "" ;
    var cs_extField_2 = "" ;
    var cs_extField_3 = "" ;
    var cs_extField_4 = "" ;
    var cs_extField_5 = "" ;
    var cs_extField_6 = "" ;
    var cs_extField_7 = "" ;
    var cs_extField_8 = "" ;
    var cs_extField_9 = "" ;
    var cs_beaconUrl = "" ;

    function cs_void( ) { return ; }

    function cs_callWebBeacon( ) {
        //
        var cs_imgSrc = "" ;
        var cs_inQry = false ;

        if ( cs_beaconUrl && cs_beaconUrl != "" ) {
            cs_imgSrc += cs_beaconUrl ;
        }

        if ( cs_obj_dID && cs_obj_dID != "" ) {
```

```
    if ( cs_inQry ) {
        cs_imgSrc += "&" ;
    } else {
        cs_imgSrc += "?" ;
        cs_inQry = true ;
    }
    cs_imgSrc += "sct_dID=" + cs_obj_dID ;
}

if ( cs_obj_dDocName && cs_obj_dDocName != "" ) {
    if ( cs_inQry ) {
        cs_imgSrc += "&" ;
    } else {
        cs_imgSrc += "?" ;
        cs_inQry = true ;
    }
    cs_imgSrc += "sct_dDocName=" + cs_obj_dDocName ;
}
if ( cs_obj_uriStem && cs_obj_uriStem != "" ) {
    if ( cs_inQry ) {
        cs_imgSrc += "&" ;
    } else {
        cs_imgSrc += "?" ;
        cs_inQry = true ;
    }
    cs_imgSrc += "sct_uriStem=" + cs_obj_uriStem ;
}

if ( cs_extField_1 && cs_extField_1 != "" ) {
    if ( cs_inQry ) {
        cs_imgSrc += "&" ;
    } else {
        cs_imgSrc += "?" ;
        cs_inQry = true ;
    }
    cs_imgSrc += "sct_Ext_1=" + cs_extField_1 ;
}

if ( cs_extField_2 && cs_extField_2 != "" ) {
    if ( cs_inQry ) {
        cs_imgSrc += "&" ;
    } else {
        cs_imgSrc += "?" ;
        cs_inQry = true ;
    }
    cs_imgSrc += "sct_Ext_2=" + cs_extField_2 ;
}

<!-- and so on for the remaining extended fields -->

if ( cs_inQry ) {
    cs_imgSrc += "&" ;
} else {
    cs_imgSrc += "?" ;
    cs_inQry = true ;
}

var dc = Math.round( Math.random( ) * 2147483647 ) ;
cs_imgSrc += "sct_defeatCache=" + dc ;
```

```
        var wbImg = new Image( 1, 1 ) ;
        wbImg.src = cs_imgSrc ;
        wbImg.onload = function( ) { cs_void( ) ; }

    }

</script>

<script type="text/javascript">
    //
    var cs_obj_dID = "1" ;
    var cs_obj_dDocName = "BOPR" ;
    var cs_obj_uriStem =
"http://IFHE.comcast.net/idc/groups/public/documents/adacct/bopr.pdf" ;
    var cs_extField_1 = "Sample_Javascript_Beacon_Access" ;
    var cs_beaconUrl =
"http://IFHE.comcast.net/idc/groups/public/documents/adacct/wb_bmp.bmp" ;

    cs_callWebBeacon( ) ;

</script>
```

14.5.7.3 Served JavaScript Example

The hard-coded web beacon server problem described in the Embedded JavaScript Example can be overcome by splitting the code into two fragments:

- The managed code fragment contains the `cs_callWebBeacon` function. It can be checked in and managed by an Oracle WebCenter Content Server instance, either the instance that manages the web beacon or some other instance. The `src` attribute contained in the in-page code fragment refers to the managed code fragment and causes it to be dynamically loaded into the web page.
- The in-page code fragment still consists of two `<script>` tags, but the first contains only a reference to the `cs_callWebBeacon` code instead of the code itself. The advantage for this is that changes to the `cs_callWebBeacon` function can be managed centrally instead of having to modify each and every tagged web page.

  This solution incurs the additional network overhead of loading the managed code into the web page on the user's browser. However, the requirement for a web beacon assist to tracking implies that the network environment includes an efficient reverse proxy server, or other caching mechanism. The same cache that conceals managed object access also minimizes the impact of the code download.

Managed Code Fragment

```
// WebBeaconServedJavascript_Checkin.js - Check this in to your Content Server,
then fixup
// the JavaScript include src attribute in
WebBeaconManagedJavascriptIncludeSample.js
//
    var cs_obj_dID = "" ;
    var cs_obj_dDocName = "" ;
    var cs_obj_uriStem = "" ;
    var cs_extField_1 = "" ;
    var cs_extField_2 = "" ;
    var cs_extField_3 = "" ;
    var cs_extField_4 = "" ;
    var cs_extField_5 = "" ;
    var cs_extField_6 = "" ;
```

```
    var cs_extField_7 = "" ;
    var cs_extField_8 = "" ;
    var cs_extField_9 = "" ;
    var cs_beaconUrl =
"http://IFHE.comcast.net/idc/groups/public/documents/adacct/wb_bmp.bmp" ;

    function cs_void( ) { return ; }

    function cs_callWebBeacon( ) {
        //
        var cs_imgSrc = "" ;
        var cs_inQry = false ;

        if ( cs_beaconUrl && cs_beaconUrl != "" ) {
            cs_imgSrc += cs_beaconUrl ;
        }

        if ( cs_obj_dID && cs_obj_dID != "" ) {
            if ( cs_inQry ) {
                cs_imgSrc += "&" ;
            } else {
                cs_imgSrc += "?" ;
                cs_inQry = true ;
            }
            cs_imgSrc += "sct_dID=" + cs_obj_dID ;
        }

        if ( cs_obj_dDocName && cs_obj_dDocName != "" ) {
            if ( cs_inQry ) {
                cs_imgSrc += "&" ;
            } else {
                cs_imgSrc += "?" ;
                cs_inQry = true ;
            }
            cs_imgSrc += "sct_dDocName=" + cs_obj_dDocName ;
        }

        if ( cs_obj_uriStem && cs_obj_uriStem != "" ) {
            if ( cs_inQry ) {
                cs_imgSrc += "&" ;
            } else {
                cs_imgSrc += "?" ;
                cs_inQry = true ;
            }
            cs_imgSrc += "sct_uriStem=" + cs_obj_uriStem ;
        }

        if ( cs_extField_1 && cs_extField_1 != "" ) {
            if ( cs_inQry ) {
                cs_imgSrc += "&" ;
            } else {
                cs_imgSrc += "?" ;
                cs_inQry = true ;
            }
            cs_imgSrc += "sct_Ext_1=" + cs_extField_1 ;
        }

        if ( cs_extField_2 && cs_extField_2 != "" ) {
            if ( cs_inQry ) {
                cs_imgSrc += "&" ;
```

```
            } else {
                cs_imgSrc += "?" ;
                cs_inQry = true ;
            }
            cs_imgSrc += "sct_Ext_2=" + cs_extField_2 ;
        }

        <!-- and so on for the remaining extended fields -->

        if ( cs_inQry ) {
            cs_imgSrc += "&" ;
        } else {
                cs_imgSrc += "?" ;
                cs_inQry = true ;
        }

        var dc = Math.round( Math.random( ) * 2147483647 ) ;
        cs_imgSrc += "sct_defeatCache=" + dc ;

        var wbImg = new Image( 1, 1 ) ;
        wbImg.src = cs_imgSrc ;
        wbImg.onload = function( ) { cs_void( ) ; }

    }
```

In-Page Code Fragment

```
<script type="text/javascript"
src="http://IFHE.comcast.net/idc/groups/public/documents/adacct/wbmjcs.js" >
</script>

<script type="text/javascript">
    //
    var cs_obj_dID = "1" ;
    var cs_obj_dDocName = "BOPR" ;
    var cs_obj_uriStem =
"http://IFHE.comcast.net/idc/groups/public/documents/adacct/bopr.pdf" ;
    var cs_extField_1 = "Sample_Managed_Javascript_Beacon_Access" ;

    cs_callWebBeacon( ) ;

</script>
```

15

Customizing Content Categorizer

This chapter provides information about the customization of Content Categorizer, an optional component that is automatically installed with Oracle WebCenter Content. When enabled, Content Categorizer suggests metadata values for documents being checked into Oracle WebCenter Content Server.

15.1 About Content Categorizer

Content Categorizer suggests metadata values for new documents being checked into Content Server, and for existing documents that may or may not already have metadata values. These metadata values are determined according to search rules provided by the System Administrator.

For Content Categorizer to recognize structural properties, the content must go through XML Conversion (eXtensible Markup Language).

The Content Categorizer Batch utility can search a large number of files and create a Batch Loader control file containing appropriate metadata field values. The Batch utility can also be used to recategorize existing content (already checked into the repository).

15.2 Setting Up and Customizing Content Categorizer for Your Site

To customize Content Categorizer for your site, you can set the XML conversion method, define field properties for the metadata fields, and define search rules for each file type. You can also define your own eXtensible Style Sheet Language Transformations (XSLT) for the XML translation, to accommodate your site's document processing needs.

For details about setting up Content Categorizer and customizing it, see "Categorizing and Linking Content" in *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

16

Downloading Custom Components

This chapter describes how to download packaged custom components to Oracle WebCenter Content Server.

This chapter includes the following sections:

16.1 About Downloading Custom Components

You can download custom components for Content Server with the Advanced Component Manager or from Oracle Technology Network.

16.2 Downloading a Component from the Advanced Component Manager

You can use the Advanced Component Manager to download a component for Content Server.

To download a component from the advanced component manager:

1. In the **Administration** tray or menu, choose **Admin Server**.

   The Admin Server displays the Component Manager page.

2. In the first paragraph on the Component Manager page, click **advanced component manager**.

   This displays the Advanced Component Manager page, which has a list of components available for downloading.

3. Choose the component to be packaged from the **Download Component** list
4. Click **Download** to display the File Download screen.
5. Select **Save this file** to disk, and click **OK**.
6. In the Save As dialog box, navigate to a directory, change the file name if necessary, and click **Save**.

16.3 Downloading a Component from Oracle Technology Network

You can download a component for Content Server from Oracle Technology Network (OTN).

To download a component from OTN:

1. In a web browser, go to the OTN website at `http://www.oracle.com/technetwork/middleware/downloads/index.html`.
2. On the Oracle Fusion Middleware 11*g* Software Downloads page, click **WebCenter Content** on the left.
3. On the Oracle WebCenter Content page, click **Core Capabilities**.
4. On the Oracle WebCenter Content Core Capabilities page, click **Download**.
5. On the Downloads page, click **Individual UCM Component Downloads**, and download the component you want.

17

Creating Custom Components

This chapter describes how to create custom components to use with Oracle WebCenter Content Server.

This chapter includes the following sections:

17.1 About Creating Custom Components

Custom components can alter defaults for your system, add new functionality, or streamline repetitive functions. You can create and use custom components to modify a Content Server instance without compromising the system integrity.

17.2 Creating Resources for a Component

You can use the following types of resources to customize Content Server:

- HTML includes
- Dynamic data tables
- String resources
- Dynamic tables
- Static tables
- Queries
- Services
- Templates
- Environment resources

17.2.1 HTML Includes

An include is defined within <@dynamichtml *name*@> and <@end@> tags in an HTM resource file. The include is then called using this syntax:

<$include *name*$>

Includes can contain Idoc Script and valid HTML code, including JavaScript, Java applets, cascading style sheets, and comments. Includes can be defined in the same file as they are called from, or they can be defined in a separate HTM file. Standard HTML includes are defined in the *IdcHomeDir*/resources/core/idoc files.

HTML includes, strings, and static tables can be present in the same HTM file. An HTML include resource does not require merge rules.

17.2.1.1 The Super Tag

The super tag is used to define exceptions to an existing HTML include. The super tag tells the include to start with an existing include and then add to it or modify using the specified code.

The super tag is particularly useful when making a small customization to large includes or when you customize standard code that is likely to change from one software version to the next. When you upgrade to a new version of Oracle WebCenter Content Server, the super tag ensures that your components are using the most recent version of the include, modifying only the specific code you need to customize your instance.

The super tag uses the following syntax:

```
<@dynamichtml my_resource@>
    <$include super.my_resource$>
    exception code
<@end@>
```

You can use the super tag to refer to a standard include or a custom include. The super tag incorporates the include that was loaded last.

Note: The placement of a super tag will determine how the Idoc Script is evaluated.

Example 17–1 shows the use of a super tag.

Example 17–1 super Tag

In this example, a component defines the my_resource include as follows:

```
<@dynamichtml my_resource@>
    <$a = 1, b = 2$>
<@end@>
```

Another component that is loaded later enhances the my_resource include using the super tag. The result of the following enhancement is that a is assigned the value 1 and b is assigned the value 3:

```
<@dynamichtml my_resource@>
    <$include super.my_resource$>
    <!--Change "b" but not "a" -->
    <$b = 3$>
<@end@>
```

17.2.1.2 Editing an HTML Include Resource

Use the following procedure to edit an existing HTML include resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource in the **Custom Resource Definition** list.
3. If the resource file contains multiple types of resources, click the **Includes** tab on the right.
4. Modify the includes in the **Custom HTML Includes** list.
   - To edit an existing include, choose the include, click **Edit**, modify the code, and then click **OK**.
   - To add an include to the resource file, click **Add**.
   - To remove an include, choose the include, click **Delete**, and then click **Yes** to confirm.

17.2.2 Dynamic Data Tables

A dynamic data table resource is a `dynamicdata` table. This type of resource enables you to define tables of data from within Idoc Script to load an HTML table definition, interface menu actions, or information about metadata fields or from within Java code as an alternative to static tables loaded into SharedObjects.

While tables loaded into SharedObjects are static and rarely change, a lot of code within Content Server will modify the contents of a `dynamicdata` table when it is loaded into a user's context. You can use `dynamicdata` resources to display different data to users depending on anything from their security attributes to the specific actions they are performing. Components can do targeted merging into tables created with this resource type, and Idoc Script pages can select and sort rows.

You can declare a `dynamicdata` resource in any resource file that can contain `dynamichtml` constructions, as Example 17–2 shows.

Example 17–2 dynamicdata Resource

```
<@dynamicdata NameOfTable@>
<?formatoftable properties-of-table?>
table-data
<@end@>
```

A `dynamicdata` table is defined within `<@dynamicdata name@>` and `<@end@>` tags in a resource file. To reference `dynamicdata` tables, you need to use the Idoc Script functions whose names begin with `dd`, such as `ddLoadResultSet`, which loads a merged `dynamicdata` table and creates a ResultSet in the current data binder.

The *IdcHomeDir*`/resources/core/idoc` files define standard `dynamicdata` resources.

17.2.2.1 Specifying Table Formats

For the *formatoftable* parameter in a `dynamicdata` resource, you can specify either of two format types:

- `commatable`
- `htmltable`

The default format is `commatable`.

commatable

The `commatable` format is for tables with values that do not have line feeds or carriage returns. In this format, you enter a comma-separated list of field names on one line followed by a comma-separated list of values on the following lines, one line for each field, as Example 17–3 shows.

Example 17–3 commatable Format

```
<@dynamicdata SampleTable@>
<?commatable?>
col1,   col2
val1_1, val1_2
val2_1, val2_2
<@end@>
```

If you need to insert a comma (`,`) into a value, then use a circumflex (`^`) instead of the comma. If you need to insert a circumflex, then enter the escape sequence hash-circumflex (`#^`, and if you need to insert a hash (`#`) that is followed by a hash or a circumflex, then enter the escape sequence hash-hash (`##`), as Example 17–4 shows.

Example 17–4 Special Characters in Values

```
<@dynamicdata SampleTable@>
field1, field2
A^B,    C##^D#^E#F^G
<@end@>
```

This `dynamicdata` resource would load a table row whose value for `field1` would be `A,B` and for `field2` would be `C#^D^E#F,G`.

You cannot escape line feeds or carriage returns. If you need to specify a value that contains either of those characters, then you should use the `htmltable` format.

htmltable

The `htmltable` format is the same as the format used for static HTML table constructs in Content Server, as Example 17–5 shows.

Example 17–5 htmltable Format

```
<@dynamicdata SampleTable@>
<?htmltable?>
<table>
<tr>
        <td>col1</td>
        <td>col2</td>
</tr>
<tr>
        <td>val11</td>
        <td>val12</td>
</tr>
<tr>
        <td>val21</td>
        <td>val22</td>
</tr>
</table>
<@end@>
```

17.2.2.2 Editing a Dynamic Data Table Resource

Use the following procedure to edit an existing `dynamicdata` resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource from the **Custom Resource Definition** list.
3. If the resource file contains multiple types of resources, click the **Includes** tab on the right.
4. You can modify any of the `dynamicdata` tables in the custom resource definition, add a `dynamicdata` table, or remove a `dynamicdata` table:
   - To edit an existing `dynamicdata` table, choose the table, click **Edit**, modify the code, and then click **OK**.
   - To add a `dynamicdata` table to the resource file, click **Add**.
   - To remove a `dynamicdata` table, choose the table, click **Delete**, and then click **Yes** to confirm.

17.2.2.3 Specifying Table Properties

The `properties-of-table` parameter in a `dynamicdata` resource has this format:

`field1="value1" field2="value2" . . .`

The properties are like attributes defined in an XML node. Example 17–6 shows a typical table declaration.

Example 17–6 Table Properties in a Table Definition

```
<@dynamicdata ExampleTable@>
<?commatable mergeField="fieldA" indexedColumns="fieldA,fieldB"?>
fieldA, fieldB
1,      2
3,      4
<@end@>
```

The quotation marks that enclose the values are optional for values that have no spaces, and you can use either single or double quotation marks. Also, the default property value is `"1"`, so you can omit the assignment of a value for a table property if it is `"1"`.

Omitting the value is useful for Boolean properties such a `notrim` and `mergeBlanks`. Example 17–7 shows a declaration specifying a table that is not to trim its values:

Example 17–7 notrim Property

```
<@dynamicdata ExampleTable@>
<?commatable mergeField="fieldA" indexedColumns="fieldA,fieldB" notrim?>
fieldA, fieldB
1, 2
3, 4
<@end@>
```

In this example, the space would not be trimmed before the `2` or the `4`. (Field names are always trimmed.)

You can specify the following kinds of table properties:

- Merge properties
- Assembly properties
- Sort properties
- Filter and `dynamicdata` table properties

17.2.2.3.1 Merge Properties The `dynamicdata` tables can be merged together automatically, which is part of the power of using these tables. If two `dynamicdata` tables have the same name but are in separate resource files, they will be automatically merged. You can use the `mergeOtherData` option to merge another existing table into the current existing table. Using this technique, you can build very complicated tables all merged from various other tables. This merging can improve the readability of the data and enable you to have some tables as subsets of other tables.

You can specify one or more of the following merge properties in the `properties-of-table` parameter in a `dynamicdata` resource:

- **mergeKey** -- The name of the field to do a merge on. This value applies to both this and the existing tables when doing an overlay unless mergeNewKey is set in which case it only applies to the existing table. If this value is not set, then the merge key defaults to the first column of this table. If the mergeKey refers to a column in the existing table that does not exist, then the result will be to append this table to the existing table unless the mergeRule is set to a value that dictates a different outcome. This property has merge scope.
- **mergeNewKey** -- The name of the field in this table to use as a basis of comparison with the mergeKey column in the existing table. The default is to be the value of mergeKey. This property has merge scope.
- **mergeRule** -- The rule to use when performing a merge of two tables. This property has three possible values, the default being merge. This property has merge scope.
- **merge** -- Merge using the mergeKey (and if specified, the mergeNewKey) properties to perform the merge.
- **mergenoappend** -- Perform the merge, but do not append any new rows. If there is no valid merge to perform (for example, if the mergeKey does not refer to a valid column in the existing table), then the result is to not perform a merge at all and the overlaying table has no effect on the final result.
- **replace** -- Replace the existing table with this table. This option has the outcome of suppressing any prior table resource. This would be similar to not using the super include in a `dynamichtml` resource.
- **mergeBlanks** -- By default, when values are merged from this table to the existing table, any values that are blank in this table do not replace the overlaid value in the existing table. This allows for targeted replacement of column values in the existing table by this table. But if this option is enabled (set without a value, or set with the value `1` or true), then blanks in this table replace non-blank values in the existing table. The default is `0` (or false) and the property has merge scope.
- **mergeAppendColumns** -- This is a comma-separated list of columns in this table. For any column mentioned in the list, column values in this table for that column do not replace values in the existing table for that column but instead append or replace the new value (using comma as the separator) to the current value. Each of the subvalues in the comma-separated list is assumed to be of the form `key=value` with the `=value` part being optional. If this table has the same key in

its comma-separated list, then that key=value pair will replace the value in the existing table. For example, if the existing table has a column value of the form `a=1,b=2` and this table has the value `b=3,c=4`, then the merged result will be `a=1,b=3,c=4`. This property has merge scope.

- **cssStyleMergeAppendFormat** -- This is a boolean property and changes the separator values used for the mergeAppendColumns property. Normally the value of a field mentioned in the mergeAppendColumns is a comma separated list of name equal value pairs with the equal operator (=) being the assignment operator. If this property is enabled, then the lists separator becomes a semi-colon (;) and the name value pairs use a colon (:) for the assignment. So, instead of the field value looking like `A=1,B=2`, it would be `A:1;B:2`. The default is `false`, and the property has merge scope.
- **wildcard** -- Normally when a merge is performed, the merge test is a case insensitive match comparison. When this option is enabled, the comparison is a standard Content Server wildcard match (* = 0 or more of any character, ? = any single character). Typically the option is used with mergeNewKey being set to a column different from mergeKey and in many cases the mergeKey does not even refer to a valid column in this table. The default is 0 (or false) and the property has merge scope.
- **mergeOtherData** -- A comma separated list of other `dynamicdata` resources to merge into this one. Each of the other `dynamicdata` resources are fully merged before they are merged into this resource (if those other resources also are using mergeOtherData, then those merges are done first -- the code does have recursion detection). If the one of the referenced `dynamicdata` resources has multiple definitions in multiple files, then the merge keys used to merge into this resource are the ones defined that is highest in merge order (the one that is merged into last) for that other resource. If this `dynamicdata` resource (the one that has the mergeOtherData property on it) has multiple definitions in multiple files, the mergeOtherData parameter is produced by merging all the referenced named resources from all the resources in the merge stack. The default is null and has global scope.

17.2.2.3.2 Assembly Properties You can specify one or more of the following assembly properties in the `properties-of-table` parameter in a `dynamicdata` resource:

- **notrim** -- This option only applies to the commatable format. Normally, all the values that are parsed for a table resource are trimmed. Setting this option prevents the values from being trimmed. It is presumed that this will be a rarely used option. The default is 0 (or false) and the property has local table scope.
- **indexedColumns** -- This property lists columns that should be optimized for indexed lookup. Specialized Idoc Script functions exist to take advantage of the any of the indexed columns. When a lookup is done against an indexed column, the column name and a value must be specified. A filtered table consisting of just the rows whose values for the indexed column match (case insensitive) the value passed in to the function is returned. Note that these *indexed* column lookups are all computed at load and merge time and stored in a hash table for fast retrieval. The list of indexed column values for all the overlaying tables are merged together and the index computations are done after the merge is finished. This property has global table scope.
- **countColumn** -- This value specifies a column in the fully merged table into which the values of a row count is put. The count starts at 1 and increments for each row in the table. Any existing values in that column of the merged table are replaced by the count value. This property can be used to create a quick unique key for each

row. The default value for this property is "count", so any table with the column name "count" that does not specify a different countColumn will automatically have counter values put into that column. If the value of this property does not match a column name in the final merged table, then it is ignored. If an overlaying table resource specifies a different countColumn from one specified in a prior table resource, then the overlaying one will be used. The property has global table scope.

- **defaultValues** -- This property specifies a comma-separated list of default values to apply to the table. Each default value in this list is of the format fieldname:value. If the value is an empty string then the colon can be dropped. For example, the string field1:val1,field2:val2,field3 specifies the default value val1 for field1, val2 for field2, and the empty string for field3. A colon can be escaped with a star (*) and a star can be escaped by preceding it with a hash (#). If either a hash or a star follows a hash, then the hash can be escaped by adding another hash (see the similar rule for escaping commas given earlier). If a field specified in a default value construction does not exist in the final merged table, then it is added as a new field and given the default value for all rows in that table. If the field exists, then the default value will override any blank values in that table for that field. The definitions of defaultValues from the newer overlaying tables are collated with the active definition of the existing table. If there is a conflict in the definition of a particular default value, the newer overlaying table wins. The default for this property is null and it has global table scope.
- **derivedColumns** -- This property specifies columns to be built up from values from other columns. The general syntax is a comma separated list of derived column definitions of the form derivedColumnDef1,derivedColumnDef2,... with each column definition being of the form fieldName:sourceField1+sourceField2+.... The fieldName refers to the name of the field to be created and the sourceFieldN refer to fields whose value will be sourced to create the derived column. The derived value will hold the values of the source fields separated by a double colon (::). If the derived column exists and has a non empty value, then it is not replaced. As with the defaultValues property, there is a second pass after the final table is assembled to determine whether any derived values still need to be filled in. The most typical usage for derived columns is to allow one `dynamicdata` resource to use multiple columns for specifying a merge criteria instead of just one. The derived column is used as the target of a merge and is defined in the definition of the existing table. The derived column definitions are inherited into the newer overlaying tables and if there is a conflict in definition of a particular derived column then the newer table's definition wins. Otherwise, the definitions of derived columns from the existing and new tables are collated together. The default value for this property is null and it has global table scope.

17.2.2.3.3 Sort Properties You can specify one or more of the following sort properties in the `properties-of-table` parameter in a `dynamicdata` resource:

- **sortColumn** -- Specifies a column to sort on. If an overlaying table resource specifies a different sortColumn from one specified in a prior table resource, then the overlaying one will be used. If the name of the column does not match any column name in the final merged table, then no sort is performed. The default value is "sortOrder". So, creating a column with this name will cause the table to be automatically sorted. This property has global table scope
- **sortType** -- Specifies what data type should be assumed for the column being sorted. This type applies to both the sortColumn and the sortParentColumn. The values can be "int", "string", or "date". The default value for this property is "int".

Rules for overlaying tables both specifying this property are identical to sortColumn. This property has global table scope.

- **sortOrder** -- Specifies what sort order to use when performing a sort. The possible values are "asc" (for ascending) and "desc" (for descending). The default is "asc". Rules for overlaying tables both specifying this property are identical to sortColumn. This property has global table scope.
- **sortIsTree** -- Specifies whether the sort is actually a tree sort with a sortParentColumn being sorted along with the sortChildColumn. The assumption is that the child to parent row mapping relationship is done by using the child row's value in the sortParentColumn to the find the parent row with a matching value in its sortChildColumn field. The sort is performed so that the top level parents are sorted first, then the children of each parent are sorted as a subgroup for each parent, and so on recursively for all the children of the children. The default value is 0 (or false). Rules for overlaying tables both specifying this property are identical to sortColumn. This property has global table scope.
- **sortParentColumn** -- This value must be specified if the `sortIsTree` option is enabled. If the value of this property is missing or specifies an invalid column, then the `sortIsTree` option is ignored and has no effect. For more information about what it does, see the preceding description of the `sortIsTree` property. The default for the `sortParentColumn` property is null. Rules for overlaying tables both specifying this property are identical to `sortColumn`. This property has global scope.
- **sortChildColumn** -- This value must be specified if the sortIsTree option is enabled. If the value of this property is missing or specifies an invalid column, then the `sortIsTree` option is ignored and has no effect. For more information about what it does, see the preceding description of the `sortIsTree` property. The default for the sortChildColumn property is null. Rules for overlaying tables both specifying this property are identical to sortColumn. This property has global scope.
- **sortNestLevelColumn** -- This value is only available if the sortIsTree option is enabled. If the value of this property references an invalid column then it has no effect. If a valid column is specified, then that column will get an integer value that specifies its nest level (starting at 0). The nest level is defined as the number of immediate parents that have to be traversed before reaching a parent that itself has no parent. The default value for this property is "nestLevel" and it has global scope.

17.2.2.3.4 Filter and Include Properties You can specify one or more of the following filter and include properties in the `properties-of-table` parameter in a `dynamicdata` resource:

- **filterInclude** -- This property specifies an include to be executed for each row of a table (or subtable if an indexed column is being used to select a subtable). This execution will happen when the table is loaded into the current user's context. Its main purpose is either to create a side effect or to determine if the row should be excluded. To prevent the row from being loaded into the final ResultSet, you can set the variable `ddSkipRow` to `1` (`<$ddSkipRow=1$>`). During execution of this include, the table is made *active*, allowing for easy access and replacement of values in the table. The default value of this property is `null`, and it has global scope.
- **includeColumns** -- This property specifies a comma-separated list of columns whose row values are the names of resource includes to be executed. After the resource includes are executed, the result is fed back into the ResultSet to become

the new value for that column for that row. The timing and rules for execution are similar to `filterInclude` except that `includeColumns` cannot suppress the loading of a row. If a filter include is specified and there are active include columns, then during the looping of the temporary active ResultSet, the include column values are executed first and then the filter include. If one of the specified include columns in not present in the final merged table, then it will have no effect. Empty values in an include column are ignored. The `includeColumns` attribute is commonly combined with the `defaultValues` attribute to create columns whose values are derived from other columns. The default value of this property is `null`, and it has global scope.

> **Note:** Using `includeColumns` may not be as useful as it first appears. The resource includes are executed at the time the Idoc Script function is executed to load the table, but a component that customizes output may determine the value for the column only after further processing (after other tables are merged into this table, summaries of row statistics are calculated, and so on).

17.2.2.4 Using Dynamicdata Idoc Script Functions

For dynamic data tables, you can use the following `dynamicdata` Idoc Script functions:

- `ddAppendIndexedColumnResultSet`
- `ddAppendResultSet`
- `ddApplyTableSortToResultSet`
- `ddGetFieldList`
- `ddIncludePreserveValues`
- `ddLoadIndexedColumnResultSet`
- `ddLoadResultSet`
- `ddMergeIndexedColumnResultSet`
- `ddMergeResultSet`
- `ddMergeUsingIndexedKey`
- `ddSetLocal`
- `ddSetLocalByColumnsFromFirstRow`
- `ddSetLocalByColumnsFromFirstRowIndexed`
- `ddSetLocalEmpty`
- `ddSetLocalEmptyByColumns`

17.2.3 String Resources

A `string` resource defines locale-sensitive text strings that are used in error messages and on Content Server web pages and applets. Strings are resolved by Content Server each time a web page is assembled, an applet is started, or an error message is displayed.

A `string` is defined in an HTM file using the following format:

```
<@stringID=Text string@>
```

A `string` is called from an HTM template file using the following Idoc Script format:

```
<$lc("ww<i>StringID</i>")$>
```

> **Note:** On Content Server web pages, you should use only the strings in the ww_strings.htm file.

Standard English strings are defined in the *IdcHomeDir*`/resources/core/lang` directory. Strings for other supported languages are provided by the Localization component.

HTML includes, strings, and static tables can be present in the same HTM file. A `string` resource does not require merge rules.

You must use HTML escape encoding to include the following special characters in a `string` value.

Escape Sequence	Character
`&at;`	@
`\&lf;`	line feed (ASCII 10)
`\&cr;`	carriage return (ASCII 13)
`\&tab;`	tab (ASCII 9)
`\&eatws;`	Eats white space until the next nonwhite space character.
`\<`	< (less than)
`\>`	> (greater than)
`\&sp;`	space (ASCII 32)
`\&#xxx;`	ASCII character that a decimal number represents (*nnn*)

You can specify strings for multiple languages in the same resource file using the language identifier prefix, if the languages all have single-byte characters or all have multibyte characters. Example 17–8 shows prefixes for strings in several languages in a resource file.

Example 17–8 Multiple Languages in the Same Resource File

```
<@myString=Thank you@>
<@es.myString=Gracias@>
<@fr.myString=Merci@>
<@de.myString=Danke@>
```

> **Caution:** Do not specify single-byte strings and multibyte strings in the same resource file. You should create separate resource files for single-byte and multibyte strings.

If you are specifying multibyte strings in your custom string resource, ensure that the character set specification on your HTML pages changes to the appropriate encoding. Resource files should have a correct `http-equiv charset` tag so that Content Server reads them correctly.

17.2.3.1 String Parameters

Text strings can contain variable parameters, which are specified by placing the parameter argument inside curly braces (for example, {1}). When a string is localized, the arguments are passed along with the string ID and the `ExecutionContext` value that contains the locale information. The following table describes the syntax for parameterized strings.

Syntax	Meaning	Examples
{{}	Opening curly brace. (Note that only the opening curly brace must be expressed as a literal.)	{{}Text in braces}
{n}	Substitute the *n*th argument.	Content ID {1} not found
{ni}	Substitute the *n*th argument, formatted as an integer.	dID {1i} does not exist
{nx}	Substitute the *n*th argument, formatted as an integer in hexadecimal.	
{nd}	Substitute the *n*th argument, formatted as a date.	The release date is {1d}
{nD}	Substitute the *n*th argument, formatted as a date. The argument should be ODBC-formatted.	The release date is {1D}
{nt}	Substitute the *n*th argument, formatted as a date and time.	The release date is {1t}
{ne}	Substitute the *n*th argument, formatted as elapsed time.	
{nT}	Substitute the *n*th argument, formatted as a date and time. The argument should be ODBC-formatted.	The release date is {1T}
{nfm}	Substitute the *n*th argument, formatted as a float with m decimal places.	The distance is {1f3} miles.
{nk}	Substitute a localized string using the *n*th argument as the string ID.	Unable to find {1k} revision of {2}
{nm}	Localize the *n*th argument as if it were a string-stack message. (For example, the argument could include concatenated text strings and localized string IDs.)	Indexing internal error: {1m}
{nl}	Substitute the *n*th argument as a list. The argument must be a list with commas (,) and carets (^) as the separators.	Add-ons: {1l}
{nK}	Takes a list of localization key names, separated by commas, and localizes each key into a list.	Unsupported byte feature(s): {1K}
{nM}	Takes a list of message strings and localizes each message into a list.	{1q} component, version {2q}, provides older versions of features than are currently enabled. {3M}

Syntax	Meaning	Examples
{nq}	If the *n*th argument is non-null and nonzero in length, substitute the argument in quotation marks. Otherwise, substitute the string "*syUndefined*".	Content item {1q} was not successfully checked in
{no}	Performs ordinal substitution on the *n*th argument. For example, 1st, 2nd, 3rd, and so on. The argument must be an integer.	"I am {1o}." with the argument 7 would localize into "I am 7th."
{n?text}	If the value of the *n*th argument is not 1, substitute the text.	{1} file{1?s} deleted
{n?text1:text2}	▪ If the value of the *n*th argument is not 1, substitute *text1*. ▪ If the value of the *n*th argument is 1, substitute *text2*. The (n?) function can be extended with as many substitution variables as required. The last variable in the list always corresponds to a value of 1.	There {1?are:is} currently {1} active search{1?es}.
{n?text1:text2:text3}	▪ If the value of the *n*th argument is not 1 or 2, substitute *text1*. ▪ If the value of the *n*th argument is 2, substitute *text2*. ▪ If the value of the *n*th argument is 1, substitute *text3*. The (n?) function can be extended with as many substitution variables as required. The last variable in the list always corresponds to a value of 1.	Contact {1?their:her:his} supervisor.

17.2.3.2 Editing a String Resource

Use the following procedure to edit an existing string resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource from the **Custom Resource Definition** list.
3. If the resource file contains multiple types of resources, click the **Strings** tab on the right.
4. Modify the strings in the **Custom Strings** list.
   - To edit an existing string, select the string, click **Edit**, modify the string text, and then click **OK**.
   - To add a string to the resource file, click **Add**.
   - To remove a string, select the string, click **Delete**, and then click **Yes** to confirm.

17.2.4 Dynamic Tables

Dynamic table resources are defined in the HDA file format. For more information and an example of an HDA ResultSet table, see Section 11.1.3.1.1, "Elements in HDA Files."

17.2.4.1 Merge Rules for Dynamic Tables

Merge rules are required for a dynamic table resource if data from the custom resource replaces data in an existing table. Merge rules are not required if data from the custom resource is to be placed in a new table.

17.2.4.2 Editing a Dynamic Table Resource

Use the following procedure to edit an existing dynamic table resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource file from the **Custom Resource Definition** list.
3. Click **Launch Editor**.
4. Modify the table in the text editor.
5. Save and close the resource file.

   Changes are reflected on the right of the **Resource Definition** tab.

17.2.5 Static Tables

Static tables, HTML includes, and strings can be present in the same HTM file.

17.2.5.1 Merge Rules for Static Tables

Merge rules are required for a static table resource if data from the custom resource replaces data in an existing table. Merge rules are not required if data from the custom resource is to be placed in a new table.

17.2.5.2 Editing a Static Table Resource

Use this procedure to edit an existing static table resource with the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource file from the **Custom Resource Definition** list.
3. Click **Launch Editor**.
4. Modify the table in the text editor.
5. Save and close the resource file. Changes are reflected in the Resource Tables list.

17.2.6 Queries

A query resource defines SQL queries, which are used to manage information in the Content Server database. Queries are used with service scripts to perform tasks such as adding to, deleting, and retrieving data from the database.

The standard Content Server queries are defined in the `QueryTable` table in the *`IdcHomeDir`*`/resources/core/tables/query.htm` file. You can also find special-purpose queries in the `indexer.htm` and `workflow.htm` files that are stored in the *`IdcHomeDir`*`/resources/core/tables` directory. Merge rules are not required for a query resource.

A query resource is defined in an HTM file using a ResultSet table with three columns: `name`, `queryStr`, and `parameters`.

- The `name` column defines the name for each query. To override an existing query, use the same name for your custom query. To add a new query, use a unique query name. When naming a new query, identify the type of query by starting the name with one of the following characters.

First Character	Query Type
`D`	Delete
`I`	Insert
`Q`	Select
`U`	Update

- The `queryStr` column defines the query expression. Query expressions are in standard SQL syntax. If there are any parameter values to pass to the database, their place is held with a question mark (?) as an escape character.
- The `parameters` column defines the parameters that are passed to the query from a service. A request from a web browser calls a service, which in turn calls the query. It is the responsibility of the web browser to provide the values for the query parameters, which are standard HTTP parameters The browser can pass query parameters from the URL or from FORM elements in the web page. For example, the `QdocInfo` query requires the `dID` (revision ID) to be passed as a parameter, so the value is obtained from the service request URL.

17.2.6.1 Query Example

The standard *QdocInfo* query, which Figure 17–1 shows, is defined in the *`IntradocDir`*`/core/config/resources/query.htm` file. This query obtains the metadata information to display on the DOC_INFO template page, which is the page displayed when a user clicks the **Information** icon on a search results page.

Figure 17–1 Standard QDocInfo Query

<@table QueryTable@>

Query Definition Table

name	queryStr	parameters
QdocInfo	SELECT Revisions.*, Documents.*, DocMeta.* FROM Revisions, Documents, DocMeta WHERE Revisions.dID=? AND Revisions.dID=Documents.dID AND DocMeta.dID = Documents.dID AND Revisions.dStatus<>'DELETED' AND Documents.dIsPrimary<>0	dID int

<@end@>

The parameter passed from the web browser URL is the dID, which is the unique identification number for the content item revision. The query expression selects the data that matches the dID for the primary revision from the **Revisions**, **Documents**, and **DocMeta** database tables, if the revision does not have the `DELETED` status.

Example 17–9 shows the contents of a `query.htm` file.

Example 17–9 query.htm File

```
<HTML>
<HEAD>
<META HTTP-EQUIV='Content-Type' content='text/html; charset=iso-8859-1'>
<TITLE>Query Definition Resources</TITLE>
</HEAD>
<BODY>
<@table QueryTable@>
<table border=1><caption><strong>Query Definition Table</strong></caption>
<tr>
    <td>name</td>
    <td>queryStr</td>
    <td>parameters</td>
</tr>
<tr>
    <td>QdocInfo</td>
    <td>SELECT Revisions.*, Documents.*, DocMeta.*
    FROM Revisions, Documents, DocMeta
    WHERE Revisions.dID=? AND Revisions.dID=Documents.dID AND DocMeta.dID =
Documents.dID AND Revisions.dStatus<>'DELETED' AND Documents.dIsPrimary<>0</td>
    <td>dID int</td>
</tr>
</table>
<@end@>
</BODY>
</HTML>
```

17.2.6.2 Editing a Query Resource

Use the following procedure to edit a query resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource from the **Custom Resource Definition** list.
3. If there are multiple tables in the resource, choose the query table to edit from the **Table Name** list.
4. Modify the selected query table.
   - To add a query to the table, click **Add**.
   - To edit an existing query, select the query, click **Edit**, modify the query expression or parameters or both, and then click **OK**.
   - To remove a query, select the query, click **Delete**, and then click **Yes** to confirm.

17.2.7 Services

A service resource defines a function or procedure that is performed by Content Server. A service call can be performed from either the client or server side, so services can be performed on behalf of the web browser client or within the system itself. For example:

- **Client-side request:** When you click a **Search** link on a Content Server web page, the standard search page is delivered to your web browser by the GET_DOC_ PAGE service, using the following URL segment:

  ```
  IdcService=GET_DOC_PAGE&Action=GetTemplatePage&Page=STANDARD_QUERY_PAGE
  ```

- **Server-side request:** You can use the START_SEARCH_INDEX service to update or rebuild the search index automatically in a background thread.

Services are the only way a client can communicate with the server or access the database. Any program or HTML page can use services to request information from Content Server or perform a specified function.

The standard Content Server services are defined in the `StandardServices` table in the *`IdcHomeDir`*`/resources/core/tables/std_services.htm` file. You can also find special-purpose services in the `workflow.htm` file in the *`IdcHomeDir`*`/resources/core/tables/` directory. For more information about standard and special-purpose services that Content Server provides, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

Services depend on other resource definitions to perform their functions. Any service that returns HTML requires a template to be specified. A common exception is the `PING_SERVER` service, which does not return a page to the browser.

Most services use a query. A common exception is the `SEARCH` service, which sends a request directly to the search collection. Merge rules are not required for a service resource.

Figure 17–2 shows an example of a service definition.

Figure 17–2 Service Definition Example

<@table StandardServices@>

Scripts For Standard Services

Name	Attributes	Actions
UPDATE_DOCINFO	DocService 2 null null documents !csUnableToUpdateInfo (dDocName)	3:doSubService:UPDATE_DOCINFO_SUB:12:null

<@end@>

A service resource is defined in an HTM file using a ResultSet table with the following three columns:

- The `Name` column defines the name for each service. For client-side service requests, this is the name called in the URL. To override an existing service, use the same name for your custom service. To add a new service, use a unique service name.
- The `Attributes` column defines the following attributes for each service.

Attribute	Description	Example (attributes from the DELETE_DOC service)
Service class	Determines, in part, what actions can be performed by the service.	**`DocService`** `4 MSG_PAGE null documents !csUnableToDeleteItem(dDocName)`
Access level	Assigns a user access level to the service. This number is the sum of the following possible bit flags: READ_PRIVILEGE = 1 WRITE_PRIVILEGE = 2 DELETE_PRIVILEGE = 4 ADMIN_PRIVILEGE = 8 GLOBAL_PRIVILEGE = 16 SCRIPTABLE_SERVICE=32	`DocService` **`4`** `MSG_PAGE null documents !csUnableToDeleteItem(dDocName)`

Attribute	Description	Example (attributes from the DELETE_DOC service)
Template page	Specifies the template that presents the results of the service. If the results of the service do not require presentation, this attribute is `null`.	`DocService 4 **MSG_PAGE** null documents !csUnableToDeleteItem(dDocName)`
Service type	If the service is to be executed inside another service, this attribute is `SubService`; otherwise, this attribute is `null`.	`DocService 4 MSG_PAGE **null** documents !csUnableToDeleteItem(dDocName)`
Subjects notified	Specifies the subjects (subsystems) to be notified by the service. If no subjects are notified, this attribute is *null*.	`DocService 4 MSG_PAGE null **documents** !csUnableToDeleteItem(dDocName)`
Error message	Defines the error message returned by the service if no action error message overrides it. This can be either an actual text string or a reference to a locale-sensitive string. For more information, see Section 1.5.5, "Localized String Resolution."	`DocService 4 MSG_PAGE null documents **!csUnableToDeleteItem(dDocName)**`

- The `Actions` column defines the actions for each service. An action is an operation to be performed as part of a service script. The action can execute an SQL statement, perform a query, run code, cache the results of a query, or load an option list. Each service includes one or more actions, which specify what happens upon execution.

  The `<br>` tags in the `Actions` column are for browser display purposes only, so they are optional. However, the `</td>` tag must occur immediately after the actions, without a line break in between. An action is defined using the following format:

  type`:`*name*`:`*parameters*`:`*control mask*`:`*error message*

Section	Description	Example (first action from the DELETE_DOC service)

type	Defines the type of action:
	`QUERY_TYPE = 1`
	`EXECUTE_TYPE = 2`
	`CODE_TYPE = 3`
	`OPTION_TYPE = 4`
	`CACHE_RESULT_TYPE = 5`
		`**5**:QdocInfo:DOC_ INFO:6:!csUnableToDeleteItem(dDocName) !csRevisionNoLongerExists`

name	Specifies the name of the action.	`5:**QdocInfo**:DOC_ INFO:6:!csUnableToDeleteItem(dDocName) !csRevisionNoLongerExist`
parameters	Specifies parameters required by the action. If no parameters are required, leave this part empty (two colons in a row).	`5:QdocInfo:**DOC_ INFO**:6:!csUnableToDeleteItem(dDocName) !csRevisionNoLongerExist`

Section	Description	Example (first action from the DELETE_DOC service)
control mask	Controls the results of queries to the database. This number is the sum of the following possible bit flags: No control mask = 0 CONTROL_IGNORE_ERROR = 1 CONTROL_MUST_EXIST = 2 CONTROL_BEGIN_TRAN = 4 CONTROL_COMMIT_TRAN = 8 CONTROL_MUST_NOT_EXIST = 16	5:QdocInfo:DOC_INFO:**6**:!csUnableToDeleteItem(dDocName) !csRevisionNoLongerExist
Error message	Defines the error message to be displayed by this action. This error message overrides the error message provided as an attribute of the service. This can be either an actual text string or a reference to a locale-sensitive string. For more information, see Section 1.5.5, "Localized String Resolution."	5:QdocInfo:DOC_INFO:6:**!csUnableToDeleteItem(dDocName)** **!csRevisionNoLongerExist**

17.2.7.1 Service Example

The DOC_INFO service provides a good example of how services, queries, and templates work together. Figure 17–3 shows the actions that the DOC_INFO service can take.

Figure 17–3 DOC_INFO Service

<@table StandardServices@>

Scripts For Standard Services

Name	Attributes	Actions
DOC_INFO	DocService 33 DOC_INFO null null ! csUnableToGetRevInfo	5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2 3:mapNamedResultSetValues:DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null 3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName) 3:getDocFormats:QdocFormats:0:null 3:getURLAbsolute::0:null 3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null 3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null 3:getWorkflowInfo:WF_INFO:0:null 3:getDocSubscriptionInfo:QisSubscribed:0:null 5:QrevHistory:REVISION_HISTORY:0:! csUnableToGetRevHistory(dDocName)

<@end@>

Example 17–10 shows the definition of the DOC_INFO service in the *IntradocDir*/config/resources/std_services.htm file.

Example 17–10 DOC_INFO Service Definition in std_services.htm File

```
    <HTML>
    <HEAD>
    <META HTTP-EQUIV='Content-Type' content='text/html; charset=iso-8859-1'>
    <TITLE>Standard Scripted Services</TITLE>
    </HEAD>
    <BODY>
    <@table StandardServices@>
    <table border=1><caption><strong>Scripts For Standard
Services</strong></caption>
    <tr>
    <td>Name</td><td>Attributes</td><td>Actions</td>
    </tr>
    <tr>
    <td>DOC_INFO</td>
    <td>DocSgervice
        33
```

```
        DOC_INFO
        null
        null<br>
        !csUnableToGetRevInfo</td>
    <td>5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2
        3:mapNamedResultSetValues:DOC_
INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null
        3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName)
        3:getDocFormats:QdocFormats:0:null
        3:getURLAbsolute::0:null
        3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null
        3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null
        3:getWorkflowInfo:WF_INFO:0:null
        3:getDocSubscriptionInfo:QisSubscribed:0:null
        5:QrevHistory:REVISION_HISTORY:0:!csUnableToGetRevHistory(dDocName)</td>
    </tr>
    </table>
    <@end@>
    </BODY>
    </HTML>
```

17.2.7.1.1 Attributes The following table describes the attributes of the preceding `DOC_INFO` service.

Attribute	Value	Description
`Service class`	`DocService`	This service is providing information about a content item.
`Access level`	`33`	`32` = This service can be executed with the `executeService` Idoc Script function. `1` = The user requesting the service must have `Read` privilege for the content item.
`Template page`	`DOC_INFO`	This service uses the `DOC_INFO` template (`doc_info.htm` file). The results from the actions are merged with this template and presented to the user.
`Service type`	`null`	This service is not a subservice.
`Subjects notified`	`null`	No subjects are affected by this service.
`Error message`	`!csUnableToGetRevInfo`	If this service fails on an English Content Server system, it returns this error message string: `Unable to retrieve information about the revision`

17.2.7.1.2 Actions The `DOC_INFO` service executes the following actions:

- `5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2`

Action Definition	Description
`5`	Cached query action that retrieves information from the database using a query.
`QDocInfo`	This action retrieves content item information using the `QDocInfo` query in the `query.htm` file.

Action Definition	Description
DOC_INFO	The result of the query is assigned to the parameter DOC_INFO and stored for later use.
2	The CONTROL_MUST_EXIST control mask specifies that either the query must return a record, or the action fails.
!csItemNoLongerExists2	If this action fails on an English Content Server system, it returns this error message string: This content item no longer exists

- 3:mapNamedResultSetValues:DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
mapNamedResultSetValues	This action retrieves the values of dStatus and dDocTitle from the first row of the DOC_INFO ResultSet and stores them in the local data. (This increases speed and ensures that the correct values are used.)
DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle	Parameters required for the mapNamedResultSetValues action.
0	No control mask is specified.
null	No error message is specified.

- 3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName)

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
checkSecurity	This action retrieves the data assigned to the DOC_INFO parameter and evaluates the assigned security level to verify that the user is authorized to perform this action.
DOC_INFO	Parameter that contains the security information to be evaluated by the checkSecurity action.
0	No control mask is specified.
!csUnableToGetRevInfo2(dDocName)	If this action fails on an English Content Server system, it returns this error message string: Unable to retrieve information for ''{dDocName}."

- 3:getDocFormats:QdocFormats:0:null

Action Definition	Description
`3`	Java method action specifying a module that is a part of the Java class implementing the service.
`getDocFormats`	This action retrieves the file formats for the content item using the `QdocFormats` query in the query.htm file. A comma-delimited list of the file formats is stored in the local data as `dDocFormats`.
`QdocFormats`	Specifies the query used to retrieve the file formats.
`0`	No control mask is specified.
`null`	No error message is specified.

- `3:getURLAbsolute::0:null`

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
getURLAbsolute	This action resolves the URL of the content item and stores it in the local data as *DocUrl*.
blank	This action takes no parameters.
0	No control mask is specified.
null	No error message is specified.

- `3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null`

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
getUserMailAddress	This action resolves the e-mail address of the content item author.
dDocAuthor,AuthorAddress	This action passes *dDocAuthor* and *AuthorAddress* as parameters.
0	No control mask is specified.
null	No error message is specified.

- `3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null`

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
getUserMailAddress	This action resolves the e-mail address of the user who has the content item checked out.
dCheckoutUser,CheckoutUserAddress	This action passes *dCheckoutUser* and *CheckoutUserAddress* as parameters.
0	No control mask is specified.
null	No error message is specified.

- `3:getWorkflowInfo:WF_INFO:0:null`

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
getWorkflowInfo	This action evaluates whether the content item is part of a workflow. If the WF_INFO ResultSet exists, then workflow information is merged into the DOC_INFO template.
WF_INFO	This action passes WF_INFO as a parameter.
0	No control mask is specified.
null	No error message is specified.

- `3:getDocSubscriptionInfo:QisSubscribed:0:null`

Action Definition	Description
3	Java method action specifying a module that is a part of the Java class implementing the service.
getDocSubscriptionInfo	This action evaluates if the current user has subscribed to the content item: ■ If the user is subscribed, an **Unsubscribe** button is displayed. ■ If the user is not subscribed, a **Subscribe** button is displayed.
QisSubscribed	Specifies the query used to retrieve the subscription information.
0	No control mask is specified.
null	No error message is specified.

- `5:QrevHistory:REVISION_HISTORY:0:!csUnableToGetRevHistory(dDocName)`

Action Definition	Description
5	Cached query action that retrieves information from the database using a query.
QrevHistory	This action retrieves revision history information using the *QrevHistory* query in the *query.htm* file.
REVISION_HISTORY	The result the query is assigned to the parameter REVISION_HISTORY. The DOC_INFO template uses this parameter in a loop to present information about each revision.
0	No control mask is specified.
!csUnableToGetRevHistory(dDocName)	If this action fails on an English Content Server system, it returns the error message string: `Unable to retrieve revision history for ''{dDocName}.''`

17.2.7.2 Editing a Service Resource

Use the following procedure to edit a service resource using the Component Wizard:

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource from the **Custom Resource Definition** list.

3. If there are multiple tables in the resource, choose the service table to edit from the **Table Name** list.
4. Modify the selected service table.
   - To add a service to the table, click **Add**.
   - To edit an existing service, select the service, click **Edit**, modify the service attributes or actions or both, and then click **OK**.
   - To remove a service, select the service, click **Delete**, and then click **Yes** to confirm.

17.2.8 Templates

A template resource defines the names, types, and locations of custom template files to be loaded for the component.

The actual template pages are separate `.htm` files that are referenced in the template resource file. **Template HTM** files contain the code that Content Server uses to assemble web pages. HTML markup in a template file defines the basic layout of the page, while Idoc Script in a template file generates additional HTML code for the web page at the time of the page request. Because HTM template files contain a large amount of script that is not resolved by Content Server until the final page is assembled, these files are not viewable web pages.

The template type of HTM file is used to define the following component files:

- **Template pages:** Standard template pages are located in the *IdcHomeDir*`/resources/core/templates` directory.
- **Report pages:** Standard report pages are located in the *IdcHomeDir*`/resources/core/reports` directory.

A template resource (templates.hda) is defined in the HDA file format. The standard templates are defined in the *IdcHomeDir*`/resources/core/templates/templates.hda` file. For more information and an example of an HDA ResultSet table, see Section 11.1.3.1.1, "Elements in HDA Files."

Merge rules are required for merging the new template definition into the **IntradocTemplates** table or **SearchResultTemplates** table. Typically, the merge is on the **name** column. Example 17–11 shows a MergeRules ResultSet for a template.

Example 17–11 MergeRules ResultSet

```
@ResultSet MergeRules
4
fromTable
toTable
column
loadOrder
MultiCheckinTemplates
IntradocTemplates
name
1
@end
```

The standard templates.hda file defines three ResultSet tables:

- The **IntradocTemplates** ResultSet table defines the template pages for all Content Server web pages except search results pages. This table consists of five columns:
  - The **name** column defines the name for each template page. This name is how the template is referenced in the Content Server CGI URLs and in code.
  - The **class** column defines the general category of the template. The most common *class* type is `Document`.
  - The **formtype** column defines the specific type of functionality the page is intended to achieve. The `formtype` is typically the same as the name of the form, except in lowercase characters.
  - The **filename** column defines the path and file name of the template file. The location can be an absolute path or can be relative to the template resource file when the template page is in the same directory as the template resource file.
  - The **description** column defines a description of the template.
- The **VerifyTemplates** ResultSet table is no longer used by Content Server, but this table remains in the templates.hda file as legacy code for reverse compatibility.
- The **SearchResultTemplates** table defines the template pages for search results pages. Template pages define how query results are displayed on the search results pages in the Library. Query result pages are a special type of search results page. This table consists of six columns:
  - The `name` column defines the name for each template page. This name is how the template is referenced in the Content Server CGI URLs, in code, and in the Web Layout Editor utility.

    Note: The `StandardResults` template (search_results.htm file) is typically used as the global template for standard search results pages and the query results pages in the Library. You can create a new template or change the "flexdata" of the `StandardResults` template through the Web Layout Editor, but these changes are saved in a separate file (*`IntradocDir`*`/data/results/custom_results.hda`) rather than in the `SearchResultTemplates` table in the templates.hda file.

  - The `formtype` column defines the specific type of functionality the page is intended to achieve. ResultsPage is the only form type currently supported for search results pages.
  - The `filename` column defines the path and file name of the template file. The location can be an absolute path or can be relative to the template resource file when the template page is in the same directory as the template resource file.
  - The `outfilename` column is for future use; the value is always null.
  - The `flexdata` column defines the metadata to be displayed for each row on the search results page. The format of text in the `flexdata` column follows:

    ```
    Text1 "text 1 contents"%<Tab>Text2 "text 2 contents"%
    ```

    In the format, the `Text1` value appears on the first line in each search results row, and the Text2 value appears on the second line. `<Tab>` represents a literal tab character.

Idoc Script can be used to define the contents in the `flexdata` field. You can also change the `flexdata` of the `StandardResults` template through the Web Layout Editor, but these changes are saved in a separate file (*`IntradocDir`*`/data/results/custom_results.hda`) rather than in the `SearchResultTemplates` table in the templates.hda file.

– The `description` column defines a description of the template.

Example 17–12 shows a custom template resource file that points to a custom Content Management page (multicheckin_doc_man.htm) and a custom search results page (MultiCheckin_search_results.htm).

Example 17–12 Custom Template Resource File

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
blFieldTypes=
@end
@ResultSet MultiCheckinTemplates
5
name
class
formtype
filename
description
DOC_MANAGEMENT_LINKS
DocManagement
DocManagementLinks
multicheckin_doc_man.htm
Page containing links to various document management functions
@end
@ResultSet MultiCheckin_2
6
name
formtype
filename
outfilename
flexdata
description
StandardResults
SearchResultsPage
MultiCheckin_search_results.htm
null
Text2 <$dDocTitle$> <$dInDate$>%Text1 <$dDocName$>%
apStandardResultsDesc
@end
```

17.2.8.1 Template and Report Pages

Template pages and report pages are also called **presentation pages**, because Content Server uses them to assemble, format, and present the results of a web page request.

The standard template pages are located in the *`IdcHomeDir`*`/resources/core/templates` directory. The standard report pages are located in the *`IdcHomeDir`*`/resource/core/reports` directory.

17.2.8.1.1 Template Page Example The template file for the standard Content Management page is `doc_man.htm`. Example 17–13 shows the contents of this file.

Example 17–13 The doc_man.htm File

```
<$include std_doctype_html_decl$>

<head>
        <$defaultPageTitle=lc("wwContentMgmt")$>
        <$include std_html_head_declarations$>
</head>

<$include body_def$>

<$include std_page_begin$>

<$include std_header$>

<table border="0" cellpadding="2" cellspacing="2" width="450" summary="">

<$include std_doc_man_pages$>

</table>

<table  cellpadding="7" cellspacing="7" summary="">
<$if showQuickHelp$>
<tr><td><form><INPUT type=Button onClick="QuickHelp('<$getHelpPage("QH_DocMan")$>', 'QH_DocMan')"
        value="<$lc("wwQuickHelp")$>"></form></td></tr>
<$endif$>
</table>

<$include std_page_end$>

</body>
</html>
```

In the example, the `std_doctype_html_decl` include references the standard Content Server document type. The `<head>` element references the page title, and the code for the `head` section is built using the `std_html_head_declarations` include code from the `std_page.htm` resource file. Other elements of the page definition follow:

1. Page elements common to most Content Server web pages are built using the `body_def`, `std_page_begin`, and `std_header` include code from the `std_page.htm` resource file.
2. The links on the Content Management page are built using include code from the `std_page.htm` resource file.
3. The `<table>` element in the example defines whether a **Quick Help** button should appear on the Content Management page.
4. The code at the end of the page is built using the `std_page_end` include code from the `std_page.htm` resource file.

Figure 17–4 shows a Content Management page.

Figure 17–4 Content Management Page

17.2.8.1.2 Report Page Example The template file for the standard Document Types report page is in the `doc_types.htm` file. Example 17–14 shows the contents of this file.

Example 17–14 The doc_types.htm File

```
<$include std_doctype_html_decl$>

<head>
        <$defaultPageTitle=lc("wwDocumentTypes")$>
        <$include std_html_head_declarations$>
</head>

<$include body_def$>

<$include std_page_begin$>

<$include std_header$>

<!--Directory Title--->
<table border="0" cellpadding="0" cellspacing="0" summary="">
<tr>
        <td width="75"><$if PageParent$><a href="<$PageParent$>">
                <$strTrimWs(inc("open_folder_image"))$></a>
                <$endif$></td>
        <td colspan="2" width="390"><span class=title><$PageTitle$></span></td>
</tr>
</table>

<$if IsSavedQuery$>
<!---Parameters for historical reports-->
<table border="0" cellpadding="0" cellspacing="0" summary="">
<tr>
        <td width="75" height="45"> </td><!---Indent-->
        <td><strong><span class=highlightField><$lc("wwReportCreated")$></span>
<$ReportCreationDate$></strong></td>
</tr>
</table>
<$endif$>

<!--Directory Header--->
<table  border="0" cellspacing="0" summary="">
<tr>
        <td width="75" height="45"> </td><!---Indent-->
        <td colspan="2" width="390"><$HeaderText$></td>
```

```
</tr>
</table>

<!---Doc types report-->
<table border=0 cellpadding=0 cellspacing=0 summary="">
<tr><td> </td></tr>
<tr>
        <td align=center width=<$StdPageWidth$>>
                <h1 class="underlinePageTitle"><$lc("wwDocumentTypes")$></h1>
        </td>
</tr>
<tr><td> </td></tr>

<$if IsMultiPage$>
<!---Navigation Bar-->
<tr>
        <td width=565 align="center"><$include std_page_nav_bar$></td>
</tr>
<$endif$>

<tr>
        <td>
        <table class="xuiTable" width=<$StdPageWidth$>
summary="<$stripXml(lc("wwReportResultsTable"))$>">
                <tr class="xuiAltHeader">
                        <td width=12% class="xuiAltHeader" scope="col"></td>
                        <td width=29% class="xuiAltHeader"
scope="col"><$lc("wwDocumentType")$></td>
                        <td width=49% class="xuiAltHeader" scope="col"><$lc("wwDescription")$></td>
                        <td width=12% class="xuiAltHeader"
scope="col"><$lc("wwImageFileName")$></td>
                </tr>
                <$rowCount=0$>

                <$loop DocTypes$>
                <$if rowCount%2 == 0$>
                        <$rowClass="xuiRow"$>
                <$else$>
                        <$rowClass="xuiAltRow"$>
                <$endif$>
                <tr class="<$rowClass$>">
                        <!--Document types are localized to each instance, so we must use direct
path to images directory.-->
                        <td><img src="<$HttpWebRoot$>images/docgifs/<$dGif$>"
alt="<$stripXml(lc("wwDoctypeIcon"))$>" border=0></td>
                        <td><$dDocType$></td>
                        <td><$dDescription$></td>
                        <td><$dGif$></td>
                </tr>
                <$rowCount=rowCount+1$>
                <$endloop$>
                </table>
        </td>
</tr>
</table>

<$include std_page_end$>

</body>
</html>
```

In the example, the `std_doctype_html_decl` include references the standard Content Server document type. The `<head>` element in the example references the page title and metadata, and the code for the head section is built using the `std_html_head_declarations` include code from the `std_page.htm` resource file. Other elements of the page definition follow:

1. Page elements common to most Content Server web pages are built using the `body_def`, `std_page_begin`, and `std_header` include code from the `std_page.htm` resource file.
2. The `Directory Title` section in the example displays the open folder image, links it to the parent page, and displays the page title.
3. The `Parameters for historical reports` section displays the original query date for a historical report.
4. The `Directory Header` section displays the report description.
5. The `Doc types report` section displays the table title.
6. The `Navigation Bar` section displays the page navigation bar if a historical report requires more than one page.
7. In the next `<table>` element, the first part displays the table column headers.
8. The last part of the `<table>` element loops on the document types to create the rows of the report table.
9. The code at the end of the page is built using the `std_page_end` include code from the `std_page.htm` resource file.

17.2.8.2 Editing a Template Resource

Use the following procedure to edit an existing template resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource from the **Custom Resource Definition** list.
3. To remove a template definition table or edit a template definition manually, click **Launch Editor** in the Custom Resource Definition area.
4. If there are multiple tables in the resource, choose the template table to edit from the **Table Name** list.
5. Modify the selected template table.
   - To add a template definition to the table, click **Add**.
   - To edit an existing template definition, select the definition, click **Edit**, modify the parameters, and then click **OK**.
   - To remove a template definition, select the definition, click **Delete**, and then click **Yes** to confirm.

17.2.9 Environment Resources

An environment resource defines configuration variables, either by creating new variable values or replacing existing values. Because custom resources are loaded after the standard config.cfg file is loaded, the variable values defined in the custom environment resource replace the original variable values.

An environment resource is defined in a CFG file using a name/value pair format:

```
variable_name=value
```

After defining a variable value, you can reference the variable in templates and other resource files with the following Idoc Script tag:

```
<$variable_name$>
```

Environment resource files can include comment lines, which are designated with a # symbol:

```
#Set this variable to true to enable the function.
```

17.2.9.1 Environment Resource Example

Example 17–15 shows the contents of an environment resource file.

Example 17–15 Environment Resource File

```
# Use this to turn on or off alternate row coloring
nsUseColoredRows=0

# These are the nested search field definitions.

nsFld1Caption=Document Text
nsFld1Name=
nsFld1Type=FullText
nsFld1OptionKey=
#
nsFld2Caption=Text
nsFld2Name=xtext
nsFld2Type=Text
nsFld2OptionKey=
#
nsFld3Caption=Date
nsFld3Name=xdate
nsFld3Type=Date
nsFld3OptionKey=
#
nsFld4Caption=Integer
nsFld4Name=xinteger
nsFld4Type=Int
nsFld4OptionKey=
#
nsFld5Caption=Option List
nsFld5Name=xoptionlist
nsFld5Type=OptionList
nsFld5OptionKey=optionlistList
#
nsFld6Caption=Info Topic
nsFld6Name=xwfsInfoTopic
nsFld6Type=OptionList
nsFld6OptionKey=wfsInfoTopicList
```

The `colored_search_resource.htm` template resource file in the Nested Search component references the `nsUseColoredRows` variable as follows:

```
<$if isTrue(#active.nsUseColoredRows)$>
    <$useColoredRows=1, bkgHighlight=1$>
<$endif$>
```

Standard configuration variables are defined in the *IntradocDir*`/config/config.cfg` file. For a complete list of configuration variables, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

17.2.9.2 Editing an Environment Resource

Use the following procedure to edit an existing environment resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Choose the resource file from the **Custom Resource Definition** list.
3. Click **Launch Editor**.
4. Modify the configuration variables in the text editor.
5. Save and close the resource file.

   Changes are reflected in the **Custom Environment Parameters** list.

> **Note:** The configuration settings might not appear in the **Custom Environment Parameters** list in the order they actually appear in the resource file. For easier viewing, launch the text editor.

17.3 Creating a Component Definition File

You can use the Component Wizard to create a component definition file or make changes to it.

Example 17–16 shows a component definition file that points to an environment resource file called `customhelp_environment.cfg`.

Example 17–16 Component Definition File for an Environment Resource

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
blFieldTypes=
@end
@ResultSet ResourceDefinition
4
type
filename
tables
loadOrder
environment
customhelp_environment.cfg
null
1
@end
```

17.4 Restarting Content Server to Apply a Component

Before you can apply a custom component to Content Server, you need to restart it. You can restart Content Server by restarting the WebCenter Content Managed Server with the Administration Console, shutdown and startup scripts, or Fusion Middleware Control.

The following example shows how to restart WebCenter Content with the `stopManagedWebLogic` and `startManagedWebLogic` scripts.

For more information, see "Managing System Processes" in the *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

To restart the WebCenter Content Managed Server with scripts on the command line:

1. Stop the WebCenter Content Managed Server with the `stopManagedWebLogic` script.
   - **UNIX script:** *DomainHome*`/bin/stopManagedWebLogic.sh UCM_server1`
   - **Windows script:** *DomainHome*`\bin\stopManagedWebLogic.cmd UCM_server1`
2. Stop the Administration Server with the `stopWebLogic` script.
   - **UNIX script:** *DomainHome*`/bin/stopWebLogic.sh`
   - **Windows script:** *DomainHome*`\bin\stopWebLogic.cmd`
3. Start the Administration Server with the `startWebLogic` script.
   - **UNIX script:** *DomainHome*`/bin/startWebLogic.sh`
   - **Windows script:** *DomainHome*`\bin\startWebLogic.cmd`
4. Start the WebCenter Content Managed Server with the `startManagedWebLogic` script.
   - **UNIX script:** *DomainHome*`/bin/startManagedWebLogic.sh UCM_server1`
   - **Windows script:** *DomainHome*`\bin\startManagedWebLogic.cmd UCM_server1`

18 Installing Components

This chapter describes how to install additional components in Oracle WebCenter Content Server.

This chapter includes the following sections:

18.1 About Installing Components

Server components for Content Server are installed by default, however, custom components and components downloaded from Oracle Technology Network must be installed and enabled before they can be used.

> **Note:** If you need only to enable or disable an installed component, see Chapter 12, "Enabling and Disabling Components for Content Server."

You can install components using one the methods that the following sections describe:

- Section 18.3, "Installing a Component with the Advanced Component Manager"
- Section 18.4, "Installing a Component with the Component Wizard"
- Section 18.5, "Installing a Component with the ComponentTool Utility"

Before installing a component, you must first download it to your instance. A component cannot be downloaded unless it meets the following requirements:

- The component must exist outside of the *IdcHomeDir*`/system` directory (that is, *DomainHome*`/ucm/idc/system`). This excludes all packaged components unless a patch has been uploaded to a component.
- The component must have a ZIP file with the appropriate name and be located inside the custom component or core component directory.

18.2 Packaging a Component for Installation

You can package a custom component in a ZIP file for installation on multiple Content Server instances with the Component Wizard.

To package a component for installation:

1. In the **Administration** tray or menu, choose **Admin Server**.

   The Admin Server displays the Component Manager page.

2. In the first paragraph on the Component Manager page, click **advanced component manager**.

   This displays the Advanced Component Manager page, which has a list of components available for downloading.

3. Choose the component to be packaged from the **Download Component** list
4. Click **Download** to display the File Download screen.
5. Select **Save this file** to disk, and click **OK**.
6. In the Save As dialog box, navigate to a directory, change the file name if necessary, and click **Save**.

   This creates a component ZIP file that can be used to install the component.

18.3 Installing a Component with the Advanced Component Manager

Follow these steps to install the component using the Advanced Component Manager:

1. In the **Administration** tray or menu, choose **Admin Server**.

   The Admin Server displays the Component Manager screen.

2. Click the **Browse** button, and find the ZIP file that was downloaded and saved.
3. Highlight the component name, and click **Open**.
4. Click **Install**. A message is displayed, detailing what will be installed.
5. Click **Continue** to continue with installation or **Cancel** to stop installation.
6. If you click **Continue**, a message appears after successful installation. You can select one of two options:
   - To enable the component and restart Content Server.
   - To return to the Component Manager, where you can continue adding components. When done, highlight the components you want to enable, and click **Enable**. When finished enabling components, restart the server.

18.4 Installing a Component with the Component Wizard

Follow these steps to install the component using the Component Wizard:

1. Start the Component Wizard:
   - (Windows operating system) From the **Start** menu, choose **Programs**, then **Oracle WebCenter Content Server**, then your server, then **Utilities**, and then **Component Wizard**.
   - (UNIX operating system) Run the `ComponentWizard` script in the *`DomainHome`*`/ucm/cs/bin` directory.

The Component Wizard main screen and the Component List screen are displayed.

2. On the Component List screen, click **Install**.

   The Install screen is displayed.

1. Click **Select**.
2. Navigate to the ZIP file that was downloaded and saved, and select it.
3. Click **Open**.

   The ZIP file contents are added to the Install screen list.

4. Click **OK**. You are prompted to enable the component.
5. Click **Yes**. The component is listed as enabled on the Component List screen.
6. Exit the Component Wizard.
7. Restart Oracle WebCenter Content Server.

Depending on the component being installed, a new menu option appears in the **Administration** tray or on the Admin Applet page. Some components simply extend existing functionality and do not appear as separate new options. For more information, see the component's documentation.

18.5 Installing a Component with the ComponentTool Utility

Run the ComponentTool utility and specify the ZIP file for the component to install and enable:

```
DomainHome/ucm/cs/bin/ComponentTool path_to_file/component.zip
```

19 Uninstalling Components

This chapter describes how to uninstall components from Oracle WebCenter Content Server.

This chapter includes the following sections:

- Section 19.1, "About Uninstalling Components"
- Section 19.2, "Uninstalling a Component from Content Server"

19.1 About Uninstalling Components

The Component List screen of the Content Server Component Wizard lists all the currently installed components. Starting the Component Wizard displays this screen, from which you can uninstall components.

19.2 Uninstalling a Component from Content Server

You can select a component and uninstall it from the Component List screen of the Component Wizard.

To uninstall a component from Content Server:

1. Start the Component Wizard, as described in Section 11.2.1, "Component Wizard."
2. On the Component List screen, select the component you want to uninstall.
3. Click the **Uninstall** button.

   This removes the selected component from the Content Server instance. The component files remain in the file system, but the component no longer appears on the list of components.

Part VI

Customizing Records

This part includes information about customizing Oracle WebCenter Content: Records.

Part VI contains the following chapters:

20

Customizing Disposition Actions

This chapter describes how to customize disposition actions for Oracle WebCenter Content: Records. Disposition actions are used in disposition instructions, which define the sequence of actions to be performed on content during its life cycle.

This chapter includes the following sections:

20.1 About Customizing Disposition Actions

Important: If custom dispositions were previously created using an older version of the Records system, those dispositions should be re-examined and updated to use the newest services and actions. The Action Service parameters have changed from previous versions of this software and any changes to existing custom dispositions are not mapped automatically.

A large number of built-in disposition actions are included, including Cutoff, Destroy, Transfer, Move, Declassify. Your environment may require disposition actions other than the predefined options. Disposition actions can be designed to reflect your organization's specific needs.

Custom disposition actions are based on Oracle WebCenter Content services, which can be called with specific parameters to define the behavior of the disposition actions. For example, you could create a disposition action to automatically retain the last three revisions of content items using the `DELETE_ALL_BUT_LAST_N_REVISIONS_SERVICE` service with the `NumberOfRevisions=3` parameter.

Important: Custom disposition features are available only to users with the Rma.Admin.Customization right. By default, this right is not assigned to any of the predefined roles. You must assign it to a role before this functionality is exposed.

20.2 Managing Custom Dispositions

The following tasks are for managing dispositions:

- Section 20.2.1, "Creating or Editing a Custom Disposition Action"
- Section 20.2.2, "Viewing Custom Disposition Action Information"
- Section 20.2.3, "Deleting a Custom Disposition Action"

20.2.1 Creating or Editing a Custom Disposition Action

To create a custom disposition action:

Important: Creating custom disposition actions requires in-depth technical knowledge of Oracle WebCenter Content. Contact Consulting Services to define custom disposition actions.

Permissions: The Rma.Admin.Customization right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means it must be assigned to a role for this functionality to be exposed.

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Disposition Actions** then **Custom**.
2. On the Configure Dispositions page, click **Add** in the Custom Disposition Action section.
3. On the Create or Edit Disposition Action page, enter a unique ID for the custom disposition action in the **Action ID** text box.
4. Enter a name for the custom disposition action in the **Action Name** text box.
5. Enter a description for the custom disposition action in the **Brief Description** text box.
6. Enter a group name for the custom disposition action in the **Group Name** text box. The default value for this field is stored in the `ww_strings.htm` file in the `wwOptGroupLabelCustomDispositionActionsList`. It is set to `Custom Actions` by default.

   To use a different group name than Custom Actions, modify the string value in the resource file and restart Content Server. Do not change the suggested default value in the Group Name field.
7. Choose the service to be used for the custom disposition action from the **Action Service** list.
8. (Optional) Specify one or more parameters for the selected action service.
9. (Optional) Select any of the checkboxes as required. Selections include **Must Be First**, **Must Be Last**, **Require Approval**. These actions determine when the custom disposition action will be used and how.

10. Click **Create**.

    A message is displayed saying the disposition action was created successfully, with the action information.

11. Click **OK**.

The following Action Service Parameters are required for the specific Action Service.

Disposition/Event	Service Parameters
Superseded	isScrub=1
Delete All Revisions (Destroy Metadata)	NumberOfRevisions=0, isDestroy=1, dRevRank=0
Delete Revision	NumberOfRevisions=0, isDestroy=1
Mark Transfer Completed	NumberOfRevisions=0, isDestroy=1, dRevRank=0
Mark Move Completed	NumberOfRevisions=0, isDestroy=1, dRevRank=0
Mark Accession Completed	NumberOfRevisions=0, isDestroy=1, dRevRank=0
Delete Previous Revision	NumberOfRevisions=1
Delete Old Revision	NumberOfRevisions=1
Mark Archive Completed	NumberOfRevisions=0, isDestroy=1, dRevRank=0
Archive Leave Metadata	isScrub=1
Mark Accession Completed (leave metadata)	isScrub=1
Mark Move Completed (leave metadata)	isScrub=1
Mark Transfer Complete (leave metadata)	isScrub=1
Mark Delete Revision Completed	NumberOfRevisions=0, isDestroy=1
Delete Complete	NumberOfRevisions=0, isDestroy=1
Mark Transfer Completed (prompt to keep or delete metadata)	NumberOfRevisions=0, isDestroy=1
Mark Move Complete (prompt to keep or delete metadata)	NumberOfRevisions=0, isDestroy=1
Mark Accession Complete (prompt to keep or delete metadata)	NumberOfRevisions=0, isDestroy=1
Mark Archive Complete (prompt to keep or delete metadata)	NumberOfRevisions=0, isDestroy=1
Mark Related Content	IsMarkAllRelations=1

To edit a custom disposition action:

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Disposition Actions** then **Custom**.
2. On the Configure Dispositions page, choose **Edit Action** from a disposition **Actions** menu.

3. Make modifications as required on the Create or Edit Disposition Action page, and click **Submit Update** when done.

   A message is displayed saying the disposition action was created successfully, with the action information.

4. Click **OK**.

20.2.2 Viewing Custom Disposition Action Information

To view the information about a custom disposition action:

> **Permissions:** The Rma.Admin.Customization right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means you must assign it to a role before this functionality is exposed.

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Disposition Actions** then **Custom**.
2. On the Configure Dispositions page, click the disposition name to view.
3. When done viewing, click **OK**.

20.2.3 Deleting a Custom Disposition Action

> **Permissions:** The Rma.Admin.Customization right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means you must assign it to a role before this functionality is exposed.

Custom disposition actions can be deleted only if they are no longer used in the disposition instructions for any category.

To delete a custom disposition action:

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Disposition Actions** then **Custom**.
2. On the Configure Dispositions page, choose **Delete Action** from a disposition's item **Actions** menu. You can also select the checkbox by the action name and choose **Delete** from the **Table** menu.

   A message is displayed saying the disposition action was deleted successfully.

3. Click **OK**.

To delete multiple dispositions, select the checkbox for the dispositions to delete on the Configure Dispositions page, and choose **Delete** from the **Table** menu.

20.3 Disabling Custom Disposition Actions

Permissions: The **Rma.Admin.Customization** right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means you must assign it to a role before this functionality is exposed.

Important: Some dispositions are required for processing of instructions to occur. Disabling a disposition could interfere with the processing of disposition instructions. Always verify ahead of time that it is acceptable to disable a disposition.

To disable a custom disposition action:

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Disposition Actions** then **Disable**.
2. On the Disposition Actions Configuration page, select the checkbox next to the actions that should be disabled.
3. Click **Submit Update** when done.

20.4 Creating a Custom Disposition Action

This example creates a custom disposition action that automatically retains the last three revisions of a content item.

Permissions: The Rma.Admin.Customization right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means you must assign it to a role before this functionality is exposed.

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Disposition Actions** then **Custom**.
2. On the Configure Dispositions page, in the Custom Disposition Action section, click **Add**.
3. Complete the metadata fields as follows on the Create or Edit Disposition Action page:
   a. In the Action ID field, type `RetainLast3Rev`.
   b. In the Action Name field, type `Retain Last 3 Revisions`.
   c. In the Brief Description field, type `Only keep the last 3 revisions of a content item`.
   d. In the Group Name field, type `Custom`.
   e. From the **Action Service** list, choose the *`wwString`* name of a disposition action; for example, `Notify Author`.
   f. In the Action Service Parameters field, type `NumberOfRevisions=3`.
4. Click **Create**.

The newly created disposition action can now be selected from the list of available disposition actions when creating disposition rules.

20.5 Creating Disposition Rules for Physical Content

Physical items can be assigned retention schedules, which define their life cycle. When creating a physical item you can assign a retention schedule to it. This links the physical item to a set of retention and disposition rules, which specify how long an item should be stored and when and how it should be disposed.

The same retention schedules and disposition rules may be used for physical items as for electronic items, but disposition rules used only for physical items can also be defined.

21

Customizing Bar Codes

This chapter describes how to customize bar codes in Oracle WebCenter Content: Records. You can add a custom bar code range and process nonstandard bar code data.

This chapter includes the following sections:

21.1 About Customizing Bar Codes

The PCM software is shipped with a default set of bar codes ranges and bar code transaction types (check in, check out, and set locations). You can add a set of bar code numbers to coincide with the system in place at your site and use them to provide additional custom functionality. After adding the numbers, a customized service must be created to use the new functionality. Consulting Services should be used to design this service.

The system can also be customized to process bar code files that are in a format other than the standard format used by PCM. This is done by altering a processing file to accommodate the format in use at your site. A detailed knowledge of Idoc Script is required to customize the processing file used to upload bar code data.

PCM bar codes are prefixed with a value when printing that should be stripped before processing. User bar codes are prefixed with `U`, storage bar codes with `S`, and object bar codes with `O`.

Permissions: The `Rma.Admin.Customization` right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means you must assign it to a role before this functionality is exposed.

21.2 Adding a Custom Bar Code Range

To add custom bar code ranges:

1. Choose **Physical** then **Configure** then **Function Barcodes** from the **Top** menu.
2. On the Configuring Custom Barcode page, click **Add**.
3. In the Create Custom Barcode dialog, enter the bar code and the activity or event associated with the bar code (for example, `Inventory` or `Storage Disposal`). A custom code must be a number between `7000` and `9999`. Click **OK** when done.
4. The Configuring Custom Barcode page opens showing the new bar code in the listing.

After defining the bar code range, contact Consulting Services to define a service called by the custom bar codes. The type of service used will vary depending on the type of functionality defined.

21.3 Processing Nonstandard Bar Code Data

Important: A detailed knowledge of Idoc Script is required to customize the file that processes bar code data. Contact Consulting Services for more assistance if needed.

Bar code processing in PCM uses code written with Idoc script in a *processing file* to evaluate each line in a *data file*. The processing file can be modified to customize how the system parses and processes the bar code data files.

The processing file is stored in the `barcode\resources\` directory and is named bar code_process_resource.htm.

The following is an example of a standard bar code data file:

```
H YYYYMMYYHHMMSS 00 0000000000       - Header
20050721125151 00 1000                - Transaction code
20050721125201 00 URMUSER             - Location
20050721125204 00 OB1                 - Object to be processed
20050721125204 00 OB2                 - Object to be processed
T 000                                 - Footer
```

21.3.1 Header and Footer Information

The header line in the standard data file begins with the value `H` and is ignored by the processing file. This can be customized if a header line is different or if one is absent. To modify this, change the `barcodeHeaderStartsWith` variable in the bar code environment file.

The standard footer line in the data file begins with `T 000`. When the processing file encounters this notation, processing stops and the processed data is uploaded. The `barcodeFooterStartsWith` variable can be changed to indicate a different footer type.

21.3.2 Data Information

Each line in the file that is not a Header or a Footer is parsed as data. Each valid transaction must have a Transaction Code, a Location, and Items to assign to the location.

21.3.2.1 Transaction Codes

Three default transaction codes are available:

- 1000: Check in
- 2000: Check out
- 3000: Set permanent and actual locations; this can also be used as a check-in transaction.

As noted in Adding a Custom Bar Code Range you can also create custom Transaction Codes. If custom codes are used, the location must be set to a user, storage item or object (for example, a box, folder, or tape).

The Transaction Date (dTransDate) and the Transaction Type (dTransType, which is the code designation of 1000, 2000, 3000, or custom number) must be set in the processing file. The following values should be cleared in the processing file:

- Location Type (`dLocationType`)
- Location (`dLocation`)
- Object Type (`dExtObjectType`)
- Bar Code (`dBarcode`)
- Bar Code Date (`dBarcodeDate`). Dates must be in the format `MM/dd/yyyy HH:MM:SS`.

The following variables should be set to `FALSE`:

- Bar Code Transaction Location (`barcodeTransLocation`)
- Bar Code Item (`barcodeItemSet`)

21.3.2.1.1 Location The `dLocationType` and `dLocation` values must be set to set the location. In addition, the `barcodeTransLocation` variable must be set to `TRUE`. This indicates that a location has been set for the current transaction.

If the location is a user, `dLocationType` must be set to `wwUser`. If the location is a storage location, `dLocationType` must be set to `wwStorage`. If the location is an object, `dLocationType` can be left blank and the processing code will determine the object type of the object during processing.

Multiple items can be assigned to the same location with one transaction. If the value for `barcodeTransLocation` is set to `TRUE`, it is assumed that the item being processed is an object being assigned to the current location set earlier. Make sure the `barcodeItemSet` value is set to `TRUE` after each item is parsed so it is processed.

21.3.2.1.2 Object To set the item, set `dBarcode` and `dBarcode Date` values. Also set `barcodeItemSet` to `TRUE`. This indicates that an item is ready to be processed.

22

Adding a Mobile Bar Code Reader

This chapter describes how to add a bar code reader for a mobile device.

This chapter includes the following sections:

22.1 About Adding a Mobile Bar Code Reader

You can install bar code scanner software on a mobile device that can be enabled for that functionality. Consult the device documentation for complete details about installing and enabling software on the device.

22.2 Installing Bar Code Scanner Software on a Mobile Device

The following files are needed for installation. Note that Windows Mobile version 5.0 is currently the only supported version:

- `BarcodeUtilityMobile.cab`: Installed with the PCM software and usually found at *`IntradocDir`*`\ucm\urm\config\MobileEdition`. This must be installed on the mobile device.
- The following files to be installed on the computer:
  - Microsoft ActiveSync, version 4.1 or later
  - The .NET Compact Framework 2.0 sp2 Redistributable file. This should already be installed.
  - The Symbol Managed Class Libraries. Download the Symbol Mobility Developer Kit v1.7 for .NET for from the following location:

    `http://support.symbol.com/support/search.do?cmd=displayKC&docType=kc&externalId=11683&sliceId=&dialogID=485552565&stateId=0%200%207340740`

    Install the file. After installation, the file is stored on the local disk in `C:\Program Files\Symbol Mobility Developer Kit for .NET\v1.7\SDK\Smart Devices\wce500\armv4i`.

22.3 Verifying Installation of the Mobile Bar Code Reader

Depending on the type of device and the software downloaded, usage instructions may vary. Some general instructions for use follow. For details about using default scanner wands and for background information about bar codes, see *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

1. Start the application on the mobile device.
2. A login page opens. After logging in, a Direct Scan window opens on the device.
3. Scan items and click the **Process** icon.
4. A Results page opens, showing the effects of scanning.

The following menu options are available from the main menu on the Direct Scan window:

- Real-time processing: Used to process and immediately upload data.
- Save transaction code: Used to temporarily save a transaction.
- Options: Used to set different defaults (for example, upload time out or the application locale).

On the Results page, two different views are available: **List**, to display a text list of all transactions and **Detail**, a table list of all transactions.

23

Creating Custom Reports

This chapter describes how to create custom reports for Oracle WebCenter Content: Records. to tailor data presentation for your site.

This chapter includes the following sections:

23.1 About Creating Custom Reports

Permissions: The `Rma.Admin.Customization` right is required to perform this task. This right is not assigned by default to any of the predefined roles, which means you must assign it to a role before this functionality is exposed. To create custom reports with a report source type of Query, a user must also have the `Rma.Admin.NoSecurity` right.

To create a custom report for your site, data is gathered for the report, a report template is chosen, the data is populated, and the report is generated. The data is gathered in XML format then formatted for use using a template. This process allows you to keep the data separate from the presentation of the data.

Several default reports and templates are provided when you install the Records system. New reports can be created by using the current reports as a base then editing them, or entirely new reports can be designed.

When creating custom reports, content on Content Server Adapter systems is not included in the report even if the content is managed by the Records system. To generate reports concerning Adapter content, run the reports on the Content Server Adapter system.

This section describes how to create customized reports using Oracle WebCenter Content services and queries. To create reports about users and content using the default reports provided with the software, see *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

To create new templates, the Oracle Business Intelligence Publisher (BI Publisher) functionality must be purchased and installed. This documentation describes how to use the default templates provided with the system. For details about creating new templates or editing the default templates, see the BI Publisher documentation.

Note: Before using custom reports, verify that the report library has been configured correctly. For more information, see "Configuring the Report Library for Records Management in Content Server" in the *Oracle WebCenter Content Installation Guide*.

After creating a new template, it can be added to the list of available templates for others to use. Templates as well as the report (that is, a report with data included) can be checked in as content. These are separate chickens, thus keeping data separate from the report format.

A similar interface is used to create the different elements of a report (report type, template, report sources).

Choose **Browse Content** then **Custom Physical Reports** or **Custom Record Reports** to access reports that have been created.

Note that user permissions are needed to access the data for a report as well as permissions to the report itself. Therefore, if two different people run a report, they might see different results depending on their rights.

To create a custom report using default templates and sources:

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Reports** to create a report for content items. Choose **Physical** then **Configure** then **Reports** to create reports for physical items.

   Click **Create New Report** and choose a report option:

   - **Create New Report**: Used to start the process of creating a new report. This option opens a check-in page, used to save the report name and save other detailed report information.
   - **Report Sources**: Opens the Configure Report Sources page, used to choose the criteria to gather data for the report.
   - **Templates**: Opens a search results page listing templates for reports.
   - **Download BI XML Data**: Opens a dialog where a user can choose to open the XML data used for the reports or save the data.

2. On the Configure Report Element page, choose the type of report to create and click **Configure**.

3. On the Report Check-in page, enter content information for the report as required at your site. Also enter the following report-specific information for the report:

   - Report Template: Choose a template from the list or click **Add New** to create a new template. For details about creating a new template, Section 23.2, "Creating Custom Templates."
   - Report Format: Choose a format from the list. If creating a bar code report, use PDF as your format type. Choices include:
     - System
     - User
     - HTML
     - PDF
     - RTF
     - XLS

- Report Source Type: Choose the type of source to use to gather the data for the report. Options include:
  - Service: Use Oracle WebCenter Content services to build the report data.
  - Query: Use Oracle WebCenter Content queries to gather data.
  - Dynamic Query: Use a dynamic query to gather data.

  If Services or Query are chosen, the correct service or query must be used to work with the template. A dynamic query will gather the appropriate data for use.
- Report Source: Choose a previously configured report source for the data or click **Add New** to create a new source using the Configure Report Sources page. For details, see Section 23.3, "Creating or Editing New Report Sources."

23.2 Creating Custom Templates

Templates are files that can be checked in and checked out of the system. They are treated as content items. For details about creating a new template without using an existing template as a guide, see the BI Publisher documentation.

To edit or create a new custom template:

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Reports** to create a template for content items. Choose **Physical** then **Configure** then **Reports** to create reports for physical items.

   Click **Templates**.
2. On the Configure Report Element page, choose the type of template to create. Click **Configure**.
3. The following options are available on the Report Templates page:
   - To create a new template based on an existing template, click **Get Native File** from the **Actions** menu of the template to be used. You can then save the file, edit the file as needed, and check it in as a new template.
   - To check out and edit an existing template, click **Check Out** from the **Actions** menu of the template to be used. Save the file, edit it, then check it back in to the system.
   - If a template exists and you would like to check it in using similar metadata as another template, click **Check In Similar** from the **Actions** menu of the template with metadata to be copied.

The Table menu on this page is used to perform actions for all template files or for selected files. The options shown depend on the configuration of the system and the permissions of the user accessing the page.

- **Select**: Allows the selection of all items or the deselection of items.
- **Actions**: Used to add items to a Content Basket or to a folio.
- **Edit**: Used to freeze or unfreeze selected items.
- **Set Dates**: Used to mark dates associated with items such as review dates, rescind dates, and other action dates.
- **Create Reports**: Used to create a report using a specified template.
- **Delete Metadata History**: Used to clear the metadata history changes.

- **Change view**: Used to change the way search results are displayed.
- **Search actions**: Used to save the search under a search name.

Additional options on this page are used to freeze or unfreeze template files, add files to a folio or Content Basket for later use, set dates for processing files, or create reports from a template.

23.3 Creating or Editing New Report Sources

Important: Creating custom report sources requires in-depth technical knowledge of services and queries. Contact Consulting Services for further assistance if needed.

To edit report sources:

1. Choose **Records** then **Configure** from the **Top** menu. Choose **Reports** to edit sources for a report for content items. Choose **Physical** then **Configure** then **Reports** to edit sources for reports for physical items.

   Click **Report Sources**.

2. On the Configure Report Element page, choose the report whose sources will be altered. Click **Configure**.
3. On the Configure Report Sources page, highlight a query or a service on the left side of the page. Click the right arrow to move the query or service to the right column for use.

   To remove a query or service from use, highlight the name and click the left arrow to move the item to the left column.

4. Click **Update** when done. The report sources for that particular type of report are altered and are used the next time the report is run.

Follow this procedure to create a new report source:

1. Use the previously described procedure to create a new report.
2. Click **Add New** in the Report Source section of the Report Check-in page.
3. A dialog opens. Follow the previously described procedure to add queries and services for the new source.
4. Click **OK** when done. The report source is added and is available for use with the new report.

23.4 Downloading a BI XML Data File

The provided XML data files can be used in conjunction with BI Publisher to customize report templates Using the XML data file, you can import data into a Word document and then edit the template to create a specialized report.

To select an XML data file:

1. Choose **Records** or **Physical** then **Configure** from the **Top** menu. Choose **Reports** then **Download BI XML Data**.
2. On the Configure Report Element page, choose the type of XML data file to download. Click **Download**.

3. You can open the data file to examine the contents in a browser window or you can save the file for later use.
4. Click **OK** when done.

Part VII

Integrating WebCenter Content into Your Environment

This part describes how to integrate Oracle WebCenter Content with enterprise applications.

> **Note:** Content Integration Suite (CIS) has been deprecated. Developers and system integrators are directed to use Remote Intradoc Client (RIDC), which provides a thin communication API for communication with Oracle WebCenter Content Server. For details, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*. For more information, see Section 29, "Using RIDC to Access Content Server."

Part VII contains the following chapters:

- Chapter 24, "Getting Started with Integrating WebCenter Content into Your Environment"
- Chapter 25, "Configuring WebCenter Content Web Services for Integration"
- Chapter 26, "Integrating JavaServer Pages with Content Server"
- Chapter 27, "Using the IdcCommand Utility to Access Content Server"
- Chapter 28, "Using the COM API for Integration"
- Chapter 29, "Using RIDC to Access Content Server"
- Chapter 30, "Using the Content Server JCR Adapter"
- Chapter 31, "Configuring Web Services with WSDL, SOAP, and the WSDL Generator"
- Chapter 32, "Customizing the DesktopTag Component"

24

Getting Started with Integrating WebCenter Content into Your Environment

This chapter describes how to integrate Oracle WebCenter Content with enterprise applications.

This chapter includes the following sections:

24.1 About Integration Methods

Several methods are available for integrating Oracle WebCenter Content with enterprise applications, such as application servers, catalog solutions, personalization applications, enterprise portals, and client-side software. In general, these integration methods serve to translate or pass methods and associated parameters with the goal of executing Oracle WebCenter Content Server services.

A Content Server service is a *window* for accessing the content and content management functions within Oracle WebCenter Content. For example, one simple integration option is to reference content that is managed within WebCenter Content by a persistent URL. Some other integration options enable you to use the Java API, the Microsoft Component Object Model (COM) interface, or the ActiveX control.

The focus of this chapter is to present the available integration options, suggest an approach, (like IdcCommand X, or persistent URL, or SOAP), and provide information about where to get the detailed documentation on that approach. Specifically, this chapter provides basic conceptual information about the integration of Oracle WebCenter Content within network system environments using various protocols, interfaces, and mapping services.

For information about using the IdcCommand utility to access Content Server services from other applications, see Chapter 27, "Using the IdcCommand Utility to Access Content Server."

For information about the COM interface, see Chapter 28, "Using the COM API for Integration."

For information about Remote Intradoc Client (RIDC) integration, see Chapter 29, "Using RIDC to Access Content Server."

24.2 Overview of Web Services

Web services reside as a layer on top of existing software systems, such as application servers, .NET servers, and Content Server. Adapted to the Internet as the model for communication, web services rely on the HyperText Transfer Protocol (HTTP) as the default network protocol. You can use web services as a bridge between dissimilar operating systems or programming languages to build applications with a combination of components.

> **Note:** The web services information in this document applies to Oracle WebLogic Server. For information about IBM WebSphere web services, see the *Oracle Fusion Middleware Third-Party Application Server Guide*.

WebCenter Content supports two ways of using web services to build applications that are integrated with Content Server:

- WebCenter Content web services together with Oracle WebLogic Server web services, with security configuration and Security Assertion Markup Language (SAML) support (introduced in WebCenter Content 11*g*)

  Content Server provides some web services built into the core product. Oracle WebLogic Server provides SOAP capabilities, and Oracle WebCenter Content Server supports several SOAP requests through Oracle WebLogic Server. For more information, see Chapter 25, "Configuring WebCenter Content Web Services for Integration."

- Web Services Definition Language (WSDL) and SOAP (Simple Object Access Protocol) files, with or without the WSDL generator component of Content Server (introduced in Oracle Universal Content Management 10*g*)

  The WSDL Generator component, `WsdlGenerator`, provides integration technologies for accessing the functionality of Content Server. This Content Server system component is installed and enabled by default. The WSDL Generator can create WSDLs for the services of Content Server, or the service calls can be written in SOAP. For more information, see Chapter 31, "Configuring Web Services with WSDL, SOAP, and the WSDL Generator."

With either way of using web services, you can use Oracle Web Services Manager (Oracle WSM) for security. For more information about Oracle WSM, see Chapter 25, "Configuring WebCenter Content Web Services for Integration," and the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

24.3 Virtual Folders and WebDAV Integration

Oracle WebCenter Content Server provides a hierarchical folder interface, the Folders feature, for organizing and managing content in the repository. Content Server also provides the legacy hierarchical folder interface, Contribution Folders. Oracle recommends Folders (the `FrameworkFolders` component) as the folder interface for WebCenter Content because it resolves performance issues that occur with Contribution Folders (the `Folders_g` component) and includes other enhancements.

To use a folder interface, you need to enable either the `FrameworkFolders` component or the `Folders_g` component. You cannot have both enabled on a Content Server instance. Having both the Folders and Contribution Folders features enabled is not a supported configuration because some other features, such as the

CoreWebdav system component, would not work correctly with both enabled. If you have both features enabled after an upgrade, you need to disable one of them.

The CoreWebdav component is installed with Oracle WebCenter Content Server and enabled by default. You can use the CoreWebdav component to author and manage your content remotely, using clients that support the WebDAV (Web-Based Distributed Authoring and Versioning) protocol.

- A folders interface is required for WebDAV functionality and the WebDAV Client product.
- The CoreWebdav component enables WebDAV functionality for remotely authoring and managing your content using clients that support the WebDAV protocol. For example, you can use Microsoft Windows Explorer to check out content from the repository, modify the content, and check it in rather than using a web browser interface.

24.3.1 Virtual Folders

The Folders component sets up an interface to Oracle WebCenter Content Server in the form of virtual folders (also called *hierarchical folders*). Virtual folders enable you to create a multilevel folder structure.

Virtual folders provide two main benefits:

- Users can find content by drilling down through a familiar folder-type interface.
- Users can apply default metadata to content items by checking them in through a particular folder.

The following structure is used for the Folders component:

- Each Oracle WebCenter Content Server instance has a common set of virtual folders. Any change to the folders is applied systemwide.
- There is one default system-level folder, called Contribution Folders. If you are using a custom folders interface, folders for these products may also appear at the system level of the Folders hierarchy.
- The system administrator can change the name of a system-level folder, but cannot delete it or add a custom system-level folder except through changes to the database. (Deleting a system-level folder disables it, but does not remove it from the system.)
- Each folder in the hierarchy contains content items that have the same numeric Folder value, which is assigned automatically upon creation of the folder. Changing the value of the Folder field for a content item places it in a different folder.
- The number of folders and number of files in each folder can be limited by the system administrator so that virtual folder functions do not affect system performance.

For detailed information about configuring Content Server for WebDAV integration, see *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

24.3.2 WebDAV Integration

WebDAV (Web-Based Distributed Authoring and Versioning) provides a way to remotely author and manage your content using clients that support the WebDAV protocol. For example, you can use Microsoft Windows Explorer to check in, check out, and modify content in the repository rather than using a web browser interface.

WebDAV is an extension to the HTTP/1.1 protocol that allows clients to perform remote web content authoring operations. The WebDAV protocol is specified by RFC 2518.0.

For more information, see the WebDAV Resources website at

http://www.webdav.org

WebDAV provides support for the following authoring and versioning functions:

- Version management
- Locking for overwrite protection
- Web page properties
- Collections of web resources
- Name space management (copy or move pages on a web server)
- Access control

When WebDAV is used with a content management system such as Content Server, the WebDAV client serves as an alternate user interface to the native files in the content repository. The same versioning and security controls apply, whether an author uses the Content Server web browser interface or a WebDAV client.

In Content Server, the WebDAV interface is based on the hierarchical Folders interface. For more information, see Section 24.3.1, "Virtual Folders."

24.3.2.1 WebDAV Clients

A WebDAV client is an application that can send requests and receive responses using a WebDAV protocol (for example, Microsoft Windows Explorer, Word, Excel, and PowerPoint). Check the current WebDAV client documentation for supported versions. The WebCenter Content WebDAV Client is a different product that enhances the WebDAV interface to Oracle WebCenter Content Server.

You can use WebDAV virtual folders in Windows Explorer to manage files that were created in a non-WebDAV client, but you cannot use the native application to check content in to and out of the Oracle WebCenter Content Server repository.

The Desktop software package also includes a WebDAV Client component and a Check Out and Open component.

24.3.2.2 WebDAV Servers

A WebDAV server is a server that can receive requests and send responses using WebDAV protocol and can provide authoring and versioning capabilities. Because WebDAV requests are sent over HTTP protocol, a WebDAV server typically is built as an add-on component to a standard web server. In Content Server, the WebDAV server is used only as an interpreter between clients and Content Server.

24.3.2.3 WebDAV Architecture

WebDAV is implemented in Oracle WebCenter Content Server by the WebDAV component. The architecture of a WebDAV request follows these steps:

1. The WebDAV client makes a request to Oracle WebCenter Content Server.
2. The message is processed by the web server (through a DLL in IIS).

3. On Oracle WebCenter Content Server, the WebDAV component performs these functions:
   - Recognizes the client request as WebDAV.
   - Maps the client request to the appropriate WebDAV service call on Oracle WebCenter Content Server.
   - Converts the client request from a WebDAV request to the appropriate Oracle WebCenter Content Server request.
   - Connects to the core Oracle WebCenter Content Server and executes the Oracle WebCenter Content Server request.
4. The WebDAV component converts the Oracle WebCenter Content Server response into a WebDAV response and returns it to the WebDAV client.

25

Configuring WebCenter Content Web Services for Integration

This chapter describes how to use Oracle WebCenter Content web services and Oracle WebLogic Server web services to integrate a client application with Oracle WebCenter Content Server.

This chapter includes the following sections:

- Section 25.1, "About Configuring WebCenter Content Web Services for Integration"
- Section 25.2, "Configuring Web Service Security Through Web Service Policies"
- Section 25.3, "Configuring SAML Support"

For general information about web services that you can use with Content Server, see Section 24.2, "Overview of Web Services."

The way to use web services described in this chapter was introduced in Oracle Universal Content Management 11*g*. If you want to use the way introduced in Oracle Universal Content Management 10*g*, with Web Services Definition Language (WSDL) and SOAP (Simple Object Access Protocol) files and the WSDL generator, see Section 31, "Configuring Web Services with WSDL, SOAP, and the WSDL Generator."

25.1 About Configuring WebCenter Content Web Services for Integration

WebCenter Content web services work with Oracle WebLogic Server web services to perform management functions for Content Server. Oracle WebLogic Server web services provide SOAP capabilities, and WebCenter Content web services include several built-in SOAP requests. WebCenter Content web services are automatically installed with Content Server, but they require additional configuration to set up security.

25.1.1 Technologies for Web Services

The core enabling technologies for WebCenter Content web services follow:

- SOAP (Simple Object Access Protocol) is a lightweight XML-based messaging protocol used to encode the information in request and response messages before sending them over a network. SOAP requests are sent from WebCenter Content web services to Oracle WebLogic Server web services for implementation. For more information about SOAP, see *Simple Object Access Protocol (SOAP)* at `http://www.w3.org/TR/soap12`.

- Web Services Security (WS-Security) is a standard set of SOAP extensions for securing web services for confidentiality, integrity, and authentication. For WebCenter Content web services, WS-Security is used for authentication, either for a client to connect to the server as a particular user or for one server to talk to another as a user. For more information, see the OASIS Web Service Security page at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.
- Web Service Policy (WS-Policy) is a standard for attaching policies to web services. For WebCenter Content web services, policies are used for applying WS-Security to web services. The two supported policies are `username-token` security and SAML security.

  Historically, Oracle used Oracle Web Services Manager (Oracle WSM) to secure its web services, and Oracle WebLogic Server used Web Services Security Policy (WS-SecurityPolicy) to secure its web services. Because web services security is partially standardized, some Oracle WSM and WS-SecurityPolicy policies can work with each other.

  Notes: Use Oracle WSM policies over Oracle WebLogic Server web services whenever possible. You cannot mix your use of Oracle WSM and Oracle WebLogic Server web services policies in the same web service.

  WebCenter Content web services (`idcws/` as context root) are SOAP based, while WebCenter Content native web services (`idcnativews/` as context root) are JAX_WS based. Both kinds of web services can be assigned Oracle WSM policies through the Oracle WebLogic Server Administration Console.

  The generic WebCenter Content web services are JAX-WS based and can be assigned Oracle WSM policies and managed by Oracle WSM. The native WebCenter Content web Services are SOAP based and can only support WS-Policy policies managed through the Oracle WebLogic Server Administration Console.

  For more information about Oracle WSM, see the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

  A subset of Oracle WebLogic Server web services policies interoperate with Oracle WSM policies. For more information, see "Interoperability with Oracle WebLogic Server 11g Web Service Security Environments" in the *Oracle Fusion Middleware Interoperability Guide for Oracle Web Services Manager*.

  Web Services Security Policy (WS-SecurityPolicy) is a set of security policy assertions for use with the WS-Policy framework. For more information, see the Web Services Security Policy (WS-SecurityPolicy) specification at `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html`.
- SAML is an XML standard for exchanging authentication and authorization between different security domains. For more information, see the Security Assertion Markup Language (SAML) specification at `http://docs.oasis-open.org/security/saml/v2.0/`.
- WebLogic Scripting Tool (WLST) is a command-line tool for managing Oracle WebLogic Server. For more information, see *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

25.1.2 WebCenter Content Web Services

WebCenter Content provides two types of web services: a general (generic) JAX-WS based web service, and a native SOAP based web service. The two types of web services reside in two different context roots. The *context root* is the primary identifier in the URL for accessing the web services.

The context roots follow:

- `idcws`

  Use this context root for general access to Content Server through any regular web services client.

- `idcnativews`

  The Remote Intradoc Client (RIDC) uses the native web services. Oracle recommends that you *do not* develop a custom client against these services. For more information about RIDC, see Chapter 29, "Using RIDC to Access Content Server."

The following table describes the WebCenter Content web service in the `idcws` context root.

WebCenter Content Web Service	Descriptions
`GenericSoapService`	This service uses a generic format similar to HDA for its SOAP format. It is almost identical to the generic SOAP calls that you can make to Content Server when you set `IsSoap=1`. For details of the format, see the published WSDL at `idcws/GenericSoapPort?WSDL`. You can apply WS-Security to `GenericSoapService` through WS-Policy. Content Server supports Oracle WSM policies for SAML and `username-token`. As a result of allowing WS-Security policies to be applied to this service, streaming Message Transmission Optimization Mechanism (MTOM) is not available for use with this service. Very large files (greater than the memory of the client or the server) cannot be uploaded or downloaded. `GenericSoapService` automatically has `oracle/wsmtom_policy` applied to it. Content Server cannot accept SOAP requests that have this policy applied. For `GenericSoapService` to work, the following policy must be applied to it: oracle/no_mtom_policy

The following table describes the WebCenter Content web services in the `idcnativews` context root.

WebCenter Content Web Services	Descriptions
IdcWebRequestService	This is the general WebCenter Content service. Essentially, it is a normal socket request to Content Server, wrapped in a SOAP request. Requests are sent to Content Server using streaming Message Transmission Optimization Mechanism (MTOM) in order to support large files. Streaming MTOM and WS-Security do not mix. As a result, do not apply WS-Security to this service because it will break the streaming file support. In order to achieve security, you must first log in using the IdcWebLoginService, then use the same JSESSIONID received from that service in the next call to IdcWebRequestService as a cookie.
IdcWebLoginService	This service is solely for adding security to IdcWebRequestService calls. There are no parameters for this service; it simply creates a session. The important field to retrieve is the JSESSIONID value for future calls to IdcWebRequestService. If you want to use WS-Security with IdcWebRequestService, then apply it here. Content Server supports Oracle WSM policies for SAML and username-token.

25.2 Configuring Web Service Security Through Web Service Policies

The WebCenter Content web services are installed and ready to use by default with the WebCenter Content EAR. However, unless you configure web service security (WS-Security) on any of the WebCenter Content web services, all connections to Content Server will use the anonymous user. To configure security for WebCenter Content web services, you configure WS-Security through WS-Policy. Additional configuration is required to enable authentication.

WS-Security is set through the use of web service policies (WS-Policy). Security policies can be set for web services to define their security protocol. In particular, the WebCenter Content web services support Oracle WSM policies.

Note: GenericSoapService automatically has oracle/wsmtom_policy applied to it. Content Server cannot accept SOAP requests that have this policy applied. For GenericSoapService to work, the following policy must be applied to it:

oracle/no_mtom_policy

WebCenter Content supports two general classes of policies, username-token and SAML, and the following Oracle WSM policies:

- oracle/wss11_saml_token_with_message_protection_service_policy
- oracle/wss11_username_token_with_message_protection_service_policy

You can use Oracle Enterprise Manager 11g Fusion Middleware Control to apply Oracle WSM policies to web services. For more information, see "Attaching Policies to Web Services" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

25.3 Configuring SAML Support

You can also provide SAML support for client-side certificate authentication. To provide SAML support so that the client can be the identity provider (that is, assert credentials), you need to configure a keystore, configure a Java Platform Security (JPS) provider to use the keystore, create a client credential store (CSF), and configure a Java client to use the keystore and CSF.

25.3.1 Configuring a Keystore

Both the server and client need a copy of a keystore. The server uses the keystore to authenticate the credentials passed by the client. A self-signed certificate can work for this situation, because the keystore is used only as a shared secret.

You can use the `keytool` utility to generate a self-signed certificate. Many of the values in the following example are the default values for the domain's `config/fmwconfig/jps-config.xml` file, described in Section 25.3.2, "Configuring JPS for WebCenter Content to Use the Keystore":

```
$ keytool -genkey -alias orakey -keyalg RSA -keystore default-keystore.jks
-keypass welcome -storepass welcome
```

You can enter any relevant data in the `keytool` command. The specifics do not matter except for the passwords for the keystore and the certificate, which the client uses.

25.3.2 Configuring JPS for WebCenter Content to Use the Keystore

Configuring the keystore in an Oracle WebLogic Server domain involves editing the *DomainHome*`/config/fmwconfig/jps-config.xml` file.

To configure JPS for WebCenter Content to use the keystore:

1. Verify that a provider is defined in the `<serviceProviders>` element, or define one.

   A provider should be defined in this element by default. If not, you need to add a `<serviceProvider>` element that defines a provider, as Example 25–1 shows.

Example 25–1 Service Provider Definition in jps-config.xml

```
<serviceProviders>
  <serviceProvider type="KEY_STORE" name="keystore.provider"
      class="oracle.security.jps.internal.keystore.KeyStoreProvider">
      <description>PKI Based Keystore Provider</description>
      <property name="provider.property.name" value="owsm"/>
  </serviceProvider>
</serviceProviders>
```

2. Verify that a keystore instance is defined in `<serviceInstances>`.

   A keystore instance should be defined by default.

   A keystore instance should be defined in this element by default. If not, you need to add a `<serviceInstance>` element that defines a keystore instance, as Example 25–2 shows.

Example 25–2 Keystore Instance Definition in jps-config.xml

```
<serviceInstances>
  <serviceInstance name="keystore" provider="keystore.provider"
      location="./default-keystore.jks">
      <description>Default JPS Keystore Service</description>
      <property name="keystore.type" value="JKS"/>
      <property name="keystore.csf.map" value="oracle.wsm.security"/>
      <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
      <property name="keystore.sig.csf.key" value="sign-csf-key"/>
      <property name="keystore.enc.csf.key" value="enc-csf-key"/>
  </serviceInstance>
</serviceInstances>
```

The location of the keystore instance must be set to the same location as where you created the keystore.

3. Verify that a reference to the keystore is in the `<jpsContexts>` element.

   This setting should be in the `jps-config.xml` file by default. If not, you need to add the setting, as Example 25–3 shows.

Example 25–3 Keystore in the JPS Context

```
<jpsContext name="default">
    <serviceInstanceRef ref="credstore"/>
    <serviceInstanceRef ref="keystore"/>
    <serviceInstanceRef ref="policystore.xml"/>
    <serviceInstanceRef ref="audit"/>
    <serviceInstanceRef ref="idstore.ldap"/>
</jpsContext>
```

4. Save the `jps-config.xml` file, and restart the WebCenter Content Managed Server and the Administration Server, as described in Section 17.4, "Restarting Content Server to Apply a Component."

25.3.3 Creating a Client CSF

On the client, there must be a credential store to store the keys to unlock the keystore. Oracle WebLogic Server provides a variety of ways to create a Credential Store Framework (CSF). One way you can create a CSF is with Oracle WebLogic Server Scripting Tool (WLST) commands.

To create a client CSF

1. Connect to the Oracle WebLogic Server domain, as Example 25–4 shows.

Example 25–4 Creating a Client CSF with WLST Commands

```
$ ./wlst.sh

$ connect()

$ createCred(map="oracle.wsm.security", key="keystore-csf-key", user="keystore",
password="welcome")
$ createCred(map="oracle.wsm.security", key="sign-csf-key", user="orakey",
password="welcome")
$ createCred(map="oracle.wsm.security", key="enc-csf-key", user="orakey",
password="welcome")
```

2. Use WLST `createCred` commands to define the CSF, as Example 25–4 shows.

   Change the values in the example to match the alias and passwords from the keystore you created.

   WLST creates a CSF wallet at *DomainHome*`/config/fmwconfig/cwallet.sso`. You can use the wallet only on the client.

3. Exit from WLST, and restart the Administration Server for the domain.
4. Send a copy of the wallet to the client.

25.3.4 Configuring a Java Client to Use the Keystore and CSF

Before you can configure a Java client to use the keystore and CSF, the client must have these items:

- A copy of the keystore
- A copy of the CSF wallet
- A client version of the `jps-config.xml` file

To configure a Java client to use the keystore and CSF:

1. Edit the `jps-config.xml` file for the Java client.
2. Add the locations of the keystore and the CSF wallet, as Example 25–5 shows, and save the file.

Example 25–5 Keystore and CSF Locations in the jps-config.xml file for a Java Client

```
<jpsConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="jps-config.xsd">
    <serviceProviders>
        <serviceProvider name="credstoressp"
                 class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider">
            <description>SecretStore-based CSF Provider</description>
        </serviceProvider>

        <serviceProvider type="KEY_STORE" name="keystore.provider"
                 class="oracle.security.jps.internal.keystore.KeyStoreProvider">
            <description>PKI Based Keystore Provider</description>
            <property name="provider.property.name" value="owsm"/>
        </serviceProvider>
    </serviceProviders>

    <serviceInstances>
        <serviceInstance name="credstore" provider="credstoressp" location="./">
            <description>File Based Credential Store Service Instance</description>
        </serviceInstance>
        <serviceInstance name="keystore" provider="keystore.provider"
                 location="./default-keystore.jks">
            <description>Default JPS Keystore Service</description>
            <property name="keystore.type" value="JKS"/>
            <property name="keystore.csf.map" value="oracle.wsm.security"/>
            <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
            <property name="keystore.sig.csf.key" value="sign-csf-key"/>
            <property name="keystore.enc.csf.key" value="enc-csf-key"/>
        </serviceInstance>
    </serviceInstances>
```

```
    <jpsContexts default="default">
        <jpsContext name="default">
            <serviceInstanceRef ref="credstore"/>
            <serviceInstanceRef ref="keystore"/>
        </jpsContext>
    </jpsContexts>
</jpsConfig>
```

3. Set `oracle.security.jps.config`, a Java system property, to point to the `jps-config.xml` file:

   ```
   System.setProperty("oracle.security.jps.config", "jps-config.xml");
   ```

   You can set this location in the client, during execution.

26

Integrating JavaServer Pages with Content Server

This chapter describes how to integrate Oracle WebCenter Content Server with JavaServer Pages (JSP).

This chapter includes the following sections:

26.1 About JSP Integration

You can access the Content Server core functionality from JavaServer Pages (JSP) to deliver forms and custom pages through the JSP page execution functionality using the built-in Apache Jakarta Tomcat Server.

> **Note:** The Content Server JSP functionality should not be enabled when you are using Site Studio for External Applications (SSXA) in conjunction with Content Server. Enabling the Content Server JSP functionality would cause parsing errors in SSXA templates.

26.1.1 JSP Execution

The JSP Execution functionality uses the built-in Apache Jakarta Tomcat Servlet/JSP Server to access the content and content management functions within Content Server.

The Apache Jakarta Tomcat Server is a free, open-source server of Java Servlet and JavaServer Pages that is run inside of Content Server when the feature is enabled. The integration of Tomcat Server with WebCenter Content provides the benefit of increased performance for content delivery.

Using JSP Execution functionality enables developers to access and modify Content Server content, ResultSets, personalization and security definitions, and predefined variables and configuration settings through JavaServer Pages rather than through standard component architecture. Services and Idoc Script functions can also be executed from JSP pages which reside as executable content in Content Server.

Important: JSP pages can execute Idoc Script functions only when the JSP page is being served on Oracle WebCenter Content Server as part of the JSP Execution functionality. JSP pages served on a separate JSP server do not have this functionality. In those cases, checking in a JSP page to Oracle WebCenter Content Server provides revision control but does not provide dynamic execution of Idoc Script functions on the presentation tier (JSP server).

26.1.2 Tomcat

The capability for JSP to call services is provided by integrating the Tomcat 5.025 server with the Content Server core functionality.

- Tomcat is a free, open-source server of Java Server and JavaServer Pages; version 5.025 complies with Servlet 2.4 and JSP 2.0 specifications.
- The main benefit of integrating Tomcat into Content Server is the increase in performance of delivering content. The direct integration eliminates the need for a socket-based interface and enables the use of all Oracle WebCenter Content Server core capabilities.
- Although Tomcat is embedded in Content Server, you can use server.xml as the configuration file to modify the internal Tomcat engine to suit your needs.

Note: This product includes software developed by the Apache Software Foundation (`http://www.apache.org/`).

26.1.3 Features

With JSP support enabled, custom components can include JSP pages of type `jsp` and `jspx`.

- The *`DomainHome`*`/ucm/cs/weblayout/jsp` directory is able to host JSP pages by default.
- The Oracle WebCenter Content distribution media also includes the current Java EE SDK.

26.2 Configuring JSP Support

Use the following procedure to enable and configure JSP support.

1. In Content Server, create a new security group to be used for JSP pages (called `jsp` in the subsequent steps). This security group should be restricted to developers. This step is not required but it is recommended for developer convenience. Any security groups to be enabled for JSP must be specified in Step 5.
   a. Display the User Admin screen.
   b. From the **Security** menu, choose **Permissions by Group**.
   c. Click **Add Group.**
   d. Enter `jsp` as the group name, enter a description, and then click **OK**.
   e. Assign **Admin** permission to the admin role and any developer roles.

f. Assign **Read** permission to all non-admin roles.

g. Click **Close**.

2. If you run on AIX, HP-UX, or Linux s390, the Java 2 SDK, which is required for the JSP integration, is not installed on your system automatically, nor is it provided on the distribution media. For the internal JSP engine to run on any of these operating systems, a 1.5 JDK must be present on the server, and the CLASSPATH value in the intradoc.cfg file must be modified to include the path to the tools.jar file. For example, for a default 1.5 install on AIX, this file should be in `/usr/java15/lib`.

3. Click one of the following options:

   - On the Admin Server page, click **General Configuration**.
   - From the System Properties utility, click the **Server** tab.

4. Enable the JSP prompt:

   - For the Admin Server: click **Enable Java Server Page (JSP)**
   - For System Properties: click **Execute Java Server Page (JSP)**

5. Enter the security groups to be enabled for JSP (including the security group you created in Step 1).

6. Save the settings, and restart Content Server.

26.3 Loading Example Pages

Use either of the following procedures to load example pages into Content Server:

- Check in the `.war` file in the JSP security group. Make sure to check in other content to the JSP security group before checking in the WAR file.
- Start the JSP Server Web App Admin from the Administration page.

27

Using the IdcCommand Utility to Access Content Server

This chapter describes how to use the IdcCommand utility to access Oracle WebCenter Content Server services from other applications

This chapter includes the following sections:

27.1 About the IdcCommand Utility

The IdcCommand utility is a standalone Java application that executes Content Server services. Almost any action you can perform from the Content Server browser interface or administration applets can be executed from IdcCommand.

The program reads a Specifying a Command File, which contains service commands and parameters, and then calls the specified services. A log file can record the time that the call was executed, whether the service was successfully executed, and if there were execution errors.

> **Note:** The IdcCommand utility returns only information about the success or failure of the command. To retrieve information from Oracle WebCenter Content Server in an interactive session, use the Java COM wrapper IdcCommandX, available on Microsoft Windows platforms.

To run the IdcCommand utility, you must specify the following parameters on the command line or in the `intradoc.cfg` configuration file:

- A command file containing the service commands and parameters
- A Content Server user name for a user who has permission to execute the services being called
- A path and file name for a log file
- The connection mode (`auto`, `server`, or `standalone`)

Certain commands that cannot be executed in `standalone` mode. In general, the server performs these commands asynchronously in a background thread. This happens in the update or rebuild of the search index.

For information about using services in custom components, see Chapter 11, "Getting Started with Content Server Components," and the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

27.2 Setting Up IdcCommand

To set up IdcCommand, you must specify the following two things:

- A Specifying a Command File, which specifies the services to be executed and any service parameters.
- Specifying Configuration Options, which specify the command file and other IdcCommand information. You can set IdcCommand configuration options in two places:
  - In a configuration file, using name/value pairs, as Example 27–1 shows.

Example 27–1 IdcCommand Options in a Configuration File

```
IdcCommandFile=newfile.hda
IdcCommandUserName=sysadmin
IdcCommandLog=C:/domain/newlog.txt
ConnectionMode=server
```

  - On the command line, when running IdcCommand, specifying option flags such as these:

Example 27–2 IdcCommand Options on the Command Line

```
-f newfile.hda -u admin -l C:/domain/newlog.txt -c server
```

> **Note:** Command-line configuration options override the settings in the configuration file.

IdcCommand is run from a command line. You can specify the Specifying Configuration Options either from the command line or in a configuration file. For more information, see Section 27.3, "Running IdcCommand."

27.2.1 Specifying a Command File

The command file defines the service commands and parameters that are executed by the IdcCommand utility. Command files must follow rules for syntax, precedence, and special tags and characters.

27.2.1.1 Command File Syntax

The command file uses the HDA (hyperdata file) syntax to define service commands.

- Each service to be executed, along with its parameters, is specified in a `@Properties LocalData` section.
- For some services, a `@ResultSet` section is used to specify additional information.

- Data from one section of the command file is not carried over to the next section. Each section must contain a complete set of data for the command.
- Service names and parameters are case sensitive.

Example 27–3 shows a command file that executes the ADD_USER service and defines attributes for two new users.

Example 27–3 Command File for the ADD_USER Service

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
# Add users
@Properties LocalData

IdcService=ADD_USER
dName=jsmith
dUserAuthType=Local
dFullName=Jennifer Smith
dPassword=password
dEmail=email@example.com
@end
@ResultSet UserAttribInfo
2
dUserName
AttributeInfo
jsmith
role,contributor,15
@end
<<EOD>>
@Properties LocalData
IdcService=ADD_USER
dName=pwallek
dUserAuthType=Local
dFullName=Peter Wallek
dPassword=password
dEmail=email@example.com
@end
@ResultSet UserAttribInfo
2
dUserName
AttributeInfo
pwallek
role,contributor,15,account,marketing,7
@end
<<EOD>>
```

27.2.1.2 Precedence

IdcCommand uses precedence to resolve conflicts among the name/value pairs within the `LocalData` section of the command file. When normal name/value pairs are parsed, they are assumed to be within the `@Properties LocalData` tag. If the section contains HDA tags, the normal name/value pairs take precedence over name/value pairs within the `@Properties LocalData` tag.

For example, if `foo=x` is in a normal name/value pair and `foo=y` is within the `@Properties LocalData` tag, the name/value pair `foo=x` takes precedence because it is outside the tag.

27.2.1.3 Special Tags and Characters

These special tags and characters can be used in a command file.

Special Character	Description
IdcService=*service_name*	Each section of the command file must specify the name of the service it is calling.
<<EOD>>	The end of data marker. The command file can include one or more sections separated with an end of data marker. For an example, see Section 27.2.1.1, "Command File Syntax."
#	The pound character placed at the beginning of a line indicates that the line is a comment.
\	The backslash is an escape character.
@Include *filename*	This tag enables you to include content from another file at the spot where the `@Include` tag is placed. This tag can be used to include a complete HDA file or to include shared name/value pairs. This inclusion takes the exact content of the specified file and places it in the location of the `@Include` tag. A file can be included as many times as desired and an included file may include other files. However, circular inclusions are not allowed.

27.2.2 Specifying Configuration Options

To run the IdcCommand utility, specify the following parameters on the command line or in the *DomainHome*`/ucm/cs/bin/intradoc.cfg` configuration file.

Parameter	Required?	Command Line Syntax	Configuration File Syntax
Command File	Yes	`-f name.txt`	`IdcCommandFile=name.txt`
User	Yes	`-u sysadmin`	`IdcCommandUserName=sysadmin`
Log File	No	`-l C:/logs/log.txt`	`IdcCommandLog=C:/logs/log.txt`
Connection Mode	No	`-c auto`	`ConnectionMode=auto`

Note: Command-line configuration options override the settings in the configuration file.

27.2.2.1 Command File

You must specify the name of the command file that contains the service commands and parameters. The command file parameter can specify a full path (such as `C:/command_files/command.txt`), or it can specify a relative path. For more information, see Section 27.2.1, "Specifying a Command File."

27.2.2.2 User

You must specify an Oracle WebCenter Content Server user name. This user must have permission to execute the services being called.

27.2.2.3 Log File

You can specify a path and file name for an IdcCommand log file. As each command is executed, a message is sent to the log file, which records the time the command was executed and its success or failure status. If the log file already exists, it is overwritten

with the new message. The log file can be used to display processing information to the user.

- If the action performed is successful, a "success" message is written to the log file.
- If the action performed is not successful, an error message is written to the log file.
- If no log file is specified, information is logged only to the screen.

27.2.2.4 Connection Mode

You can specify the connection mode for executing the IdcCommand services.

Connection Mode	Description
auto	IdcCommand attempts to connect to the Oracle WebCenter Content Server instance. If this fails, services are executed in standalone mode. This is the default connection mode.
server	IdcCommand executes services only through Content Server.
standalone	IdcCommand executes services in a standalone session. There are certain services that cannot be executed in standalone mode. In general, these services are performed asynchronously by the server in a background thread. For example, this happens during update or rebuild of the search index.

27.3 Running IdcCommand

To run IdcCommand:

1. Create a new IdcCommand working directory.

   Use this directory for your command file and configuration file.

2. Create a Specifying a Command File in the working directory to specify the desired service commands.

3. Copy the `intradoc.cfg` configuration file from the *`DomainHome`*`/ucm/cs/bin` directory into the working directory.

   Important: Do not delete the `IntradocDir` or `WebBrowserPath` information.

4. Add IdcCommand options to the `intradoc.cfg` file in the working directory, as Example 27–4 shows.

Example 27–4 IdcCommand Options in the intradoc.cfg File

```
IdcCommandFile=newfile.hda
IdcCommandUserName=sysadmin
IdcCommandLog=C:/domain/newlog.txt
```

   For more information, see Section 27.2.2, "Specifying Configuration Options."

5. Run the IdcCommand utility from the *`DomainHome`*`/ucm/cs/bin` directory:

   ```
   IdcCommand.exe
   ```

27.4 Using the Launcher

The Launcher is a native C++ application used to manage services in Windows environments and to construct command line arguments and environment settings for the Java VM.

The main operation of the Launcher is to find and read its configuration files, compute any special values, then launch an executable with a command line that it constructs. Configuration files support Bourne Shell-like substitutions, all of which start with the dollar sign ($) followed by an alphanumeric identifier or expression inside braces ({}).

The Launcher executable is installed in *DomainHome*/ucm/native/platform/bin/Launcher. On UNIX systems, symlinks are created in the bin directory to Launcher.sh, a Bourne Shell wrapper that executes the Launcher executable. The purpose of this wrapper is to locate the correct binary Launcher executable for the platform. The term Launcher is used here to refer to the native Launcher executable or to the Launcher.sh Bourne Shell script.

The Launcher or the symlink to Launcher.sh must reside in a directory with a valid intradoc.cfg configuration file and must have the same name as the Java class file to be launched (case sensitive). The Launcher uses this name to set the environment variable STARTUP_CLASS.

On Windows this name is computed by calling GetModuleFileName(). On UNIX systems, it is computed by inspecting argv[0]. The PLATFORM variable is set to the Content Server identifier for the platform. The variable BIN_DIR is set to the directory where the Launcher is located.

The Launcher reads a file named intradoc.cfg from BIN_DIR. This file should contain a value for IntradocDir. The IntradocDir directory is used as the base directory for resolving relative paths. Any unqualified path in this document should be taken as relative to the IntradocDir. Future releases of Content Server may change or remove these variable names.

If the intradoc.cfg file does not contain a value for IdcResourcesDir, the Launcher sets IdcResourcesDir to $IntradocDir/resources. If the Launcher is starting a Windows service, it sets IS_SERVICE to 1. If it is unset, the Launcher also sets PATH_SEPARATOR to the correct character for the platform.

The Launcher reads the intradoc.cfg file first to find the locations of configuration files, then reads all available configuration files in this order:

1. $IdcResourcesDir/core/config/launcher.cfg
2. $BIN_DIR/../config/config.cfg
3. $IntradocDir/config/config.cfg
4. $IntradocDir/config/config-$PLATFORM.cfg
5. $IntradocDir/config/state.cfg
6. $IdcResourcesDir/core/config/launcher-$PLATFORM.cfg
7. $BIN_DIR/intradoc.cfg
8. $BIN_DIR/intradoc-$PLATFORM.cfg
9. All files specified on the command line, using the -cfg option.

Tip: You can assign variable values directly on the command line by using the `-cfg` option *NAME=VALUE*.

27.4.1 Quotation Rules

The Launcher uses Bourne Shell-like quotation rules. A string can be inside double quotation marks (") to escape spaces. A backslash (\) can precede any character to provide that character. After a final command line is computed, the Launcher separates it into spaces without quotation marks. Each string is then used without quotation marks as an entry in the `argv` array for the command.

27.4.2 Computed Settings

After reading the configuration files, the Launcher processes variable substitutions. Some variables can have extra computations to validate directories or files, build command-line argument lists, or construct `PATH`-like variables.

These special computations are performed for variables based on their type. To set a type for a variable, set `TYPE_`*`variable_name`*`=`*`typename`* in any of the configuration files listed previously.

The following list describes Launcher variable types:

- `file`

  Example 27–5 shows some `file` type variables.

Example 27–5 file Launcher Variables

```
TYPE_PASSWD_FILE=file
PASSWD_FILE_sys5=/etc/passwd
PASSWD_FILE_bsd=/etc/master.passwd
```

  This type looks for a file. If the value of *`variable_name`* is a path to an existing file, it is kept. If not, every variable beginning with *`variable_name`*`_` is checked. The last value, which is a path to an existing file, is used for the new value of *`variable_name`*.

  In this example `PASSWD_FILE` is set to `/etc/master` if `/etc/master.passwd` exists, or it is set to `/etc/passwd</span>` if `/etc/passwd` exists. Otherwise, `PASSWD_FILE` is undefined.

- `directory`

  Example 27–6 shows some `directory` type variables.

Example 27–6 directory Launcher Variables

```
TYPE_JDK=directory
JDK_java_home=$JAVA_HOME
IdcNativeDir=$IdcHomeDir/native
DEFAULT_JDK_DIR=$OS_DIR/$PLATFORM
JDK_legacy142=$DEFAULT_JDK_DIR/j2sdk1.4.2_04
JDK_default=$DEFAULT_JDK_DIR/jdk1.5.0_07
```

  In this example `JDK` is set to the same value as the last of the `JDK_` variables that is a directory. Typically, this would point at the JDK installed with Oracle Fusion Middleware. Note that `JDK_java_home` references `$JAVA_HOME`; if a variable is

not defined in any configuration file but is in the environment, the environment value is used.

- `executable`

  Example 27–7 shows some `executable` type variables.

Example 27–7 executable Launcher Variables

```
TYPE_JAVA_EXE=executable
JAVA_EXE_default=java$EXE_SUFFIX
JAVA_EXE_jdk_default=$JDK/bin/java$EXE_SUFFIX
```

The executable type looks for an executable. It works very much like the file type, but looks through every directory in `$PATH` for each candidate value. In this example `JAVA_EXE` is set to the Java executable in the `JDK` if it exists. Otherwise it is set to the first Java executable in the `PATH`.

- `list`

  Example 27–8 shows some `list` type variables.

Example 27–8 list Launcher Variables

```
TYPE_JAVA_OPTIONS=list
JAVA_MAX_HEAP_SIZE=384
DEFINE_PREFIX=-D
JAVA_OPTIONS_BIN_DIR=${DEFINE_PREFIX}idc.bin.dir=$BIN_DIR
JAVA_OPTIONS_maxheap=${JAVA_MAX_HEAP_SIZE+-Xmx${JAVA_MAX_HEAP_SIZE\}m}
JAVA_OPTIONS_service=${IS_SERVICE+$JAVA_SERVICE_EXTRA_OPTIONS}
```

The `list` type computes a list of options for an executable. Each value that begins with *`variable_name_`* becomes a quoted option, and *`variable_name`* is set to the entire list. In this example, `JAVA_OPTIONS` is set to the string:

```
"-Didc.bin.dir=/intradocdir/bin/" "-Xmx384m"
```

- `path`

  Example 27–9 shows some `path` type variables.

Example 27–9 path Launcher Variables

```
IdcResourcesDir=${IdcResourcesDir-$IdcHomeDir/resources}
BASE_JAVA_CLASSPATH_source=$IdcResourcesDir/classes
BASE_JAVA_CLASSPATH_serverlegacy=$SharedDir/classes/server.zip
BASE_JAVA_CLASSPATH_server=$JLIB_DIR/idcserver.jar
```

The `path` type computes a path-like value.The value of each variable starting with *`variable_name_`* is appended to the value of *`variable_name`* separated by the value of `PATH_SEPARATOR`. In this example, `BASE_JAVA_CLASSPATH` is set to a very long class path.

- `lookupstring`

  Example 27–10 shows some `lookupstring` type variables.

Example 27–10 lookupstring Launcher Variables

```
TYPE_VDK_PLATFORM=lookupstring
PARAMETER_VDK_PLATFORM=${PLATFORM}_${UseVdkLegacySearch+vdk27}
VDK_PLATFORM_aix_vdk27=_rs6k41
VDK_PLATFORM_aix_=_rs6k43
```

```
VDK_PLATFORM_hpux_vdk27=_hpux11
VDK_PLATFORM_hpux_=_hpux11
VDK_PLATFORM_freebsd_vdk27=_ilnx21
VDK_PLATFORM_freebsd_=_ilnx21
VDK_PLATFORM_linux_vdk27=_ilnx21
VDK_PLATFORM_linux_=_ilnx21
VDK_PLATFORM_solaris_vdk27=_ssol26
VDK_PLATFORM_solaris_=_ssol26
VDK_PLATFORM_win32_vdk27=_nti40
VDK_PLATFORM_win32_=_nti40
```

The `lookupstring` type uses a second parameter to construct a lookup key for the final value. The second parameter is the value of `$PARAMETER_`*`variable_name`*. If this value is undefined, the current value of *`variable_name`* is used as the lookup key. In this example, `PARAMETER_VDK_PLATFORM` has the value of `${PLATFORM}_` or `${PLATFORM}_vdk27` depending on the value of `UseVdkLegacySearch`.

This value is then used to look up the value of the variable `VDK_PLATFORM_${PARAMETER_VDK_PLATFORM}` which is then enclosed in quotation marks and assigned to `VDK_PLATFORM`.

- lookuplist

  Example 27–11 shows some `lookuplist` type variables.

Example 27–11 lookuplist Launcher Variables

```
TYPE_STARTUP_CLASS=lookuplist
STARTUP_CLASS_version=Installer --version
STARTUP_CLASS_installer=Installer
STARTUP_CLASS_WebLayoutEditor=IntradocApp WebLayout
STARTUP_CLASS_UserAdmin=IntradocApp UserAdmin
STARTUP_CLASS_RepositoryManager=IntradocApp RepositoryManager
STARTUP_CLASS_Archiver=IntradocApp Archiver
STARTUP_CLASS_WorkflowAdmin=IntradocApp Workflow
STARTUP_CLASS_ConfigurationManager=IntradocApp ConfigMan
```

The `lookuplist` type uses a second parameter to construct a lookup key for the final value. The second parameter is the value of `$PARAMETER_`*`variable_name`*. If this value is undefined, the current value of *`variable_name`* is used as the lookup key.

Unlike `lookupstring`, `lookuplist` does not enclose the final value in quotation marks. For this example, the current value of `STARTUP_CLASS` is `version`. `STARTUP_CLASS` is replaced with the value `Installer --version`.

27.4.3 Launcher Environment Variables

After processing the computed settings, the Launcher iterates over all variables that begin with the string `EXPORT_`. The value of each variable is used as an environment variable name, which has the value of the second half of the `EXPORT_` variable assigned. For example, `EXPORT_IDC_LIBRARY_PATH=LD_LIBRARY_PATH` exports the value of the `IDC_LIBRARY_PATH` variable with the name `LD_LIBRARY_PATH`.

The variable `JAVA_COMMAND_LINE` is used to get the command line. Any command line arguments to the Launcher that have not been consumed are appended to the command line. On UNIX systems, the command line is parsed and quoting is undone and then `execv` is called. On Windows, a shutdown mutex is created and

CreateProcess is called with the command line. Care should be taken because CreateProcess does not undo backslash-quoting.

The principal mechanism for debugging the Launcher is to add the flag -debug before any arguments for the final command. You can also create a file named $BIN_DIR/debug.log which triggers debug mode and contain the debug output.

The Launcher has knowledge of the following configuration entries, which it either sets or uses to control its behavior. Note that these configuration variables might change or be removed in future releases of Content Server:

- IDC_SERVICE_NAME: the name of the win32 service used for service registration, unregistration, startup, and shutdown.
- IDC_SERVICE_DISPLAY_NAME: the display name of the win32 used for service registration.
- IntradocDir: the base directory for relative path names.
- IdcBaseDir: an alternate name for IntradocDir.
- IdcResourcesDir: set to $IdcHomeDir/resources if otherwise undefined.
- IdcNativeDir: defaults to $IdcHomeDir/native if otherwise unset.
- PATH_SEPARATOR: set to either colon (:) or semi-colon (;) if otherwise unset.
- STARTUP_CLASS: set to the name of the Launcher executable.
- MUTEX_NAME: the name used to create a shutdown mutex on win32.
- BEFORE_WIN_SERVICE_START_CMD: if set, is a command line that is executed before a win32 service starts.
- UseRedirectedOutput: if set tells the Launcher on win32 to redirect the output from the Java VM to a file.
- ServiceStartupTimeout: the time out used for waiting for a Java process to successfully start on win32.

Tip: By using Launcher.exe, changing the status.dat file, and altering the value of the JVM command line, you could theoretically run any Java program as a Windows service. This is not recommended for normal use, but it does explain some ways you could configure the Launcher.

27.4.4 User Interface

The UI for the Launcher is the same as the application it launches. For example, if the Launcher is renamed to IntradocApp, the following command-line arguments are specified to launch the Web Layout Editor:

```
IntradocApp WebLayout
```

This launches the Web Layout Editor as a standalone application.

By default, the application is launched without console output. However, when launching IdcServer, IdcAdmin, IdcCommandX, or the Installer, Java output is printed to the screen. In all other cases, the output is suppressed for a cleaner interface.

For some applications, such as the Batch Loader and the Repository Manager, it is desirable to view the Java output from the application. To force the Launcher to dump the Java output to the screen, use the `-console` flag in this manner:

```
IntradocApp RepMan -console
```

The output is now written to the console from which the Repository Manager was launched.

If the Launcher is renamed IdcServer, BatchLoader, SystemProperties, or any other Java class that requires no additional parameters, it can be launched with a simple double-click. In other cases, a shortcut can be used to launch them by double-clicking.

27.4.5 Configuring the Launcher

To use the Launcher, you must first rename the Launcher.exe file to an executable with the same name as the class file to be launched. Typical examples include `IdcServer.exe` and `IntradocApp.exe`.

> **Note:** If you want to make a custom application, you must create a custom directory and rename the `Launcher.exe` file to the service that is to be launched. A valid `intradoc.cfg` file must be in the same directory as the executable. The only required parameter is `IntradocDir`; however, you can include other entries to alter the way the Java application is launched.

27.4.6 Configuration File Example

You can modify the configuration file for the applications you need to run. Example 27–12 shows configuration file entries that are sufficient to launch nearly all Content Server applications.

Example 27–12 Configuration File Entries for Content Server Applications

```
<?cfg jcharset="Cp1252"?>
#Content Server Directory Variables
IntradocDir=C:/domain/idcm1/
BASE_JAVA_CLASSPATH_source=$IdcResourcesDir/classes
BASE_JAVA_CLASSPATH_serverlegacy=$SharedDir/classes/server.zip
BASE_JAVA_CLASSPATH_server=$JLIB_DIR/idcserver.jar
```

Other applications, such as Oracle WebCenter Content: Inbound Refinery, require additional classes in the class path. This file can also be modified to enable Content Server to be run with different Java Virtual Machines.

The `CLASSPATH` is designed to look for class files in order of the listed entries. In other words, the Launcher will search the entire *`DomainHome`*`/ucm/idc/native` directory before it looks in the `resources` directory or `server.zip` file. This is desirable if the users want to overload Java classes without patching the ZIP file.

Additionally, the Launcher can be used to install, uninstall, and run Java applications as Windows Services, if they follow the correct API for communicating back to the Launcher. For more details on how to make any Java application run as a Windows service with the Launcher, see the source code for `IdcServer.java` or `IdcAdmin.java`.

The `COMPUTEDCLASSPATH` is used to add class files to the `CLASSPATH` that the Launcher uses. To add class files, override this flag.

Note: The `intradoc.cfg` file is usually altered to include the locations of JDBC drivers for particular databases upon installation. If you want to use an alternate JDBC driver, place it outside of the *`IdcHomeDir`* directory for Content Server, and alter the `JDBC_JAVA_CLASSPATH_customjdbc` entry in the `intradoc.cfg` file with the location of the driver.

Example 27–13 shows a command to run Content Server with the IBM virtual machine on a Windows operating system.

Example 27–13 Command for Running Content Server with a Custom JVM

```
#customized for running the IBM VM
JAVA_EXE=full path
```

When using a custom JVM, specify the full path to the Java executable file to be used.

Caution: Avoid overriding the JVM command line. Customization is more complicated because of the custom class loader. If you do override the JVM command line, start with the `$IdcHomeDir/resources/core/config/launcher.cfg` file.

You can set `JAVA_COMMAND_LINE_SELECTION` entry in the configuration file to `idcclassloader` or `traditional`.

If you choose to change which JVM you are using, and if that VM has all the standard Sun SDK JAR files, then it is better to use the `J2SDK` configuration entry to relocate the root directory of the SDK directory rather than use `JAVA_EXE` to specify the location of the Java executable. (This is not applicable for the IBM VM.)

The `J2SDK` variable changes the directory where the Sun SDK libraries are found (such as `tools.jar`). If you change this entry without setting the `JAVA_EXE` entry, then Java executables are assumed to be in the `bin` directory of the path in `J2SDK`. The default value for `J2SDK` is `...\shared\os\win32\j2sdk1.4.2_04`.

To add a value to `JAVA_OPTIONS`, use `JAVA_OPTIONS_server=-server` or another similar value.

The following table describes commonly used command-line options. Those options noted with an asterisk (*) are available on a Windows operating system only. Unmarked options are available for a Windows or UNIX operating system.

Option	Description
`-console`	* Forces the Launcher to keep a Windows console window open so that the Java output and error streams are printed to the console.
`-debug`	Shows paths and variables in use at startup, and startup errors. Also enables Java debugging in Content Server; when repeated, this increases verbosity.
`-fileDebug`	Similar to the `-debug` option but this option dumps debug data to the debug.log file. It is usually only set in `JAVA_OPTIONS` or `JAVA_SERVICE_EXTRA_OPTIONS` in the intradoc.cfg file to debug Windows services.

Option	Description
`-install`	* Used to install the Java application referred to by the Launcher as a Windows Service.
`-install_autostart`	* Similar to the `-install` option but this option installs the application to start when the server starts.
`-uninstall`	* Used to uninstall the Java application referred to by the Launcher as a Windows Service.
`-remove`	* Same as `-uninstall`.
`-dependent` *service-name*	* Makes the Windows service dependent on whether the *service-name* service is also running. This command is useful when you want to make a dependent call for each service. For example, if you want to launch a database before starting Content Server, you can specify the Content Server startup to be dependent on the database startup.
`-dependent` *user password*	* Used with `-install`, installs the service with the credentials of the user specified by *user* with password *password*. This command will check the user regardless of the credentials, but may not install the service. The credentials of the user need to extend to the service for the auto-start to run the service automatically. For certain services, such as Inbound Refinery, the last flag is required so that the service can run with higher permissions. The user name must be in the typical Microsoft format DOMAIN\User. Once users change passwords, the service will not be able to log in, and therefore will not run.
`-help`	Provides verbose output on Launcher use.
`-version`	Displays the version number for the Launcher and exits.
`-asuser` *user password*	* Used during an install to install a service as a specified *user* with a specific *password*.
`-exec` *path _name*	Overrides the argv[0] setting. Used by the Launcher.sh to specify the target *path_name* because the target of the symlink does not know its source.
`-cfg` *configfilename*	Specifies additional config files to read before determining computed settings.
`-idcServiceName` *servicename*	* Specifies the name of the Windows service. This can used with `-remove` to uninstall another Content Server service without using that Content Server Launcher (for example, if an entire installation directory has been removed).

Tip: To customize the class path to alter the system path to load Oracle `.dll` files, you can set the path as follows:

`IDC_LIBRARY_PATH_`*`customfiles`*`=/`*`path-to-customfiles`*

Custom shared objects and `.dll` files must not be installed into *`IdcHomeDir`*.

If you want to load custom `.dll` files, you should put them in the *`IdcHomeDir`*`/native/win32/lib` directory.

27.5 Calling Services Remotely

To use services remotely, you must have these files on the remote system:

- *DomainHome*/ucm/cs/bin/IdcCommand.exe
- *DomainHome*/ucm/cs/bin/intradoc.cfg (same file as on Oracle WebCenter Content Server)
- *IntradocDir*/config/config.cfg

In addition, the following configuration entries must be defined in the `#Additional Variables` section of the config.cfg file on the remote system:

- IntradocServerPort=4444
- IntradocServerHostName=IP or DNS

28

Using the COM API for Integration

This chapter describes Microsoft Component Object Model (COM) integration. Oracle WebCenter Content Server utilizes a COM-based API, which provides the capability to call functionality from within a COM environment.

This chapter includes the following sections:

28.1 About the COM API

You can use a COM interface to integrate Content Management with Microsoft environments and applications. An ActiveX control and an OCX component are provided as interface options to gain access to the content and content management functions within Content Server. Additionally, you can communicate with ODMA-aware applications through a COM interface.

28.2 Calling Content Server Services with the IntradocClient OCX component

An Object Linking and Embedding Control Extension (OCX) control is provided for connecting to a remote Oracle WebCenter Content Server instance and executing Content Server services. The IdcClient OCX control is used within a Windows Visual Basic development environment to gain access to the content and content management functions within Content Server.

You can call Content Server services with the IdcClient OCX control. The `IdcClient.ocx` control is used to connect to a remote Oracle WebCenter Content Server instance and perform typical server functions.

Note: A Visual Basic or Visual C++ development environment is required for using the IdcClient OCX component.

28.2.1 OCX Interface

The IntradocClient OCX component is used within a Windows Visual Basic development environment to gain access to the content and content management functions within Content Server. The OCX integration is designed to call services in a visual development environment, or to connect to a remote Oracle WebCenter Content Server instance.

The IntradocClient OCX component provides functionality that you can access with a *method call*. Methods perform actions and often return results. Information is passed to methods using parameters. Some functions do not take parameters; some functions take one parameter; some take several.

The IntradocClient OCX component requires a username and password to execute the commands. The user must have the appropriate permissions to execute the commands. Some commands will require an administrative access level, other commands may require only write permission.

Outside of the `init` and `connection` managing methods, all methods use the serialized HDA format for communication. The returned serialized HDA format string contains information about the success or failure of the command. The `StatusCode` will be negative if a failure occurs, and `StatusMessage` indicates the error.

For more information, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*. This guide also contains information about the IntradocClient OCX API specifications listing the properties, methods, and events.

28.2.2 IdcClient OCX Description

IdcClient is an ActiveX control that allows a program to perform actions such as executing a service and retrieving file path information. The IdcClient control is also a wrapper for the Microsoft Internet Explorer browser.

The IdcClient OCX control is designed to use the Unicode standard and in most cases exchanges data with Oracle WebCenter Content Server in UTF-8 format. Unicode uses two bytes (16 bits) of storage per character and can represent characters used in a wide range of languages (for example, English, Japanese, Arabic). Since English language ASCII (American Standard Code for Information Interchange) characters only require one byte (8 bits), when an ASCII character is represented the upper byte of each Unicode character is zero.

See the Unicode Consortium on the Web for additional information about the Unicode standard at `http://www.unicode.org`.

Important: IdcClient OCX is built atop the Microsoft Layer for Unicode, which allows Unicode applications to run on Win9x platforms. When distributing the IdcClient OCX Control on 9x platforms, the "unicows.dll" must also be distributed. This companion DLL cannot be distributed on Windows-based systems.

In most cases, the methods use the serialized HDA format for communication. A serialized HDA format is a Java method used for communication. The returned serialized `HDA` format string contains information about the success or failure of the command.

The IdcClient OCX control provides functionality that can be performed with a method call. Methods perform actions and often return results. Information is passed to methods using parameters. Some functions do not take parameters; some functions

take one parameter; some take several. For example, a function with two parameters passed as strings would use this format:

```
Function(Parameter As String, Parameter As String) As String
```

- IdcClient OCX enables users to write client applications to execute services. The OCX control takes name/value pairs containing commands and parameters and calls the specified services. Execution results are passed back to the calling program.
- IdcClient OCX requires a user name and password to execute the commands. The user must have the appropriate permissions to execute the commands. Some commands will require an administrative access level, other commands may require only write permission.

For more information, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

28.2.2.1 OCX Events

Events are executed when the user or server performs an action. For example:

- The `IntradocBrowserPost` event executes every time a user submits a form from within a browser.
- The `IntradocServerResponse` event executes after the server completes a requested action.

28.2.2.2 OCX Methods

The Visual Basic Standard Controls provide methods that are common to every Visual Basic development environment. In addition, the IdcClient OCX control provides methods that are private and unique to this specific control. These methods are used to perform or initiate an action rather than setting a characteristic.

For example:

- The `AboutBox()` method launches the **About** box containing product version information.
- The `GoCheckinPage` method checks in a new content item or a content item revision.

28.2.2.3 OCX Properties

Properties describe or format an object and can be modified with code or by using the property window in the Visual Basic development environment. Properties describe the basic characteristic of an object.

For example:

- The `UserName` property provides the assigned user name.
- The `WorkingDir` property specifies the location where downloaded files are placed.

28.2.2.4 IdcClient OCX Interface

The IdcClient OCX control is used within a Windows Visual Basic development environment to gain access to the content and content management functions within Content Server. The OCX integration is designed to call services in a visual development environment, or to connect to a remote Content Server instance.

In most cases, methods use the serialized HDA format for communication. The returned serialized HDA format string contains information about the success or failure of the command. The `StatusCode` will be negative if a failure occurs, and `StatusMessage` will indicate the error. If the returned HDA does not contain a StatusCode parameter, the service call succeeded.

28.2.3 IdcClient OCX Control Setup

You can set up the IdcClient OCX component and create a visual interface in the Microsoft Visual Basic development environment.

28.2.3.1 Setting Up the IdcClient OCX Component

Follow these steps to set up the IdcClient OCX component in the Microsoft Visual Basic development environment:

1. Create a new project.
2. Select **Project**, and then choose **Components**.
3. Browse to the IdcClient.ocx file on your system, and click **Open**.

   The IdcClient module is added to the Component Controls list.
4. Ensure that the checkbox for the **IdcClient ActiveX Control** module is enabled, and click **OK**.

   The **IdcClient OCX** control is placed in the list of controls.
5. (Optional) You can use the Visual Basic development environment to build your own visual interface or follow the steps provided in Section 28.2.3.2, "Creating a Visual Interface," to build a basic visual interface.

28.2.3.2 Creating a Visual Interface

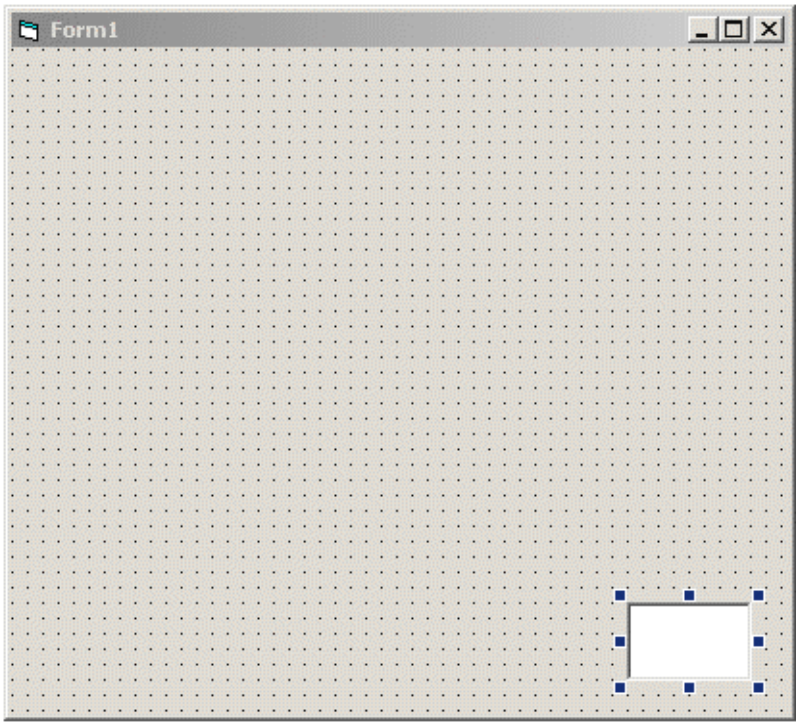
The following procedure for creating a visual interface is based on the assumption that a Visual Basic project has been created and the **IdcClient OCX** control has been placed in the list of controls. For more information, see Section 28.2.3.1, "Setting Up the IdcClient OCX Component."

Follow these steps to build a basic visual interface:

1. Select the control, and draw it on the Visual Basic form.

   Figure 28–1 shows the **IdcClient OCX** control.

Figure 28–1 OCX Control Drawn on a Visual Basic Form



2. From the drop-down list of the Properties window, choose **IdcClient OCX**.

   If the Properties window is not currently displayed, select **View**, and then choose **Properties Window** from the main menu.

3. Rename the **IdcClient OCX** control **IdcClientCtrl**.

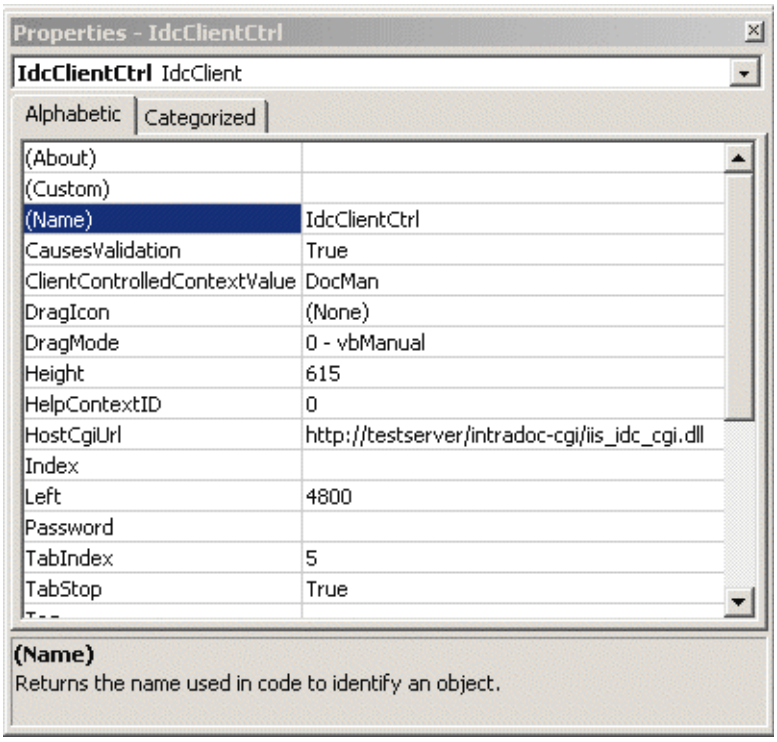
4. Define **HostCgiUrl** to reference the `iss_idc_cgi.dll` for your particular instance.

   For example:

   ```
   http://testserver/intradoc-cgi/iss_idc_cgi.dll
   ```

   Figure 28–2 shows this URL as the value of **HostCgiUrl**.

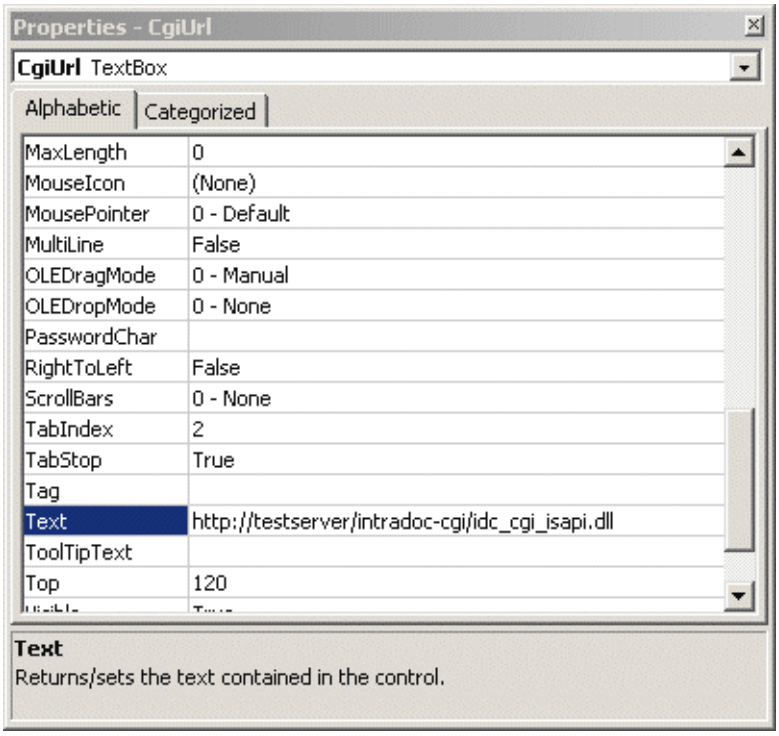
Figure 28–2 Edited IdcClient Properties



5. On the form, draw a text box, and name it `CgiUrl`.
6. For the **Text** field, enter the **HostCgiUrl** value as the text to be displayed, such as `http://testserver/intradoc-cgi/iss_idc_cgi.dll`.

   Figure 28–3 shows this URL as the **Text** value.

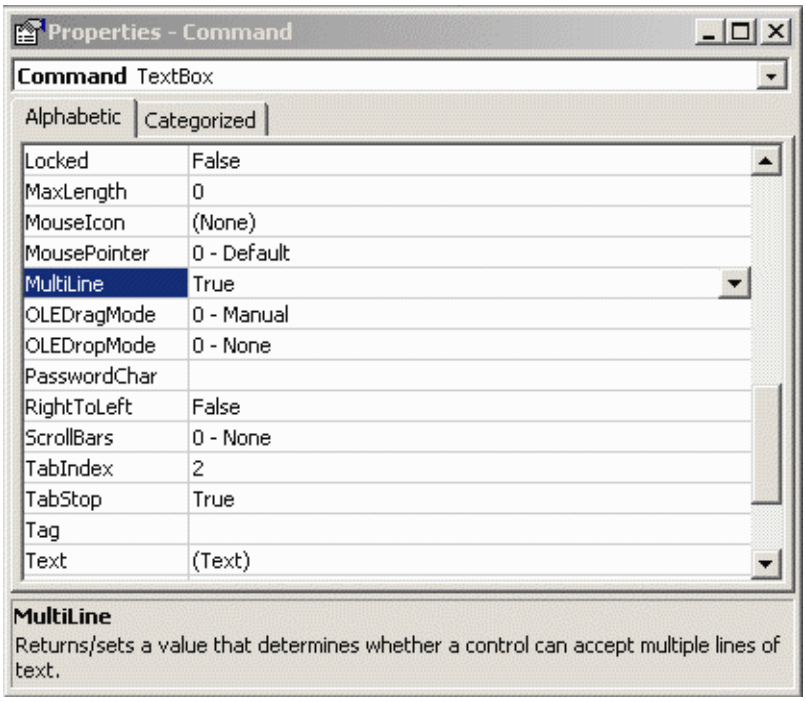
Figure 28–3 Edited CgiUrl TextBox Properties



7. On the form, draw a text box, and name it `Command`.
8. Clear the entry for the **Text** field (leave blank), and set **MultiLine** to `True`.

   Figure 28–4 shows a **MultiLine** value.

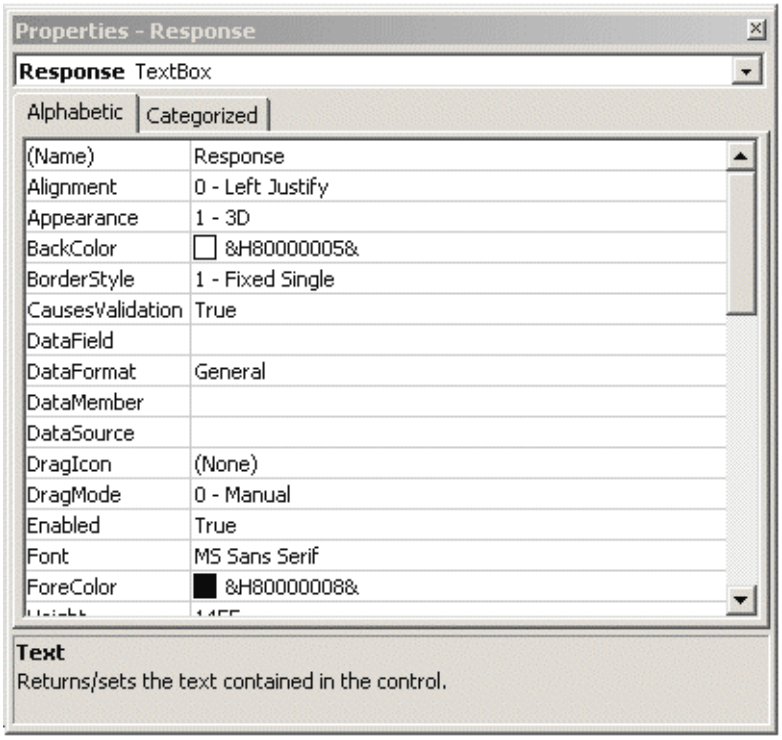
Figure 28–4 Edited Command TextBox Properties



9. On the form, draw a text box, and name it `Response`.
10. Clear the entry for the **Text** field (leave blank).

    Figure 28–5 shows field values for a `Response` text box.

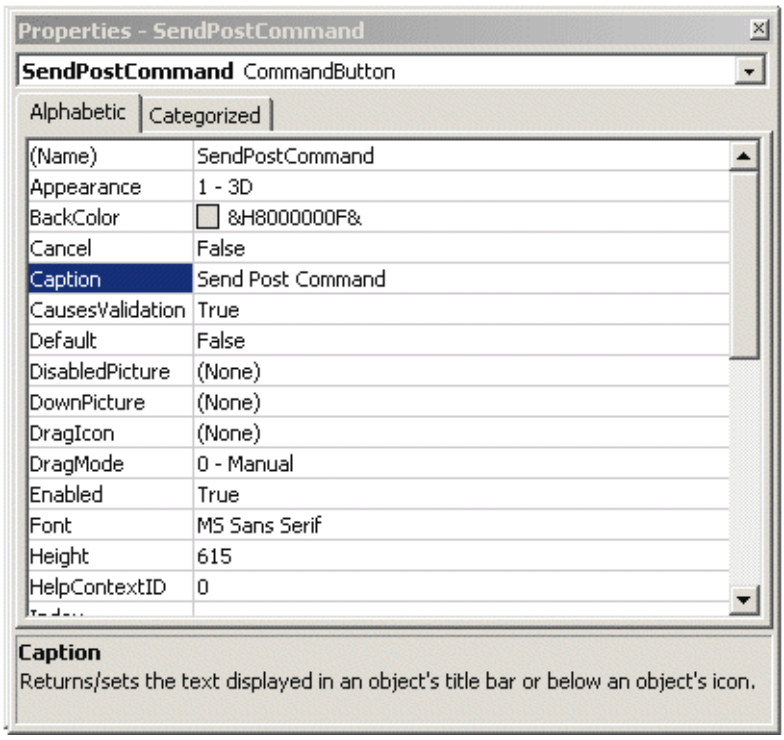
Figure 28–5 Edited Response TextBox Properties



11. On the form, draw a button, and name it `SendPostCommand`.
12. For the **Caption** field, enter `Send Post Command` as the text to be displayed.

    Figure 28–6 shows a **Caption** value to be displayed on a `SendPostCommand` button.

Figure 28–6 Edited SendPostCommand CommandButton Properties



13. On the form, select **View**, and then choose **Code**.
14. Select **SendPostCommand**, and then click the drop-down lists and modify the code to perform these actions:
    - Set the **Host Cgi URL** value.
    - Issue the command.
    - (Optional) Replace LF with CRLF to make the presentation in the edit control more readable.
    - Display the response.

    Modify the code as follows:

```
Dim R As String
IdcClientCtrl.HostCgiUrl = CgiUrl.Text
R = IdcClientCtrl.1.SendPostCommand(Command.Text)
R = Replace(R, vbLf, vbCrLf
Response.Text = R
```

    Figure 28–7 shows the code modifications.

Figure 28–7 Edited SendPostCommand_Click Code

```
Private Sub SendPostCommand_Click()
   Dim R As String

   ' Set the Host CGI Url
   IdcClientCtrl.HostCgiUrl = CgiUrl.Text

   ' Issue the command
   R = IdcClientCtrl.SendPostCommand(Command.Text)

   ' Optional--replace LF with CRLF here
    R = Replace(R, vbLf, vbCrLf)

   ' Display the response
   Response.Text = R

End Sub
```

15. Choose **Form** and then **Load** from the drop-down lists, and add the following lines to set the login prompt for the Oracle WebCenter Content Server instance:

    ```
    IdcClientCtrl.UseBrowserLoginPrompt = True
    IdcClientCtrl.UseProgressDialog = True
    ```

    Figure 28–8 shows the modified code.

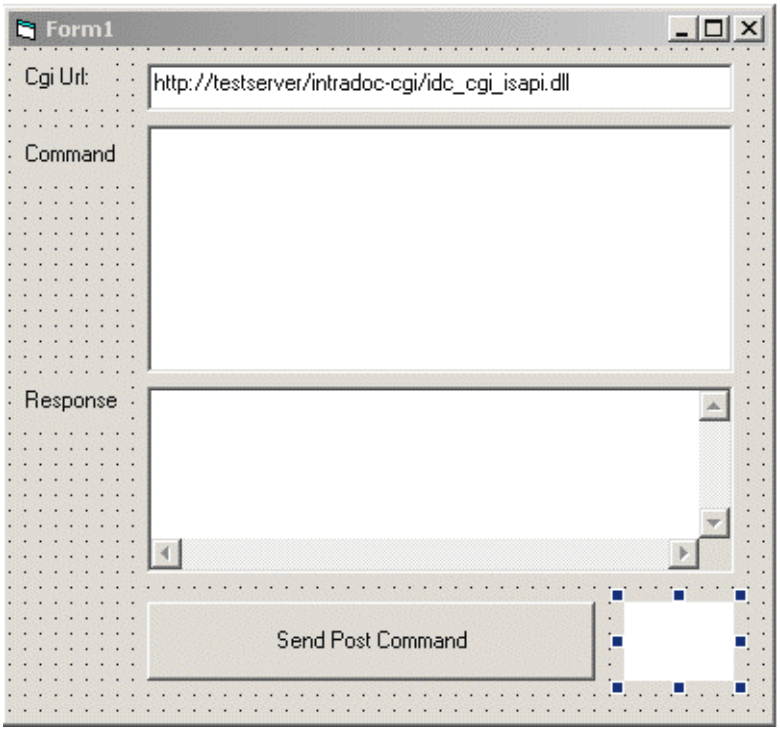
Figure 28–8 Edited Form_Load Code

```
Private Sub Form_Load()
   IdcClientCtrl.UseBrowserLoginPrompt = True
   IdcClientCtrl.UseProgressDialog = True
End Sub
```

16. (Optional) Add appropriate descriptive labels, such as **Cgi Url**, **Command**, and **Response**

    Figure 28–9 shows a form with descriptive labels.

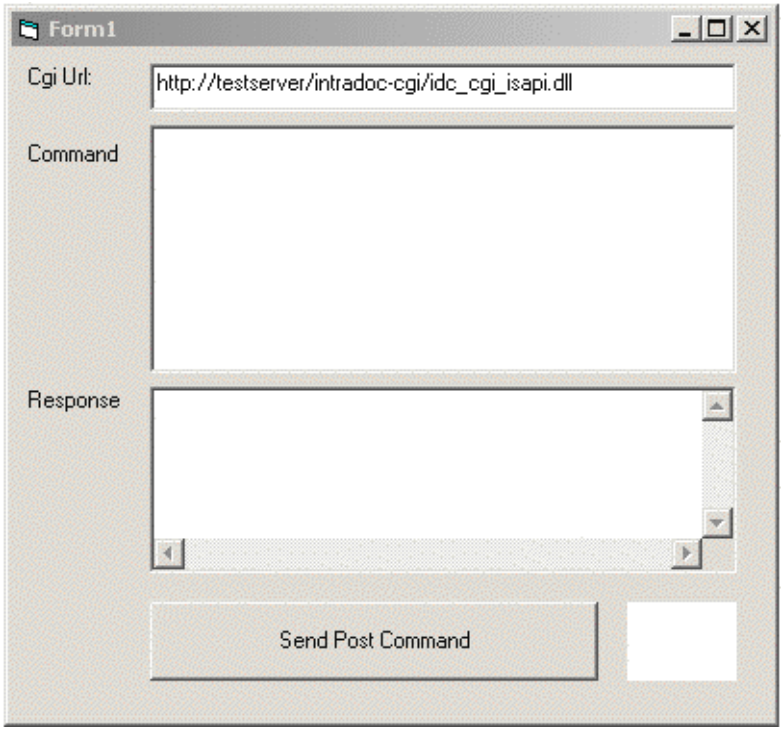
Figure 28–9 Visual Interface with a Descriptive Label



17. Select **Run**, and then choose **Start** to test the visual interface.

Figure 28–10 shows a successful test result.

Figure 28–10 Completed Visual Interface

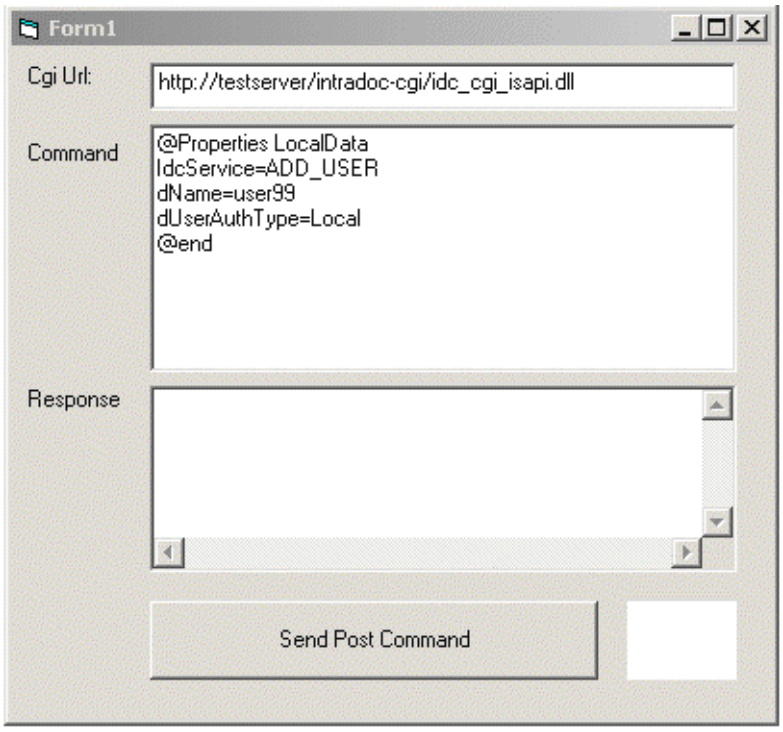


18. Enter a formatted command in the **Command** field.

Figure 28–11 shows the `ADD_USER` command to add a user.

For more information about the ADD_USER service, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

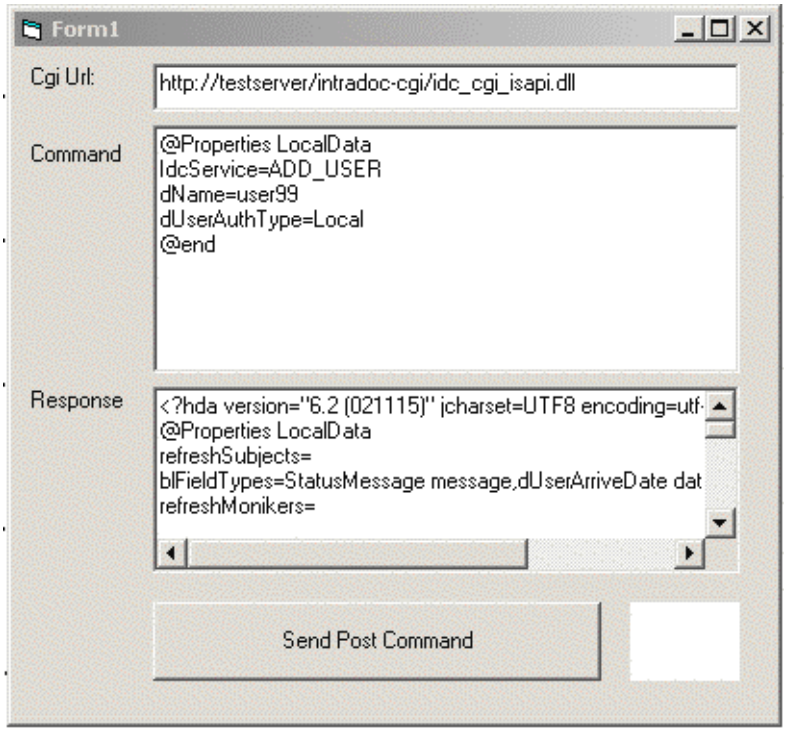
Figure 28–11 Visual Interface with Defined Command



19. Click the **Send Post Command** button to execute the command. The returned results are displayed in the **Response** field.

    Figure 28–12 shows some returned results.

Figure 28–12 Visual Interface with Returned Results



To verify the command:

1. In a web browser, log in to Oracle WebCenter Content Server as an administrator.
2. In the **Administration** tray, choose **Admin Applets**.
3. Click **User Admin**. The applet launches and displays the added user (for example, `user99`).

28.2.4 IdcClient Events

Events are executed when the user or server performs an action. The following IdcClient OCX events are available:

- IntradocBeforeDownload
- IntradocBrowserPost
- IntradocBrowserStateChange
- IntradocRequestProgress
- IntradocServerResponse

28.2.4.1 IntradocBeforeDownload

Executes before a file is downloaded.

- Initiates the server actions and updates required before a download.

Parameters

The event passes these parameters:

- ByVal `params` As String
- `cancelDownload` As Boolean

28.2.4.2 IntradocBrowserPost

Executes every time a form is submitted from within a browser.

Parameters

The event passes these parameters:

- ByVal `url` As String
- ByVal `params` As String
- `cancelPost` As Boolean

28.2.4.3 IntradocBrowserStateChange

Executes whenever the browser state changes.

Parameters

The event passes these parameters:

- ByVal `browserStateItem` As String
- ByVal `enabled` As Boolean

28.2.4.4 IntradocRequestProgress

Executes a request for a progress report to be sent from the server. This event occurs only after a method has been called.

Parameters

The event passes these parameters:

- ByVal `statusData` As String
- ByVal `isDone` As Boolean

28.2.4.5 IntradocServerResponse

Executes after the server completes a requested action. For example, after a file has been downloaded. This event handles HDA encoded data that is a response from the server. This event only occurs when an action is performed in the browser.

Parameters

The event passes one parameter:

- ByVal `response` As String

28.2.5 IdcClient OCX Methods

The following IdcClient OCX methods are available:

- AboutBox
- Back
- CancelRequest*
- DoCheckoutLatestRev
- DownloadFile
- DownloadNativeFile

- Drag
- EditDocInfoLatestRev
- Forward
- GoCheckinPage
- Home
- InitiateFileDownload*
- InitiatePostCommand*
- Move
- Navigate
- NavigateCgiPage
- Refresh Browser
- SendCommand*
- SendPostCommand*
- SetFocus
- ShowDMS
- ShowDocInfoLatestRev
- ShowWhatsThis
- StartSearch
- Stop
- UndoCheckout
- ViewDocInfo
- ViewDocInfoLatestRev
- ZOrder

Methods marked with an asterisk (*) are ones which are not related to browser activity and which return a value.

> **Important:** All parameters are required unless otherwise indicated.

28.2.5.1 AboutBox

```
Sub AboutBox()
```

Description

Launches the **About** box containing product version information.

- This method displays the product **About** box.
- The method returns `FALSE` if the call cannot be executed.

Parameters

None

28.2.5.2 Back

```
Sub Back()
```

Description

Displays the previous HTML page.

- Returns the user to the previous screen.
- The method retrieves the previous HTML page from cached information for display to the user.

Parameters

None

28.2.5.3 CancelRequest

```
Function CancelRequest() As Boolean
```

Description

This method cancels the currently active request. Returns `FALSE` if the function is unable to cancel the request or if there is no request currently active.

Parameters

None

Output

Returns a Boolean value:

- Returns TRUE if request is canceled.
- Returns FALSE if the cancel request is not performed.

28.2.5.4 DoCheckoutLatestRev

```
Sub DoCheckoutLatestRev(docName As String, curID As String)
```

Description

Checks out or locks the latest content item revision.

- Given a content item name and the version label, the method checks out the latest content item revision.
- Executes the `IntradocServerResponse` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events."

> **Note:** The `curID` value is the content item version label, not the generated content item revision ID.

This function returns the following values:

- Serialized HDA containing `dID` and `dDocName`.
- `FALSE` if the latest revision cannot be checked out or cannot be found in the system.
- The data that was passed in as parameters.

Parameters

- docName: The user-assigned content item name.
- curID: The unique identifier for the latest revision. Optional.

28.2.5.5 DownloadFile

```
Function DownloadFile(command As String, filename As String) As String
```

Description

Downloads the defined file.

- Given a currently associated command and the file type, this method performs a file download of the postconversion file (compare `DownloadNativeFile`).
- Executes the `IntradocBeforeDownload` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events."

This function returns the following:

- Serialized HDA containing the status code and status method.
- The data that was passed in as parameters.
- FALSE if it is unable to download the specified file.

Parameters

- command: The currently associated command.
- filename: The file format. This is the file type such as PDF, HTM, or other supported format.

28.2.5.6 DownloadNativeFile

```
Function DownloadNativeFile(id As String, docName As String, filename As String) As String
```

Description

Downloads the defined native file.

- Given a content item revision ID, a content item name, and a file type, this method performs a file download of the native file (compare `DownloadFile`).
- Executes the `IntradocBeforeDownload` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events.".

> **Note:** The `id` value is the generated content item revision ID, not the content item version label.

This function returns the following:

- Serialized HDA containing `dID` and `dDocName`.
- The data that was passed in as parameters.
- FALSE if it is unable to download the specified file.

Parameters

- id: The unique identifier for the latest revision.
- docName: The user-assigned content item name.
- filename: The file format. This is the file type such as DOC, RTF, or any other supported format.

28.2.5.7 Drag

```
Sub Drag([nAction])
```

Description

Begins, ends, or cancels a drag operation.

- The `Drag` method is handled the same as a Standard Control implementation.
- Refer to a Visual Basic API reference for additional information.

Parameters

- nAction: Indicates the action to perform. If you omit `nAction`, `nAction` is set to 1.

The settings for the `Drag` method are:

- 0: Cancel drag operation; restore original position of control.
- 1: (Default) Begin dragging the control.
- 2: End dragging, that is, drop the control.

28.2.5.8 EditDocInfoLatestRev

```
Sub EditDocInfoLatestRev(docName As String, curID As String, activateAction As String)
```

Description

Edits the content item information for the latest revision.

- ODMA related.
- Given a content item name, the version label, and the currently active requested action, the method edits the content item information for the latest revision.
- The function returns FALSE if the content item information for the latest revision cannot be edited or cannot be found in the system.

> **Note:** The `curID` value is the content item version label, not the generated content item revision ID.

Parameters

- curID: The unique identifier for the latest revision.
- activateAction: Passed to ODMActivate. This can be used as Idoc Script. *Optional.*
- docName: The user-assigned content item name. *Optional*.

28.2.5.9 Forward

```
Sub Forward()
```

Description

Displays the next HTML page.

- Moves the user to the next screen.
- This method retrieves cached information for the next HTML page for display to the user.

Parameters

None

28.2.5.10 GoCheckinPage

```
Sub GoCheckinPage(id As String, docName As String, isNew As Boolean, params As String)
```

Description

Checks in a new content item or a content item revision.

- Given the content item revision ID and the content item name, the function checks in a new content item or a content item revision.
- This method opens the content item check-in page and enters the unique content item identifier, user-assigned content item name, and any assigned content item parameters into the associated text fields. It is also specified whether this is a new content item or a revision.

> **Note:** The `id` value is the generated content item revision ID, not the content item version label.

Output

This function returns the following:

- FALSE if it is unable to check in the specified file.
- Serialized HDA containing `dID` and `dDocName`.
- The data that was passed in as parameters.

Parameters (All Optional)

- id: The unique identifier for the latest revision.
- docName: The user-assigned content item name.
- IsNew: Defines whether the content item to be checked in is a new content item or a revision.
  - If TRUE, a new unique content item version label is assigned.
  - Default is TRUE.
- params: The parameters that prefill the Check In page.

28.2.5.11 Home

```
Sub Home()
```

Description

Returns the user to the defined home page.

- Moves the user to the home screen.
- Executes an HTML page request and displays the defined home page to the user.

Parameters

None

28.2.5.12 InitiateFileDownload

```
Function InitiateFileDownload(command As String, filename As String) As String
```

Description

Initiates a file download.

- Given the currently associated command and the file type, the function initiates a file download. This method initiates a file download of a specific rendition of a content item, the latest revision, or the latest released revision.
- Executes the `IntradocServerResponse` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events."

Parameters

- `command`: The currently associated command.
- `filename`: The file format. This is the file type, such as `PDF`, `HTM`, or another supported format.

Output

- Returns serialized HDA containing the requested information.
- Returns the data that was passed in as parameters.

28.2.5.13 InitiatePostCommand

```
Function InitiatePostCommand(postData As String) As String
```

Description

Initiates a post command.

- Initiates a service call. Given assigned post data, this method initiates a post command.
- Executes the `IntradocServerResponse` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events."

Parameters

- postData: The serialized HDA containing the service command and any necessary service parameters.

Output

- Returns serialized HDA containing the requested information.
- Returns `StatusCode` and `StatusMessage`.
  - The StatusCode will be negative if a failure occurs, and StatusMessage will indicate the error.
  - If the returned HDA does not contain a StatusCode parameter, the service call succeeded.

28.2.5.14 Move

```
Sub Move(Left As Single, [Top], [Width], [Height])
```

Description

Moves an object.

- The `Move` method is handled the same as a Standard Control implementation.
- Refer to a Visual Basic API reference for additional information.

Parameters

- nLeft: Specifies the horizontal coordinate for the left edge of the object. This is a single-precision value.
- nTop: Specifies the vertical coordinate for the top edge of the object. This is a single-precision value.
- nWidth: Specifies the new width of the object. This is a single-precision value.
- nHeight: Specifies the new height of the object. This is a single-precision value.

28.2.5.15 Navigate

```
Sub Navigate(url As String
```

Description

Computes the URL path.

- Given a complete URL, this method computes the URL from the serialized HDA and returns the value as a string.

This function returns the following values:

- Serialized HDA containing the requested information.
- The data that was passed in as parameters.

Parameters

- url: The complete URL path.

28.2.5.16 NavigateCgiPage

```
Sub NavigateCgiPage(params As String)
```

Description

Computes the CGI path.

- Given defined content item parameters, this method computes the CGI path from the serialized HDA and returns the value as a string.

Parameters

- params: The assigned content item parameters.

28.2.5.17 Refresh Browser

Description

Refreshes the browser.

- This method refreshes the web browser and updates dynamic information.

Parameters

None

28.2.5.18 SendCommand

```
Function SendCommand(params As String) As String
```

Description

Issues a service request to Oracle WebCenter Content Server.

- Given defined content item parameters, the function executes a service from Oracle WebCenter Content Server related to content item handling.

Parameters

- params: The CGI URL encoded parameters.

Output

- Returns serialized HDA containing the requested information.
- Returns the data that was passed in as parameters.

28.2.5.19 SendPostCommand

```
Function SendPostCommand(postData As String) As String
```

Description

Sends a post command.

- Executes a service call.
- Executes the `IntradocBrowserPost` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events."

Parameters

- postData: The serialized HDA containing the service command and any necessary service parameters.

Output

- Returns serialized HDA containing the requested information.
- Returns `StatusCode` and `StatusMessage`.
  - The `StatusCode` will be negative if a failure occurs, and `StatusMessage` will indicate the error.
  - If the returned HDA does not contain a StatusCode parameter, the service call succeeded.

28.2.5.20 SetFocus

```
Sub SetFocus()
```

Description

Assigns the focus to a control.

- The `SetFocus` method is handled the same as a Standard Control implementation.
- Refer to a Visual Basic API reference for additional information.

Parameters

None

28.2.5.21 ShowDMS

```
Sub ShowDMS()
```

Description

Opens the HTML page associated with the Content Manager.

- ODMA related.
- Displays the Content Manager access page in a browser.

Parameters

None

28.2.5.22 ShowDocInfoLatestRev

```
Sub ShowDocInfoLatestRev(docName As String, curID As String, activateAction As String)
```

Description

Displays the content item information for the latest revision.

> **Note:** The `curID` value is the content item version label, not the generated content item revision ID.

Parameters

- docName: The user-assigned content item name.
- curID: The unique identifier for the latest revision. *Optional.*
- activateAction: The currently active requested action. *Optional.*

28.2.5.23 ShowWhatsThis

```
Sub ShowWhatsThis()
```

Description

Displays the *What's This Help* topic specified for an object with the WhatsThisHelpID property.

- The `ShowWhatsThis` method is handled the same as a Standard Control implementation.
- Refer to a Visual Basic API reference for additional information.

Parameters

- Object: Specifies the object for which the What's This Help topic is displayed.

28.2.5.24 StartSearch

```
Sub StartSearch()
```

Description

Displays the query page in the browser control.

- Preforms browser manipulation.

Parameters

None

28.2.5.25 Stop

```
Sub Stop()
```

Description

Stops the browser.

- This method stops or cancels the loading of information in the browser.

Parameters

None

28.2.5.26 UndoCheckout

```
Sub UndoCheckout(docName As String, curID As String)
```

Description

This service reverses a content item checkout.

- Given a content item name and a version label, this service attempts to locate the content item in the system and undo the check out. The service fails if the content item does not exist in the system, if the content item is not checked out or the user does not have sufficient privilege to undo the checkout.
- Executes the `IntradocServerResponse` event. The event is executed before the method occurs. For details, see Section 28.2.4, "IdcClient Events."

> **Note:** The `curID` value is the content item version label, not the generated content item revision ID.

Parameters

- curID: The unique identifier for the latest revision.
- docName: The user-assigned content item name. *Optional.*

28.2.5.27 ViewDocInfo

```
Sub ViewDocInfo(id As String)
```

Description

Navigates to the content item information page and displays content item information in a browser.

- Performs browser manipulation.
- Given a content item revision ID, the method displays content item information in a browser.

> **Note:** The `id` value is the generated content item revision ID, not the content item version label.

Parameters

- id: The unique identifier for the latest revision.

28.2.5.28 ViewDocInfoLatestRev

```
Sub ViewDocInfoLatestRev(docName As String, curID As String)
```

Description

Navigates to the content item information page and displays content item information for the latest revision.

- Given a content item name and a version label, the method displays the content item information for the latest revision.

> **Note:** The `curID` value is the content item version label, not the generated content item revision ID.

This function returns the following values:

- Serialized HDA containing `dID` and `dDocName`.
- The data that was passed in as parameters.

Parameters

- docName: The user assigned content item name.
- curID: The unique identifier for the latest revision.

28.2.5.29 ZOrder

```
Sub ZOrder([Position])
```

Description

Places a specified form or control at the front or back of the z-order within its graphical level.

- The `ZOrder` method is handled the same as a Standard Control implementation.
- Refer to a Visual Basic API reference for additional information.

Parameters

- `nOrder`: Specifies an integer indicating the position of the object relative to other objects. If you omit `nOrder`, the setting is `0`.

The settings for the `ZOrder` method follow:

- 0: (Default) The object is positioned at the front of the z-order.
- 1: The object is positioned at the back of the z-order.

28.2.6 IdcClient Properties

Each data item or *attribute* is implemented as a *property* in Visual Basic. Properties are exposed through the Public Interface of an object within the Visual Basic development environment. These attributes can be used to further describe elements.

These are the IdcClient OCX properties:

- ClientControlledContextValue
- HostCgiUrl
- Password
- UseBrowserLoginPrompt
- UseProgressDialog
- UserName
- Working Directory

28.2.6.1 ClientControlledContextValue

Provides the user-supplied context value. This value becomes available to Idoc Script as the variable `ClientControlled` in any web page delivered by Oracle WebCenter Content Server.

- Returns the value as a string.
- Takes no parameters.

28.2.6.2 HostCgiUrl

Provides the complete URL path of the host CGI bin.

- Returns the value as a string.
- Takes no parameters.

28.2.6.3 Password

Provides the assigned user password.

- Returns the value as a string.
- Takes no parameters.

28.2.6.4 UseBrowserLoginPrompt

Allows the use of a browser login prompt. Defines whether a dialog box for user authentication will display.

- If set to `TRUE`, control will open a dialog box for user authentication.
- The default value is `TRUE`.

Returns a Boolean value:

- `TRUE` if the login was successful
- `FALSE` if the login was denied

28.2.6.5 UseProgressDialog

Enables the use of a user progress dialog. Defines whether a dialog box for user authentication will display.

- If set to `TRUE`, control will open a dialog box for user progress.
- Default is `TRUE`.

Returns a Boolean value:

- Returns `TRUE` if the action was completed.
- Returns `FALSE` if the action failed.

28.2.6.6 UserName

Provides the assigned user name.

Returns the value as a string.

Takes no parameters.

28.2.6.7 Working Directory

Specifies the working directory as a full path. This is the location where downloaded files are placed.

- Returns the value as a string.
- Takes no parameters.

28.3 Using the ODMA API to Access Content Server from a Desktop Application

The Open Document Management Application (ODMA) is a standard API used to interface between desktop applications and file management software. The ODMA integration for Content Server is available with Desktop, a separate product. Use the ODMA-integration products to gain access to the content and content management functions within Content Server (for ODMA-compliant desktop applications).

You can publish files to your web repository directly from any ODMA-compliant application, such as Microsoft Word, Corel WordPerfect, and Adobe FrameMaker. With the web-centric adoption of ODMA, you can check in and publish information directly to the Web. This is a significant advancement over traditional ODMA client/server implementations, where information is published first to a server and is not immediately available on the Web for consumption.

For more information, refer to the ODMA or ODMA/FrameMaker online help.

28.3.1 ODMA Client

The ODMA Client is a separate product and does not ship with the core product. It is used to check in and publish information directly to the Web from your desktop applications. ODMA Client surpasses traditional ODMA client–server models, which publish information to a server and not immediately to the Web for consumption. You can use ODMA Client from within your desktop application to perform many tasks which interact with Oracle WebCenter Content Server, for example:

- Save a file and immediately check it in to Oracle WebCenter Content Server.
- Save a file to check in later.
- Check out a file from Oracle WebCenter Content Server.
- Update a file's metadata (content information).
- Save the file to your local file system and bypass the ODMA Client system.

28.3.2 ODMA Interfaces

These ODMA interfaces are available:

- **ODMA Client Interface**: The Select Document screen with the **Recent Files** option selected displays a list of files that you recently used through ODMA. This screen is displayed instead of the typical Open dialog box. If a file does not display on this screen, you can search for it in Content Server or on the local file system.
- **ODMA Desktop Shell Interface**: The Client Desktop Shell provides a *drag-and-drop* check-in functionality, and access to the ODMA Client - Select Document screen from outside of your desktop application. Through the Desktop Shell, you can:
  - Select a file from your desktop or a Windows Explorer window and drag it to the Desktop Shell to check in the file to Content Server.
  - Choose and open a file from the **Recent Files** list or from Content Server.
- **Content Server Interface with ODMA**: You can open and check out an ODMA file directly from the Content Server Content Information page. When you open a file from Content Server, the file opens in its native application so that you can edit it and quickly check the file back in to Content Server.

Note: You can also open and check out a file from within an ODMA-compliant application, and you can open a copy of a file instead of checking it out. For more information, see the ODMA Online Help.

29

Using RIDC to Access Content Server

This chapter describes how to initialize and use Remote Intradoc Client (RIDC), which provides a thin communication API for communication with Oracle WebCenter Content Server. For more information, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*.

The Remote Intradoc Client (RIDC) can be downloaded from the Oracle Technology Network (OTN) at `http://www.oracle.com/technetwork/index.html`.

> **Note:** Remote Intradoc Client (RIDC) 11*g*R1 (11.1.1.6.0 or later) requires Java Runtime Environment (JRE) 1.6 or later. The current Java JRE/JDK can be downloaded from the Oracle Technology Network (OTN) at `http://www.oracle.com/technetwork/index.html`.

This chapter includes the following sections:

29.1 About Remote Intradoc Client

Remote Intradoc Client (RIDC) is a thin communication API for talking to Content Server. It's main functionality is to provide the ability to remotely execute Content Server services. In addition, RIDC handles things like connection pooling, security, and protocol specifics.

Key Features

Remote Intradoc Client (RIDC) has these features:

- Supports Intradoc socket-based communication and the HTTP and JAX-WS protocols.
- Supports Secure Socket Layer (SSL) communication with Content Server.
- Provides client configuration including setting the socket time outs, connection pool size, and so on.
- RIDC objects follow the standard Java Collection paradigms.

Supported Protocols

Remote Intradoc Client (RIDC) 11*g*R1 (11.1.1.7.0) supports the `idc`, `idcs`, `http`, `https`, and `jax-ws` protocols.

Intradoc: The Intradoc protocol communicates to the Content Server over the over the Intradoc socket port (typically `4444`). This protocol requires a trusted connection between the client and Content Server and will not perform any password validation. Clients that use this protocol are expected to perform any required authentication themselves before making RIDC calls. The Intradoc communication can also be configured to run over SSL.

HTTP: RIDC can create an HTTP connection to Content Server using one of three supported HTTP client packages:

- Oracle HTTPClient
- Apache HttpClient version 3 (the default)
- Apache HttpClient version 4

Unlike Intradoc, this protocol requires valid user name and password authentication credentials for each request.

For details, see Section 29.1.1, "HttpClient Libraries". For additional information, see the Jakarta Commons HttpClient documentation on the HttpClient Home page of the Apache HttpClient website at

`http://hc.apache.org/`

JAX-WS: The JAX-WS protocol is supported only in Oracle WebCenter Content 11*g* with a properly configured Content Server instance and the RIDC client installed. JAX-WS is not supported outside this environment.

For more information about JAX-WS, see *Oracle Fusion Middleware Getting Started With JAX-WS Web Services for Oracle WebLogic Server* and *Oracle Fusion Middleware Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*. Also see the *Java API for XML Web Services (JAX-WS)* documentation on the Java Community Process website:

`http://www.jcp.org/`

Supported URL Formats

The following table shows the URL formats that are supported.

URL	Description
`idc://localhost:4444`	Uses the Intradoc port; requires only the hostname and the port number.

URL	Description
idcs://localhost:4443	Uses SSL over the Intradoc port; requires extra configuration to load the SSL certificates.
http://localhost:16200/cs/idcplg	Specifies the URL to the Content Server CGI path.
https://localhost:16200/cs/idcplg	Uses SSL over HTTP; requires extra configuration to load the SSL certificates.
http://wlsserver:16200/idcnativews	Uses the JAX-WS protocol to connect to the Content Server.

Required Environments

The following table summarizes the environment RIDC needs to support each connection type.

URL	Description
idc://	■ oracle.ucm.ridc-11.1.1.jar
idcs:/	■ oracle.ucm.ridc-11.1.1.jar ■ SSL certificate configuration
http:/	■ oracle.ucm.ridc-11.1.1.jar ■ HttpClient libraries
https:/	■ oracle.ucm.ridc-11.1.1.jar ■ HttpClient libraries ■ SSL certificate configuration
jax-ws	■ Oracle shiphome having WLS and JRF stacks

29.1.1 HttpClient Libraries

RIDC requires supporting HTTP client libraries to communicate with the web server attached to the Content Server instance using an HTTP connection. Currently three libraries are supported:

- Oracle HTTPClient
- Apache (Jakarta Commons) HttpClient version 3
- Apache (HttpComponents) HttpClient version 4

Apache HttpClient version 3 is the default.

To request the Oracle HttpClient in Java code:

```
IdcClient idcClient = manager.createClient("http://localhost/cs/idcplg");
idcClient.getConfig ().setProperty ("http.library", "oracle");
```

To request Apache HttpClient version 4 in Java code:

```
IdcClient idcClient = manager.createClient("http://localhost/cs/idcplg");
idcClient.getConfig ().setProperty ("http.library", "apache4");
```

If you are creating a new RIDC application using the JDeveloper extension, you can add to your connection, in the Configuration Parameters section, the parameter `http.library` with an appropriate value, such as `apache4`. For additional

information, see the Jakarta Commons HttpClient documentation on the HttpClient Home page of the Apache HttpClient website at

http://hc.apache.org/

If you are in a Site Studio for External Applications (SSXA) application in JDeveloper, because there is no user interface, you need to create your connection and save it without testing the connection first. Then open the `connections.xml` file in the **Connections** > **Descriptors** > **ADF META-INF** node. Add the `StringRefAddr` section (shown in Example 29–1) to the `connections.xml` file, and save the file.

Example 29–1 Connection Example in connections.xml

```
<Reference name="sample"
 className="oracle.stellent.ridc.convenience.adf.mbeans.IdcConnection" xmlns="">
  <Factory className=
  "oracle.stellent.ridc.convenience.adf.mbeans.IdcConnectionFactory"/>
  <RefAddresses>
    <StringRefAddr addrType="oracle.stellent.idc.connectionUrl">
       <Contents>idc://localhost:4444</Contents>
    </StringRefAddr>
    <StringRefAddr addrType="oracle.stellent.idc.idcServerURL">
       <Contents>http://localhost/cs/idcplg</Contents>
    </StringRefAddr>

    <StringRefAddr addrType="oracle.stellent.idc.http.library">
        <Contents>apache4</Contents>
    </StringRefAddr>

  </RefAddresses>
</Reference>
```

Note that the connection types for SSXA and RIDC are similar:

- When you are using SSXA connections in JDeveloper, the `addrType` value in the `connections.xml` file is `oracle.stellent.idc.http.library`.
- When you are using RIDC connections in JDeveloper the `addrType` value in the `connections.xml` file is `oracle.stellent.ridc.http.library`.

29.1.2 Convenience Classes

There are some patterns of actions that many applications perform using RIDC. The convenience package supplies some of these for reuse. The classes in the convenience package space are consumers of the RIDC code and as such don't add any new functionality. They can be thought of as a new layer on top of RIDC.

For information about using convenience classes, see Section 29.9, "Setting User Security."

29.2 Initializing Connections

This section provides sample code to initialize an Intradoc connection, an HTTP connection, and code that initializes a JAX-WS client.

The code in Example 29–2 initializes an Intradoc connection.

Example 29–2 Intradoc Connection Initialization

```
// create the manager
IdcClientManager manager = new IdcClientManager();

// build a client that will communicate using the intradoc protocol
IdcClient idcClient = manager.createClient("idc://localhost:4444");
```

The code in Example 29–3 initializes an HTTP connection. The only difference from an Intradoc connection is the URL.

Example 29–3 HTTP Connection Initialization

```
// create the manager
IdcClientManager manager = new IdcClientManager();

// build a client that will communicate using the HTTP protocol
IdcClient idcClient = manager.createClient("http://localhost:16200/cs/idcplg");
```

The code in Example 29–4 initializes a JAX-WS client. The URL includes the `idcnativews` web context root. This web context root (by default) is used by two web services exposed by Content Server: the `login` service and the `request` service.

Example 29–4 JAX-WS Client Initialization

```
// create the manager
IdcClientManager manager = new IdcClientManager();

// build a client that will communicate using the JAXWS protocol
IdcClient idcClient = manager.createClient
      ("http://wlsserver:16200/idcnativews");
```

29.3 Configuring Clients

Configuration of the clients can be done after they are created. Configuration parameters include setting the socket timeouts, connection pool size, and so on. The configuration is specific to the protocol; if you cast the IdcClient object to the specific type, then you can retrieve the protocol configuration object for that type.

29.3.1 Configuring Clients for Intradoc Connections

The code in Example 29–5 sets the socket time-out and wait time for Intradoc connections.

Example 29–5 Client Configuration for Intradoc Connections

```
// build a client as cast to specific type
IntradocClient idcClient =
  (IntradocClient)manager.createClient("http://localhost/cs/idcplg");

// get the config object and set properties
idcClient.getConfig().setSocketTimeout(30000);  // 30 seconds
idcClient.getConfig().setConnectionSize(20);    // 20 connections
```

29.3.2 Configuring SSL

Remote Intradoc Client (RIDC) allows Secure Socket Layer (SSL) communication with Content Server using the Intradoc communication protocol. The typical port used is 4444. For more information about configuring SSL and enabling ports, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

For SSL communication, you must install and enable the SecurityProviders component in the Content Server instance that you want to access. You must configure Content Server for SSL communication with a new incoming provider, and specify the truststore or keystore information. You must have a valid keystore or trust manager with signed, trusted certificates on both the client and Content Server.

Oracle does not provide signed certificates. For most implementations, you will want a certificate signed by a universally recognized Certificate Authority.

To configure SSL communication with Content Server, you need to do these tasks:

1. Install and enable the SecurityProviders component. The SecurityProviders component must be installed and enabled in the Content Server instance that you want to access with SSL communication.

   This component is installed and enabled by default in Oracle WebCenter Content Server 11*g*R1.

2. Configure an incoming provider for SSL communication.

   For more information about configuring SSL, see *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

3. Create self-signed key pairs and certificates.

The code in Example 29–6 uses the IDC protocol over a Secure Socket (SSL).

Example 29–6 IDC Protocol over SSL

```
// build a secure IDC client as cast to specific type
IntradocClient idcClient = (IntradocClient)
   manager.createClient("idcs://localhost:4443");

// set the SSL socket options
config.setKeystoreFile("ketstore/client_keystore");  //location of keystore file
config.setKeystorePassword ("password");      // keystore password
config.setKeystoreAlias("SecureClient");  //keystore alias
config.setKeystoreAliasPassword("password");  //password for keystore alias
```

29.3.3 Configuring JAX-WS

To make a JAX-WS connection, the RIDC client and Oracle WebCenter Content Server must be configured with compatible client and service web service policies, respectively.

For the RIDC client, you can either set an explicit client policy (LPA mode) using `jaxwsConfig.setClientSecurityPolicy(...)` or inherit a GPA client policy, provided the application consuming RIDC is deployed to an Oracle WebLogic Server domain with a GPA policy for `ws-client` correctly configured and targeted.

A service policy can be directly attached to the Oracle WebCenter Content web services (`IdcWebLoginPort`) end-point (LPA mode), or a GPA ws-service policy can be configured for the domain and inherited by the service.

29.3.3.1 Setting LPA Mode for a Service

You can set LPA mode for a service with Oracle Enterprise Manager Fusion Middleware Control.

To set LPA mode for a service with Fusion Middleware Control:

1. Log in to Oracle Enterprise Manager 11g Fusion Middleware Control.
2. In the navigation tree on the left, expand **Application Deployments**, and click **Oracle UCM Native Web Services**.
3. From the **Application Deployment** drop-down menu on the Oracle UCM Native Web Service page, choose **Web Services**.
4. Under **Web Service Details** on the Web Services (Oracle Infrastructure Web Services) page, click the **Web Service Endpoints** tab.
5. Click **IdcWebLoginPort** in the **Endpoint Name** column.
6. On the IdcWebLoginPort (Web Service Endpoint) page, click the **OWSM Policies** tab.
7. Under **Directly Attached Policies**, click **Attach/Detach**, and choose an appropriate available policy; for example, `oracle/wss_saml_or_username_token_service_policy`.

29.3.3.2 Setting a GPA Service Policy for a Domain

You can configure inheritance of a GPA policy with WebLogic Scripting Tool (WLST) commands.

To set a GPA ws-service policy for a domain with WLST:

1. Initialize the WebLogic Scripting Tool (WLST), using the WebLogic Server Administration Scripting Shell:

```
(/u01/app/oracle/product/Middleware/oracle_common/common/bin)% ./wlst.sh

...

Initializing WebLogic Scripting Tool (WLST) ...

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands
```

2. Invoke a sequence of commands similar to the following ones, which are for a domain named `base_domain`:

```
$MW_HOME/Oracle_ECM1/common/bin/wlst.sh
connect(username='weblogic',password='welcome1',url='t3://localhost:7001')
beginRepositorySession()
createPolicySet('base_domain-ws-service','ws-service','Domain("base_domain")')
attachPolicySetPolicy('oracle/wss_saml_or_username_token_service_policy')
validatePolicySet()
commitRepositorySession()
listPolicySets()
exit()
```

3. Verify that the GPA service policy has been set:

   a. Wait a few minutes for the GPA service policy to be picked up by `IdcWebLoginPort`.

   b. Inspect the WSDL and look for `wsp:PolicyReference` to see if changes have been applied:

      `http://server:16200/idcnativews/IdcWebLoginPort?WSDL`

For more information about setting a GPA web service client policy, see Example 29–7 and Example 29–8.

29.3.3.3 Setting a GPA Client Policy for a Domain

To determine GPA policy for a ws-client that will be leveraged by RIDC over JAX-WS should no explicit LPA be set, initialize the WebLogic Scripting Tool (WLST) and use the WebLogic Server Administration Scripting Shell.

The code in Example 29–7 provides an example.

Example 29–7 Determining GPA Policy with the WebLogic Scripting Tool

```
(/u01/app/oracle/product/Middleware/oracle_common/common/bin)% ./wlst.sh

...

Initializing WebLogic Scripting Tool (WLST) ...

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands

wls:/offline> connect('weblogic','welcome1','t3://localhost:7001')
Connecting to t3://localhost:7001 with userid weblogic ...
Successfully connected to Admin Server 'AdminServer' that belongs to domain 'base_
domain'.

wls:/base_domain/serverConfig> help('wsmManage')

Operations that provide support to manage the global policy attachments and
Oracle MDS repository.

  help('abortRepositorySession')
    Abort the current repository session,
    discarding the changes made to repository.
  help('attachPolicySet')
    Attach a policy set to the specified resource scope.
  help('attachPolicySetPolicy')
    Attach a policy to a policy set using the policy's URI.
  help('beginRepositorySession')
    Begin a session to modify the repository.
  help('clonePolicySet')
    Clone a new policy set from an existing policy set.
  help('commitRepositorySession')
    Write the contents of the current session to the repository.
  help('createPolicySet')
    Create a new, empty policy set.
  help('deletePolicySet')
    Delete a specified policy set.
```

```
  help('describeRepositorySession')
    Describe the contents of the current repository session.
  help('detachPolicySetPolicy')
    Detach a policy from a policy set using the policy's URI.
  help('displayPolicySet')
    Display the configuration of a specified policy set.
  help('enablePolicySet')
    Enable or disable a policy set.
  help('enablePolicySetPolicy')
    Enable or disable a policy attachment
    for a policy set using the policy's URI.
  help('exportRepository')
    Export a set of documents from the repository into a supported ZIP archive.
  help('importRepository')
    Import a set of documents from a supported ZIP archive into the repository.
  help('listPolicySets')
    Lists the policy sets in the repository.
  help('migrateAttachments')
    Migrates direct policy attachments to global policy attachments
    if they are identical.
  help('modifyPolicySet')
    Specify an existing policy set for modification in the current session.
  help('resetWSMPolicyRepository')
    Clean the Oracle MDS repository and re-seed with the current set
    of WSM policies.
  help('setPolicySetDescription')
    Specify a description for the policy set selected within a session.
  help('upgradeWSMPolicyRepository')
    Add newly introduced WSM policies to the Oracle MDS repository.
  help('validatePolicySet')
    Validate an existing policy set in the repository or in a session.

wls:/base_domain/serverConfig> listPolicySets()
Location changed to domainRuntime tree. This is a read-only tree with DomainMBean
as the root.
For more help, use help(domainRuntime)

Global Policy Sets in Repository:
  base-domain-ws-client

wls:/base_domain/serverConfig> displayPolicySet('base-domain-ws-client')

Policy Set Details:
-------------------
Name: base-domain-ws-client
Type of Resources: Web Service Client
Scope of Resources: Domain("base_domain")
Description: Global policy attachments for Web Service Client resources.
Enabled: true
Policy Reference: security : oracle/wss10_saml_token_client_policy, enabled=true
```

The code in Example 29–8 sets the ws-client GPA policy.

Example 29–8 Add GPA for the Web Service Client

```
# add GPA for the web service client assuming domain name is base_domain
beginRepositorySession()
createPolicySet('base_domain-ws-client','ws-client','Domain("base_domain")')
```

```
# assuming service policy is hardcoded to
# oracle/wss11_saml_token_with_message_protection_service_policy
# and that we want the RIDC client to leverage client policy:
# oracle/wss11_saml_token_with_message_protection_client_policy
attachPolicySetPolicy
  ('oracle/wss11_saml_token_with_message_protection_client_policy')
validatePolicySet()
commitRepositorySession()

# confirm policy set created
listPolicySets()

# add GPA for the web service client assuming domain name is base_domain
beginRepositorySession()
createPolicySet('base_domain-ws-client','ws-client','Domain("base_domain")')

# assuming service policy is hardcoded to
# oracle/wss11_saml_token_with_message_protection_service_policy
# and that we want the RIDC client to leverage client policy:
# oracle/wss11_saml_token_with_message_protection_client_policy
attachPolicySetPolicy
  ('oracle/wss11_saml_token_with_message_protection_client_policy')
validatePolicySet()
commitRepositorySession()

# confirm policy set created
listPolicySets()
```

29.3.4 Changing Default Settings

There are several JAX-WS specific configurations that can be done after you have created the client. However, in most cases, you should use the default settings.

This code builds a client as a cast for a JAX-WS type:

```
JaxWSClient jaxwsClient = (JaxWSClient) manager.createClient
      ("http://wlsserver:7044/idcnativews");
JaxWSClientConfig jaxwsConfig = jaxwsClient.getConfig();
```

You can set the instance name of the Content Server that you would like to connect to. This is set to "/cs/" by default which is the default webcontext for UCM installation. If the server webcontext is different than the default, then you may set it as:

```
// set the property
jaxwsConfig.setServerInstanceName("/mywebcontext/");
```

Setting the JPS configuration file location. A JPS configuration file is required for most policies such SAML and/or Message Token.

```
jaxwsConfig.setJpsConfigFile("/my/path/to/the/jps-config.xml");
```

Setting the security policy:

```
jaxwsConfig.setClientSecurityPolicy("policy:oracle/wss11_username_token_with_
message_protection_client_policy");
```

Changing the Login Port WSDL URL

RIDC uses the default values for the installed web services. If, for some reason, the web services have been modified and do not conform to the default URI/URLs, you may need to modify the default values.

Changing the login port WSDL URL:

```
jaxwsConfig.setLoginServiceWSDLUrl
  (new URL("http://server:7044/webservices/loginPort?WSDL"));
```

Change the request service URL:

```
jaxwsConfig.setRequestServiceWSDLUrl
  (new URL("http://server:7044/anotherservice/myrequestport?WSDL"));
```

The default streaming chunk size is 8192. This example changes the chunk size:

```
jaxwsConfig.setStreamingChunkSize(8190);
```

29.4 Authenticating Users

All calls to Remote Intradoc Client (RIDC) require some user identity for authentication. Optionally, this identity credential can be accompanied by other parameters such as a password as required by the protocol. The user identity is held in the `IdcContext` object; once created, it can be reused for all subsequent calls. To create a context, you pass in the user name and, optionally, some credentials.

Create a simple context with no password (for idc:// URLs):

```
IdcContext userContext = new IdcContext("weblogic");
```

Create a context with a password:

```
IdcContext userPasswordContext = new IdcContext("weblogic", "welcome1");
```

For Intradoc URLs, no password is required in the credentials because the request is trusted between Content Server and the client.

For JAX-WS URLs, the requirement for credentials will be dependent on the service policy that the web service is configured to use by the server.

29.5 Using Services

To invoke a service use the `IdcClient` class method:

```
public ServiceResponse sendRequest (IdcContext userContext, DataBinder dataBinder)
throws IdcClientException
```

For more information, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*.

The code in Example 29–9 executes a service request and gets back a data binder of the results.

Example 29–9 Executing a Service Request

```
// get the binder
DataBinder binder = idcClient.createBinder();

// populate the binder with the parameters
binder.putLocal ("IdcService", "GET_SEARCH_RESULTS");
binder.putLocal ("QueryText", "");
binder.putLocal ("ResultCount", "20");

// execute the request
ServiceResponse response = idcClient.sendRequest (userContext, binder);
```

The `ServiceResponse` contains the response from Content Server. From the response, you can access the stream from the Content Server directly, or you can parse it into a `DataBinder` and query the results.

The code in Example 29–10 takes the ServiceResponse and gets the search results, printing out the title and author value.

Example 29–10 Get the Binder and Loop Over the Results

```
// get the binder
DataBinder binder = response.getResponseAsBinder ();
DataResultSet resultSet = binder.getResultSet ("SearchResults");

// loop over the results
for (DataObject dataObject : resultSet.getRows ()) {
    System.out.println ("Title is: " + dataObject.get ("dDocTitle"));
    System.out.println ("Author is: " + dataObject.get ("dDocAuthor"));
}
```

If you consume a stream, your code is responsible for closing the stream. The code in Example 29–11 closes a stream.

Example 29–11 Closing a Stream

```
IdcContext user = new IdcContext ("weblogic", "welcome1");
IdcClientManager manager = new IdcClientManager ();
IdcClient idcClient = manager.getClient ("some url");
DataBinder binder = idcClient.createBinder ();
binder.putLocal ("IdcService", "GET_FILE");
binder.putLocal ("dID", "12345");
ServiceResponse response = idcClient.sendRequest (user, binder);

InputStream stream = null;
try {
  stream = response.getResponseStream ();
  int read = 0;
  int total = 0;
  byte[] buf = new byte[256];
  while ((read = stream.read (buf)) != -1) {
  total += read;
  }
} finally {
  if (stream != null) {
  stream.close ();
  }
}
```

For information about connection pooling and closing through the stream, see Section 29.6, "Handling Connection Pooling"

29.6 Handling Connection Pooling

The `IdcClientConfig#getConnectionPool` property determines how RIDC will handle pooling of connections. There are two options, `simple` and `pool`.

- The `simple` option is the default. The `simple` option does not enforce a connection maximum and rather lets every connection proceed without blocking and does not enforce a connection maximum. In most cases this option should be used.
- The `pool` option specifies the use of an internal pool that allows a configurable number of active connections at a time (configurable through the `IdcClientConfig#getConnectionSize` property), with the default active size set to `20`.

Usually, when the RIDC library is used to communicate from an application that itself is in an application container (such as a web application), the inbound requests have already been throttled. Thus, the `simple` option is the correct choice to use. The only scenario to use the `pool` option is if you are creating a standalone server and you are manufacturing a large number of concurrent calls to Content Server, which may cause Content Server to become overwhelmed.

A different pool implementation can be registered through the `IdcClientManager#getConnectionPoolManager()#registerPool()` method, which maps a name to an implementation of the `ConnectionPool` interface. The name can then be used in the `IdcClientConfig` object to select that pool for a particular client.

29.7 Sending and Receiving Streams

Streams are sent to the Content Server through the `TransferFile` class. This class wraps the actual stream with metadata about the stream (length, name, and so on). For information about methods that allow check-ins of files and streams, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*.

The code in Example 29–12 performs a check-in to the Content Server:

Example 29–12 Content Server Check-In

```
// create request
DataBinder binder = idcClient.createBinder();
binder.putLocal ("IdcService", "CHECKIN_UNIVERSAL");

// get the binder
binder.putLocal ("dDocTitle", "Test File");
binder.putLocal ("dDocName", "test-checkin-6");
binder.putLocal ("dDocType", "ADACCT");
binder.putLocal ("dSecurityGroup", "Public");

// add a file
binder.addFile ("primaryFile", new TransferFile ("test.doc"));

// check in the file
idcClient.sendRequest (userContext, binder);
```

Response from Content Server

Streams are received from the Content Server through the `ServiceResponse` object. For a summary of available methods, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*.

The response is not converted into a DataBinder unless specifically requested. If you just want the raw HDA data, you can get that directly, along with converting the response to a String or DataBinder.

The code in Example 29–13 executes a service, gets the response as a string, and parses it into a data binder.

Example 29–13 Parsing a String into a DataBinder

```
// create request
DataBinder binder = idcClient.createBinder ();

// execute the service
ServiceResponse response = idcClient.sendRequest (userContext, binder);

// get the response stream
InputStream stream = response.getResponseStream ();

// get the response as a string
String responseString = response.getResponseAsString ();

// parse into data binder
DataBinder dataBinder = response.getResponseAsBinder ();
```

Most Content Server service requests return a structured HDA payload that is modeled on the client using a DataBinder. The HDA payload is essentially a map-like structure optionally containing some ResultSets, which resemble tables.

Download-style service requests (such as `GET_FILE`) generally are expected to return the requested document's contents as a raw stream of bytes. However, if the parameters supplied to a `GET_FILE` request are invalid, or if the end user does not have sufficient privileges, and so on, Content Server can respond with an HDA payload containing the error information. Therefore, when performing a request such as `GET_FILE`, you should interrogate `ServiceResponse` to determine the response type returned, as Example 29–14 illustrates.

Example 29–14 Response Type Returned

```
DataBinder binder = idcClient.createBinder ();
binder.putLocal ("IdcService", "GET_FILE");
binder.putLocal ("dID", "12345");
ServiceResponse response = idcClient.sendRequest (user, binder);
if (response.getResponseType().equals(ServiceResponse.ResponseType.BINDER))
{
DataBinder responseBinder = response.getResponseAsBinder(false); // do not check
for errors
int statusCode = m_binder.getLocalData ("StatusCode").getInteger("StatusCode");
String statusMessage = m_binder.getLocal ("StatusMessage");
throw new IllegalStateException("Download response was not a stream - Error: " +
statusCode + " - " + statusMessage);
}
```

29.8 Reusing Binders for Multiple Requests

Binders can be reused among multiple requests. A binder from one request can be sent in to another request. Note that if you reuse a binder from one call to the next you need to be very careful there is nothing leftover in the binder that could impact your next call. RIDC does not clean the binder after each call.

The code in Example 29–15 provides an example that pages the search results by reusing the same binder for multiple calls to Content Server.

Example 29–15 Reusing Binders

```
// create the user context
IdcContext idcContext = new IdcContext ("sysadmin", "idc");

// build the search request binder
DataBinder binder = idcClient.createBinder();
binder.putLocal("IdcService", "GET_SEARCH_RESULTS");
binder.putLocal("QueryText", "");
binder.putLocal("ResultCount", "20");

// send the initial request
ServiceResponse response = idcClient.sendRequest (binder, idcContext);
DataBinder responseBinder = response.getResponseAsBinder();

// get the next page
binder.putLocal("StartRow", "21");
response = idcConnection.executeRequest(idcContext, binder);
responseBinder = response.getResponseAsBinder();

// get the next page
binder.putLocal("StartRow", "41");
response = idcConnection.executeRequest(binder, idcContext);
responseBinder = response.getResponseAsBinder();
```

29.9 Setting User Security

The Content Server has several security models that are controlled by settings on the Content Server. To resolve if a particular user has access to a document, three things are needed: The user's permission controls, the document's permission controls, and Content Server security environment settings.

It is assumed that the Application Program calling the UserSecurity module will fetch documents and the DOC_INFO metadata (in the document's binder, typically the result of a Search) as some superuser and cache this information. When the Application needs to know if a particular user has access to the document, a call is made to the Content Server as that user to fetch that user's permissions. Once the user's permission controls are known, then they can matched to the information in the document's metadata to resolve the access level for that user. (Access level is READ or READ/WRITE or READ/WRITE/DELETE). The need therefore is to reduce the number of calls to the Content Server (with a cache) and to provide a default implementation for matching the user's permissions information with the document's permission information. One further complication is that the Content Server controls which types of security are used in some server environment properties: `UseAccounts=true` and `UseCollaboration=true` or `UseEntitySecurity=1`. Additionally, a method allows testing to see if admin rights are assigned to a security type for that document.

The user security convenience is accessed through the `IUserSecurityCache` interface. There classes implement the optional Content Server security:

- The `UserSGAcctAclCache` class should always be called. This class will check the Content Server for security configuration and internally adjust itself to match.
- The `UserSecurityGroupsCache` class keeps a cache of user permissions and will match documents considering only Security Group information. Do not call this class directly. The `UserSGAcctAclCache` class will check the Content Server for security configuration and internally adjust itself to match.
- The `UserSGAccountsCache` class adds a resolver to also consider Account information if the Content Server has the UseAccounts=true setting. Do not call this class directly. The `UserSGAcctAclCache` class will check the Content Server for security configuration and internally adjust itself to match.

For more information, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*.

The code in Example 29–16 provides an example of setting user security.

Example 29–16 Setting User Security

```
IdcClientManager m_clientManager = new IdcClientManager ();
IdcClient m_client = m_clientManager.createClient
        ("http://localhost/scs/idcplg");

//RIDC superuser context
IdcContext m_superuser = new IdcContext("sysadmin", "idc");

//This class will self-adjust (downwards) to match the security model
// on Content Server.
IUserSecurityCache m_userSecurityCache = new UserSGAcctAclCache
   (m_client, 20, 1000, 20000, m_superuser);
ITrace trace = null;

//Example test
testDocPermission () {
  //If you don't want to do any logging, you can leave trace as null
  if (m_log.isLogEnabled(ILog.Level.TRACE)) {
  trace = new Trace ();
  }
  DataBinder m_doc1 = getDataBinder ("TEST");
  //Get the document information (typically in the first row of DOC_INFO)
  DataObject docInfo = m_doc1.getResultSet ("DOC_INFO").getRows ().get (0);
  //Get the cache id for this user
    //This makes a live call to content server to get the user ID for "Acme1"
    //CacheId acme1 = m_userSecurityCache.getCacheIdForUser
    //  (new IdcContext("Acme1", "idc"), trace);
  IdcContext context = new IdcContext("Acme1", "idc");
  CacheId acme1 = new CacheId (context.getUser (), context);
  //Get the access level for this document by this user
  int access = m_userSecurityCache.getAccessLevelForDocument
        (acme1, docInfo, trace);
  //Check if user has ACL admin permissions
  boolean aclAdmin = m_userSecurityCache.isAdmin
         (acme1, docInfo, IUserSecurityCache.AdminType.ACL, trace);
    if (m_log.isLogEnabled(ILog.Level.TRACE)) {
        m_log.log (trace.formatTrace (), ILog.Level.TRACE);
        }
}
```

```
//Example code to get a Document's DOC_INFO databinder
DataBinder getDataBinder (String dDocName) throws IdcClientException {
  DataBinder dataBinder = m_client.createBinder ();
  dataBinder.putLocal ("IdcService", "DOC_INFO_BY_NAME");
  dataBinder.putLocal ("dDocName", dDocName);
  ServiceResponse response = m_client.sendRequest (m_superuser, dataBinder);
  return response.getResponseAsBinder ();
}
//Example code to create a DataObject
DataObject dataObject = m_client.getDataFactory ().createDataObject ();
dataObject.put ("dSecurityGroup", "public");
dataObject.put ("dDocAccount", "Eng/Acme");
```

Internally, these fields from the document are examined during getAccessLevelForDocument():

- For the `AccessResolverSecurityGroups` class: `dSecurityGroup`.
- For the `AccessResolverAccounts` class: `dDocAccount`.
- For the `AccessResolverSecurityGroups` class: `xClbraUserList`, `xClbraAliasList`, and `xClbraRoleList`.

The `IAccessResolver` classes determine if they should participate based on cached information from the Content Server, if they do participate, the access levels are ANDed together. You can use the `hasAdmin()` method to determine if there is admin access. For more information, see the *Oracle Fusion Middleware Java API Reference for Oracle WebCenter Content Remote Intradoc Client (RIDC)*.

29.10 Using RIDC Filters

Remote Intradoc Client (RIDC) 11*g*R1 (11.1.1.4.0 or later), enables your application code to add a filter before the DataBinder is processed and sent to Content Server. You can create a filter by extending one of the `IdcFilterAdapter` classes, and then register that filter to execute with the `IdcFilterManager` class. Filters are executed in the order specified when registered. You can also get and remove previously registered filters.

The code in Example 29–17 extends an adapter and overrides a method to perform an action.

Example 29–17 Calling RIDC Filter Before Service Request

```
public class IdcFilterAddComment extends BeforeServiceRequestFilter {
  @Override
  public void beforeServiceRequest
       (IdcClient client, IdcContext context, DataBinder binder)
        throws IdcClientException {
    String existingComments = binder.getLocal("xComments");
    if (existingComments != null) {
        binder.putLocal("xComments", String.format
        ("%s %s", existingComments, "--DGT WAS HERE--"));
     } else {
         binder.putLocal("xComments", "--DGT WAS HERE--");
     }
  }
}
```

Remote Intradoc Client (RIDC) 11gR1 (11.1.1.5.0 or later) provides two more filter locations in the JAX-WS processing area. To use these filters, extend the `BeforeJaxwsServiceFilter` class.

The code in Example 29–18 extends the `BeforeJaxwsServiceFilter` class.

Example 29–18 Calling RIDC Filter Before JAX-WS Call

```
/**
 * RIDC filter called just before jaxws call is made to
 * loginPort.contentServerLogin () in authenticateUser ()
 **/
public void beforeJaxwsAuthenticateUser (IdcContext context, DataBinder binder,
        Map<String, Object> requestContext) throws IdcClientException {
    requestContext.put(oracle.wsm.security.util.SecurityConstants.
        ClientConstants.WSM_SUBJECT_PRECEDENCE, "false");
}

/**
 * RIDC filter called just before jaxws call is made to
 * loginPort.contentServerRequest () in performServiceRequest ()
 **/
public void beforeJaxwsServiceRequest (IdcContext context, DataBinder binder,
        Map<String, Object> requestContext) throws IdcClientException {
        //Override this class and implement your filter here
}
```

The code in Example 29–19 registers your filter class(es).

Example 29–19 Register Filer Classes

```
// If you are at the start of a pure RIDC application, you typically
// will create a ClientManager, for example:
IdcClientManager m_clientManager = new IdcClientManager();

// New method added to IdcClient to get the ClientManager
// if you do not have the ClientManager instance:
IdcClient client = myClient;
client.getClientManager();

// From the ClientManager, you can get the FilterManager:
IdcFilterManager fmanager = m_clientManager.getFilterManager();

// Then register your filter:
IIdcFilter addCommentFilter = new IdcFilterAddComment();
int slot = fmanager.registerFilter(100, addCommentFilter);

// Optionally, you can deregister. However, it might not be in the slot you
// assigned because there might have already been a filter in that slot.
// When registering, the next available higher slot will be used. You also need
// to pass in the instance currently in the slot you want to remove:
fmanager.deRegisterFilter(slot, addCommentFilter);

// Here is an example to remove all the filters,
// including the ones you did not register
for (Integer slot:fmanager.getUsedSlots()) {
  fmanager.deRegisterFilter(slot, fmanager.getFilter (slot));
}
```

30

Using the Content Server JCR Adapter

This chapter describes how to use the Java Content Repository (JCR) adapter for Oracle WebCenter Content Server.

This chapter includes the following sections:

30.1 About the Java Content Repository Adapter

The Java Content Repository API is a specification for accessing content repositories in a standardized manner. This specification was developed under the Java Community Process as JSR-170 and includes the Content Repository for Java API and the Java Content Repository (JCR).

The standard APIs associated with the JSR-170 specification are functional and exposed in the JCR adapter for Content Server. The JCR 1.0 API is required and must be predeployed and integrated as part of the underlying framework.

Oracle adapters are fully standards based and compliant with both the J2EE Connector Architecture and the Web Services Architecture. The JCR adapter can be deployed on any JSR-170-compliant application to enable communication with Content Server through the standards-based JCR specification.

30.1.1 JCR Data Model

The JCR standard uses a hierarchical data model based on extensible node types and content properties. This data model is used by the repository's underlying storage subsystems. For more information, see the JCR and JSR-170 standards.

- The **nt:folder** node type represents a structured collection of nodes. It is closely related to the directory or folder concept found in many file systems and is the node type that is normally used when mapping file system directories to a content repository.

- The **nt:resource** child node is normally used instead of a plain binary property when more resource metadata is required.
- The **nt:file** node type represents a file with some content.
- The **nt:unstructured** node type permits all kinds of properties and child nodes to be added to a node. It is normally used when nothing is known about the content that will be stored within a node.

30.1.2 JCR Adapter Data Model for Content Server

This is the data model for the Content Server JCR adapter:

```
A Folder [nt:folder]
+- jcr:content [nt:resource]
   +- jcr:created DATE
      <returns dCreateDate for the folder>
   +- ojcr:owner STRING
      <returns dCollectionOwner for the folder>
   +- ojcr:creator STRING
      <returns dCollectionCreator if it is available,
      otherwise it returns dCollectionOwner>
   +- ojcr:lastModifier STRING
      <returns dCollectionModifier if it is available,
       otherwise it returns dCollectionOwner
   +- ojcr:lastModified STRING
      <returns dLastModifiedDate>
   +- ojcr:displayName STRING
      <returns dCollectionName for the folder>
   +- idc:defaultMetadata [nt:unstructured]
      <metadata that should by default be applied to content checked
      into this folder. see idc:metadata under nt:file/jcr:content for
      example fields>
  +- idc:folderMetadata [nt:unstructured]
     +- idc:dCollectionName STRING
     +- idc:dCreateDate DATE
     +- idc:dCollectionPath STRING
     +- idc:dLastModifiedDate DATE
     +- idc:dCollectionOwner STRING
     +- idc:dCollectionGUID STRING
     +- idc:dParentCollectionID INTEGER
     +- idc:dCollectionQueries INTEGER
     +- idc:dCollectionEnabled INTEGER
     +- idc:dCollectionInherit INTEGER
     +- idc:dChildManipulation INTEGER
     +- idc:dCollectionID INTEGER
     +- idc:dCollectionCreator STRING
     +- idc:dCollectionModifier STRING
  +- idc:folderPermissions [nt:unstructured]
     +- idc:userCanRead INTEGER
     +- idc:userCanWrite INTEGER
     +- idc:userCanDelete INTEGER

A Document.txt [nt:file]
+- jcr:content [nt:resource]
   +- jcr:data=...
   +- jcr:created DATE
      <returns dDocCreatedDate from the RevClasses table>
   +- ojcr:creator STRING
      <returns dDocCreator from the RevClasses table>
   +- ojcr:lastModifier STRING
```

```
   <returns dDocLastModifier from the RevClasses table>
+- ojcr:lastModified STRING
   <returns dDocLastModifiedDate >
+- ojcr:author STRING
   <returns dDocAuthor for the document>
+- ojcr:comment STRING
   <if xComments exists as a metadata field, that is returned>
+- ojcr:displayName STRING
   <returns the filename>
+- ojcr:language STRING
   <if xIdcLanguage exists as a metadata field, that is returned>
+- idc:metadata [nt:unstructured]
   <returns values for everything in the RevClasses table,
   please see the definition of that table to see exactly what is defined
   +- idc:dID INTEGER
   +- idc:dDocName STRING
   +- idc:dDocTitle STRING
   +- idc:dDocAuthor STRING
   +- idc:dRevClassID INTEGER
   +- idc:dRevisionID INTEGER
   +- idc:dRevLabel STRING
   +- idc:dIsCheckedOut INTEGER
   +- idc:dSecurityGroup STRING
   +- idc:dCreateDate DATE
   +- idc:dInDate DATE
   +- idc:dOutDate DATE
   +- idc:dStatus STRING
   +- idc:dReleaseState STRING
   +- idc:dWebExtension STRING
   +- idc:dProcessingState STRING
   +- idc:dMessage STRING
   +- idc:dDocAccount STRING
   +- idc:dReleaseDate DATE
   +- idc:dRendition1 STRING
   +- idc:dRendition2 STRING
   +- idc:dIndexerState STRING
   +- idc:dPublishType STRING
   +- idc:dPublishState STRING
   +- idc:dWorkflowState STRING
   +- idc:dRevRank INTEGER
      <all custom metadata properties for a revision
       like idc:xComments STRING>
```

30.2 Installing Required APIs and Runtime Libraries

The JCR adapter can be used with any application that supports the JSR-170 specification, but the adapter requires a custom integration. This custom integration requires that an underlying framework consisting of several APIs and runtime libraries be installed.

> **Note:** All of these APIs and runtime libraries are provided with Oracle JDeveloper and WebCenter, with the exception of the JCR adapter and Remote Intradoc Client (RIDC).

30.2.1 Installing ADF Runtime Libraries

Several of the Application Development Framework (ADF) runtime libraries are required and must be installed on your application. These files are available in your Oracle JDeveloper instance. You can perform the installation using the ADF Runtime Installer wizard in JDeveloper, or you can do it manually.

The following ADF runtime libraries must be deployed on your application:

- `adf-share-base.jar`
- `adf-share-ca.jar`
- `adf-share-support.jar`
- `adflogginghandler.jar`

If you choose to manually install these libraries on your application, they must be installed in the `lib` directory. For example, an installation on Tomcat would use the *`TOMCAT_HOME`*`/common/lib` directory, and an installation on Oracle WebLogic Server would use the *`WL_HOME`*`/ADF/lib` directory. (For Oracle WebLogic Server, you must create the `ADF` and `lib` directories.)

30.2.2 Deploying Remote Intradoc Client (RIDC)

Remote Intradoc Client must be deployed on your application. RIDC provides a thin communication API for communication with Oracle WebCenter Content Server. This API removes data abstractions to the Oracle WebCenter Content Server instance while still providing a wrapper to handle connection pooling, security, and protocol specifics. RIDC is included with the JCR adapter distribution file and is available from the Oracle Technology Network (OTN).

For more information, see Chapter 29, "Using RIDC to Access Content Server."

30.2.3 Deploying the JCR API

The Java Content Repository (JCR) API must be deployed on your application. The JCR API is available from Oracle JDeveloper or for download from The Apache Software Foundation website at `http://www.apache.org/`

The JCR API is also part of the JSR-170 specifications download from the Java Community Process website at

`http://www.jcp.org/`

30.2.4 Installing the JCR Integration Libraries

The following JCR integration libraries are required and must be deployed on your application:

- `jcr-common-runtime.jar`
- `ojcr.jar`
- `ojdbc5.jar`

These files are available in your Oracle JDeveloper instance.

30.2.5 Installing the XML Integration Files

The following XML integration libraries are required and must be deployed on your application:

- `xmlparserv2.jar`
- `xquery.jar`

These files are available in your Oracle JDeveloper instance.

30.3 Deploying the JCR Adapter

The JCR adapter must be deployed on your application to enable communication with Content Server. The JCR adapter utilizes Remote Intradoc Client (RIDC) as part of the underlying framework and works in conjunction with the general JSR-170 architecture.

Follow the general instructions of your specific JSR-170-compliant application for deploying JCR adapters. The JCR adapter uses an embedded deployment descriptor (`rep_descriptor.xml`). Upon deployment, many applications will use the deployment descriptor to populate the configuration entries as part of an administration interface or deployment wizard. If your application does not use an administration interface or deployment wizard, you will need to edit the deployment descriptor directly and provide the required values.

30.4 Configuring Communication with Content Server

To enable communication between the JCR adapter and Content Server, you configure these items:

- Communication method
- Socket communication (listener port)
- Secure Socket Communication (SSL)
- Web communication (web servlet filter)
- User agent
- Cache settings

30.4.1 Supplying a Communication Method

You must supply the provider name and communication method with this configuration setting:

`CIS_SOCKET_TYPE_CONFIG`: This configuration setting defines the communication method with Content Server. The options are `socket`, `socketssl`, and `web`. For example:

```
oracle.stellent.jcr.configuration.cis.config.socket.type
```

- The `socket` (listener port) communication method specifies that RIDC should use the Content Server listener port. If `socket` is used as the communication method, you must provide the required configuration values.
- The `socketssl` communication method specifies that secure socket communication (SSL) be used as the communication protocol. If `socketssl` is used as the communication method, you must provide configuration values for both socket communication and secure socket communication.

- The `web` (web server filter) communication method specifies that RIDC should communicate through the web server filter, which requires individual authentication for each request. If `web` is used as the communication method, you must provide the required configuration value.

30.4.2 Configuring Socket Communication (Listener Port)

You must supply values for these configuration settings if secure socket communication (SSL) is used as the communication protocol:

- `SERVER_HOST_CONFIG`: The hostname of the machine on which Content Server is running. The default value is `localhost`.

  `oracle.stellent.jcr.configuration.server.host`

- `SERVER_PORT_CONFIG`: The port on which Content Server is listening. The default value is `16200`.

  `oracle.stellent.jcr.configuration.server.port`

30.4.3 Configuring Secure Socket Communication (SSL)

You must supply values for both socket communication (listener port) and these configuration settings if secure socket communication (SSL) is used as the communication protocol:

- `KEYSTORE_LOCATION`: The location and name of the keystore file.

  `oracle.stellent.jcr.configuration.ssl.keystore.location`

- `KEYSTORE_PASSWORD`: The password for the keystore file.

  `oracle.stellent.jcr.configuration.ssl.keystore.password`

- `PRIVATE_KEY_ALIAS`: The private key alias for authentication.

  `oracle.stellent.jcr.configuration.ssl.privatekey.alias`

- `PRIVATE_KEY_PASSWORD`: The private key password.

  `oracle.stellent.jcr.configuration.ssl.privatekey.password`

For information about socket communication values, see Section 30.4.2, "Configuring Socket Communication (Listener Port)."

30.4.4 Configuring Web Communication (Web Server Filter)

You need to supply a value for one of these configuration settings if your application is connecting through the web server filter (web communication):

- `SERVER_WEB_CONTEXT_ROOT_CONFIG`: The web server context root for Content Server, in the format `/context_root`. This setting provides a more seamless integration for Oracle WebCenter and for other application integrations.

  For example: /cs

- `SERVER_WEB_URL_CONFIG`: The full URL to the Content Server web server extension. Include the protocol (usually `http` or `https`), host name, port, relative web root, and extension root (usually `idcplg`). If a port other than port 80 is used, the port number needs to be specified.

For example:

```
http://myserver.example.com:8080/cs/idcplg/
oracle.stellent.jcr.configuration.server.web.url
```

30.4.5 Configuring the User Agent

You can optionally supply a value for this configuration setting to identify JCR requests:

- `CIS_USER_AGENT_CONFIG`: A string to append to the RIDC user agent. This value can be set to help identify requests made by the JCR adapter.

  ```
  oracle.stellent.jcr.configuration.cis.config.userAgent
  ```

30.4.6 Supplying Cache Settings

You can optionally supply values for these cache settings:

- `VCR_CACHE_INVALIDATION_INTERVAL`: Polling interval used by the WebCenter Content SPI to check for cache invalidations, in minutes. Defaults to `0` (zero), cache invalidation disabled. The minimum value is `2` minutes.

  ```
  com.oracle.content.spi.ucm.CacheInvalidationInterval
  ```

- `VCR_BINARY_CACHE_MAX_SIZE`: Maximum size of documents stored in the VCR binary cache, in bytes. The default value is `102400` (800 KB).

  ```
  com.bea.content.federated.binaryCacheMaxEntrySize
  ```

30.5 Finding Information About a Content Item

Content managed by Content Server is primarily tracked by four tables:

- Revisions
- Documents
- DocMeta
- RevClasses

These tables track the content's metadata, state, and actions as well as information that is associated with each file.

Revisions

This table tracks core information about each revision of the content:

- One row per revision
- Different revisions with the same content that share the same content ID and RevClass ID
- System metadata for each revision:
  - Metadata for revisions: content ID, title, author, check-in date, and so on
  - Metadata for categorization and security: type, security group, doc account

- State information for various actions:
  - Indexing
  - Workflow
  - Document conversion
- Numeric IDs and text labels to help track and retrieve a revision:
  - A unique `dID` value for each revision (the primary key in the table)
  - A unique `dRevClassID` value for the content
  - A revision ID to mark the revision number for each revision

Documents

This table tracks information for files that are associated with each content revision:

- One row per revision
- Multiple rows per revision, one row for each of these files:
  - Primary
  - Alternate
  - Web-viewable
- File information: original name, location, language, size, and so on

DocMeta

This table contains extended metadata fields:

- One row per revision
- One column per metadata field
- Definition for each field stored in the **DocMetaDefinition** table

RevClasses

This table tracks information for each content revision:

- One row per content item
- Row locked for content modification
- Unique **dDocName** and **RevClassId** values
- Current indexed revision
- Dates and users:
  - Creation date and creator
  - Last modified date and user
  - Owner

30.6 Using a Search Index

Content Server provides various ways to search the repository. Metadata searches can be based on the **Revisions**, **Documents**, **DocMeta**, and **RevClasses** tables. To efficiently perform text searches, the full-text search feature of Oracle Database can be utilized, and the **IdcText** table can be created to hold the search index.

IdcText

This table contains selected columns from the **Revisions**, **Documents**, **DocMeta**, and **RevClasses** tables as well as columns for other data:

- It contains a predefined list from the **Revisions**, **RevClasses**, and **Documents** tables.
- It contains custom metadata that is indicated as searchable from the **DocMeta** table.
- The **OtsMeta** column (CLOB field) contains an `SDATA` section and additional indexable fields that are not in the other columns. However, `SDATA` has significant limitations.
- The **OtsContent** column contains an indexable document.
- The **ResultSetInterface** column can be used for sorting or count estimation, or to drill down.

30.7 Using the File Store Provider

The File Store Provider can be used to distribute files managed by Content Server on the file system, a database, other devices, or any combination of these. The files are stored in SecureFiles in Oracle WebCenter Content Server 11*g*. For database-backed file storage, the **FileStorage** and **FileCache** tables store the information related to each file.

FileStorage

This table stores file information and some additional information:

- File stored in a BLOB (binary large object) field (SecureFiles in Oracle WebCenter Content Server 11*g*)

  The database administrator can turn on additional BLOB optimizations. For example, deduplication, compression, and encryption with SecureFiles.
- Values for `dID` and `dRenditionID` that point to a particular file managed by Content Server
- Tracking information in a small number of fields: last modified date and file size

FileCache

This table stores pointers for files cached on the file system, for certain types of processing (extraction, conversion, and so on), and for quick access by the web server. This pointer is also used to perform cleanup.

31

Configuring Web Services with WSDL, SOAP, and the WSDL Generator

This chapter describes how to integrate Oracle WebCenter Content into a client application with WSDL and SOAP files by using them to manage Oracle WebCenter Content Sever. It also describes how to use the WSDL Generator component, which provides integration technologies to access the functionality of Content Server.

This chapter includes the following sections:

- Section 31.1, "About Configuring Web Services with WSDL, SOAP, and the WSDL Generator"
- Section 31.2, "Accessing Content Server with a SOAP Client"
- Section 31.3, "Calling Content Server Services with SOAP"
- Section 31.4, "Using SOAP Packets in Active Server Pages"
- Section 31.5, "Generating WSDL Files to Access WebCenter Content"
- Section 31.6, "Customizing WSDL Files"

For general information about web services that you can use with Content Server, see Section 24.2, "Overview of Web Services." The way to use web services described in this chapter was introduced in Oracle Universal Content Management 10*g*. If you want to use WebCenter Content web services with security configuration and Security Assertion Markup Language (SAML) support, introduced in Oracle WebCenter Content 11*g*, see Chapter 25, "Configuring WebCenter Content Web Services for Integration."

With either way of using web services, you can use the Oracle Web Services Manager (Oracle WSM) for security. For more information about Oracle WSM, see the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

31.1 About Configuring Web Services with WSDL, SOAP, and the WSDL Generator

You can use Web Services Definition Language (WSDL) and SOAP (Simple Object Access Protocol) files to manage Content Server from a client application. SOAP is a lightweight, XML-based messaging protocol for encoding the information in web service request and response messages before sending them over a network.

The WSDL Generator component, `WsdlGenerator`, which is installed and enabled by default in Content Server, generates WSDLs for the services of Content Server. You can take the WSDLs and plug them into APIs to create web services for use with Content Server.

Some SOAP functionality has been built into the core Content Server. The WSDL Generator component is not essential for using SOAP. Administrators can still write service calls to Content Server in SOAP if needed. The WSDL Generator provides flexibility in altering existing client applications.

WebCenter Content has a WSDL 1.1 implementation that exposes the WebCenter Content IDCService (Internet Distributed Content Service), which in turn extends all of the capabilities of Content Server. With IDCService, you can do any of these tasks:

- Check in or check out content
- Create, run, or approve workflows
- Make content available for publishing
- Search content by category (metadata), content (full text), or a combination of both

You can use WSDL files to map to WebCenter Content and SOAP to access content and content management functions within WebCenter Content and to deploy your content management capabilities as a web service. Alternatively, you can write service calls to Content Server in SOAP.

31.1.1 Web Services Framework

The core enabling technologies for web services follow:

- XML Data
- WSDL Interface
- SOAP Communication
- UDDI Registry

31.1.1.1 XML Data

The eXtensible Markup Language (XML) is a bundle of specifications that provides the foundation of all web services technologies. Using the XML structure and syntax as the foundation allows for the exchange of data between different programming languages, middleware, and database management systems.

The XML syntax incorporates instance data, typing, structure, and semantic information associated with data. XML describes data independently and also provides information for mapping the data to software systems or programming languages. Because of this flexibility, any software program can be mapped to web services.

When web services are invoked, the underlying XML syntax provides the data encapsulation and transmission format for the exchanged data. The XML elements and attributes define the type and structure information for the data. XML provides the capability to model data and define the structure specific to the programming language (such as Java, C#, or Visual Basic), the database management system, or the software application. Web services use the XML syntax to specify how data is represented, how the data is transmitted, and how the service interacts with the referenced application.

31.1.1.2 WSDL Interface

The Web Services Description Language (WSDL) provides the interface that is exposed to web services. The WSDL layer enables web services to be mapped to underlying programs and software systems. A WSDL file is an XML file that describes how to connect to and use a web service.

31.1.1.3 SOAP Communication

The Simple Object Access Protocol (SOAP) provides Content Server communications for web services interfaces to communicate with each other over a network. SOAP is an XML-based communication protocol used to access web services. The web services receive requests and return responses using SOAP packets that are encapsulated within an XML document.

31.1.1.4 UDDI Registry

The Universal Description Discovery and Integration (UDDI) service provides registry and repository services for storing and retrieving web services interfaces. UDDI is a public or private XML-based directory for registering and looking up web services.

Content Server currently does not publish to any public or private UDDI sources. However, this does not prevent users from integrating Content Server with other applications using web services.

31.1.1.5 DIME Message Format

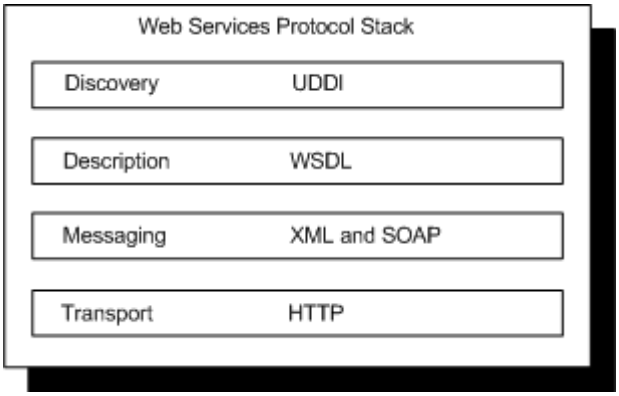
DIME is a lightweight, binary message format that can be used to encapsulate one or more application-defined groups of file content, of arbitrary type and size, into a single message construct. You can use this format for uploading or downloading content. The payloads consist of the SOAP message and one or more groups of file content.

31.1.1.6 How the Enabling Technologies Work Together

The XML, WSDL, SOAP, and UDDI technologies work together as layers on the web services protocol stack. As Figure 31–1 shows, the web services protocol stack consists of these layers:

- The service *transport* layer between applications (HTTP)
- The *messaging* layer, which provides a common communication method (XML and SOAP)
- The service *description* layer, which describes the public interface to a specific web service (WSDL)
- The service *discovery* layer, which provides registry and repository services for storing and retrieving web services interfaces (UDDI)

Figure 31–1 Layers of the Web Services Protocol Stack



Note: While several protocols are available for a transport layer (such as HTTP, SMTP, FTP, and BEEP), the HTTP protocol is most commonly used. The WSDL Generator component relies on the HTTP protocol as the transport layer.

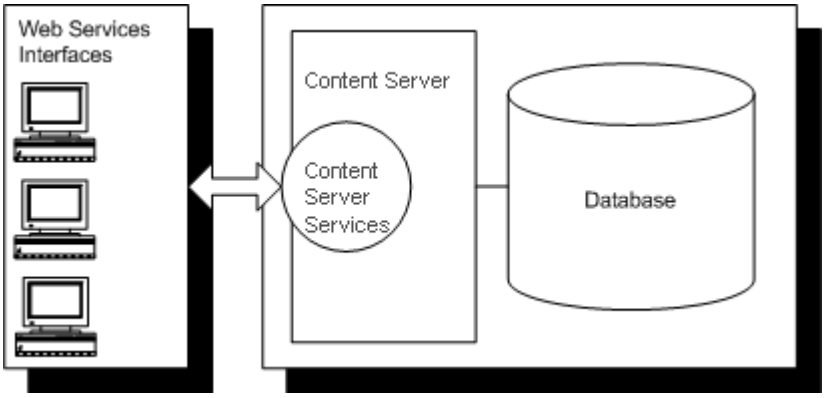
To help grasp the connection between these technologies, consider this simple analogy: Think of HTTP as the telephone wire (transport between applications) and UDDI as a telephone book (where a developer can browse a UDDI registry to locate a registered service). SOAP could be described as the voices of the people talking on the telephone (the exchange of information), and XML as the language they are speaking in (the underlying structure for the exchange of data). To continue with the telephone analogy, WSDL would be the phone number that calls a specific web service (of course, WSDL is more than just a phone number because it includes information such as the available functions and data types).

31.1.1.7 Implementation Architecture

Web services are not executable, but rather they exchange data within the development environment. So, web services are a means to exchange information with an application server or software package that is performing the communication between the programs exchanging data.

Figure 31–2 shows the web services implementation architecture for the Content Server application. The primary value of this architecture remains in the features and functions of Content Server. Web services access Content Server through the WSDL Generator and use the exposed Content Server services to execute actions and provide data transactions between the user employing web services and Content Server.

Figure 31–2 Web Services Implementation Architecture



31.1.1.8 Implementation on .NET

The Microsoft .NET products, including the .NET platform, .NET Framework, and Visual Studio .NET, all support the XML schema, WSDL, and SOAP specifications:

- The .NET platform is designed as a programming model that enables developers to build XML web services and applications. The platform provides a set of servers that integrates, executes, and manages XML web services and applications.
- The .NET Framework product enables developers to build and deploy web services and applications. It provides a structured environment for integrating web services, consists of a common language runtime and unified class libraries, and includes the ASP .NET server.

- The Visual Studio .NET product provides tools for developers to write application software according to the XML-based web service specifications.

Using the .NET architecture, development and deployment of a web service are integrated as a single step. Because every program written in a .NET language is designed to function as a web service, the .NET server is able to create and deploy the program as a web service.

31.1.1.9 The SOAP Protocol

SOAP is an XML-based messaging protocol consisting of these parts:

- An envelope that defines what is in a message and how to process it
- A set of encoding rules for defining application data types
- A convention for representing remote procedure calls and responses

Employing a SOAP integration provides a standardized interface for executing Content Server services using the Java API (`IdcCommand`) and provides XML and non-XML content managed by Content Server.

Because SOAP uses the Hypertext Transfer Protocol (HTTP) for data transmission, it can be invoked across the Web, and it can enable content to be accessible over a network in a platform-independent and language-neutral way.

31.2 Accessing Content Server with a SOAP Client

Using SOAP to access content management capabilities as a web service enables real-time programmatic interaction between applications, enables the integration of business processes, and facilitates information exchange.

Note: If you are developing SOAP client implementations, make sure that *chunking* is disabled in your client API code.

Web services are modular components that are contained in an XML wrapper and defined by the WSDL specifications. The UDDI) Web-based registry system is used to locate these services.

Tip: While .NET servers support WSDL and integrate with the SOAP Toolkit, you must specify that a SOAP packet is sending a Remote Procedure Call (RPC). The default is to evaluate SOAP messages as document-style SOAP messages, rather than RPC-style SOAP messages. Using the SOAP Toolkit client with a .NET-developed web service returns a read error for the WSDL document. To permit the SOAP Toolkit to read the generated WSDL and call your .NET web service, you must specify the `SoapRpcService()` attribute in your web service class.

31.2.1 Using a Java SOAP Client

With a Java SOAP client, you can use the command-line parameters that Table 31–1 describes.

Table 31–1 Command-Line Parameters for Java SOAP Clients

Parameter	Description
`-c` *config file*	The configuration file containing server settings (host, port, and so on)
`-x` *xml file*	The XML file containing the SOAP request to pass to Content Server
`-p` *primary file*	The file name of the primary file to upload
`-a` *alternate file*	The file name of the alternate file to upload (optional)
`-l` *log file*	The file name of the file containing the request and response data (optional)

31.3 Calling Content Server Services with SOAP

You can execute various Content Server `IdcCommand` services with the SOAP interface. Your user ID must have appropriate permissions to execute the commands. Some commands require administrative access, and other commands require only write permission.

The WSDL Generator component is installed and enabled by default with Content Server, and it must remain enabled to call services. For lists of available services and the required parameters, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

31.3.1 SOAP Packet Format

A SOAP request is an XML-based Remote Procedure Call (RPC) sent using the HTTP transport protocol. The payload of the SOAP packet is an XML document that specifies the call being made and the parameters being passed.

31.3.1.1 HTTP Headers

This entry is required in the HTTP header of a SOAP request:

```
Content-Type: text/xml; charset="utf-8"
```

This `SOAPAction` header is suggested, but not required:

```
SOAPAction: "http://www.oracle.com/IdcService"
```

31.3.1.2 Namespaces

Within the body of a SOAP message, XML namespaces are used to qualify element and attribute names in the parts of the document. Element names can be global (referenced throughout the SOAP message) or local. A local element name is provided by a namespace, and the name is used in the particular part of the message where it is located. So, SOAP messages use namespaces to qualify element names in the separate parts of a message. Application-specific namespaces qualify application-specific element names. Namespaces also identify the envelope version and encoding style.

Content Server defines a namespace called `idc` that explains the schema and allowable tags for the SOAP content.

31.3.1.3 Nodes

A SOAP node is the entity that processes a SOAP message according to the rules for accessing the services provided by the underlying protocols through the SOAP bindings. So, message processing involves mapping to the underlying services. The SOAP specification defines a correlation between the parts of a SOAP message and the software handlers that will process each part of the message.

The following nodes might be required for a service request or might be returned in the response:

- Service Node
- Document Node
- User Node
- Optionlist Node
- Option Subnode in an IDC Optionlist Node
- Resultset Subnode
- Row Subnode
- Field Subnode

> **Note:** In requests, Content Server services are lenient regarding where data is specified. If you specify a data field in a field node and it is supposed to be a document attribute, or vice versa, the service still processes the data correctly. The response puts the data in the correct node.

31.3.1.3.1 Service Node As the main node in the IDC namespace, the `<idc:service>` node has these requirements:

- This node must exist for a request to be processed.
- The required attribute `IdcService` defines the service you are requesting.
- The subnodes of `<idc:service>` are not required to carry the namespace in their tags.

  For example, you can use `<document>` rather than `<idc:document>`. However, if you do define the namespace identifier in the child nodes, it must match the identifier specified in the service tag.

Example 31–1 shows an `<idc:service>` node with a `PING_SERVER` service request.

Example 31–1 Service Node in the IDC Namespace

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="PING_
SERVER">
</idc:service>
```

31.3.1.3.2 Document Node The <document> node contains all content-item information and is the parent node of all data nodes.

Attributes that are valid for your content items are defined by your particular Content Server. For example, dID, dDocTitle, and dDocType are common attributes. These rules apply to the <document> node:

- Custom content-item information, such as xSpec, is valid if it is defined as metadata.
- All known document fields can be used as attributes.

Example 31–2 shows a <document> node that uses the CHECKOUT_BY_NAME service.

Example 31–2 Document Node in an IDC Service Node

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="CHECKOUT_
BY_NAME">
<idc:document dDocName="soap_sample">
</idc:document>
</idc:service>
```

31.3.1.3.3 User Node The <user> node contains all user information. These rules apply to the <user> node:

- Attributes that are valid for users are defined by a specific Content Server. For example, dName, dFullName, and dEmail are common attributes.
- Custom user information is valid if it is defined as metadata.
- All known user fields can be used as attributes.

Example 31–3 shows a <user> node that specifies a user for the GET_USER_INFO service request.

Example 31–3 User Node in an IDC Service Node

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_USER_
INFO">
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
```

31.3.1.3.4 Optionlist Node The <optionlist> node contains any option lists. The name attribute specifies the name of the option list. Each <option> subnode contains a value in the <optionlist> node.

Example 31–4 shows an <optionlist> node with an <option> subnode that has a locale value.

Example 31–4 Optionlist Node for Locale

```
<idc:optionlist name="Users_UserLocaleList">
<idc:option>
English-US
</idc:option>
</idc:optionlist>
```

31.3.1.3.5 Option Subnode in an IDC Optionlist Node The <option> subnode is specified within the <optionlist> node. The option attribute specifies the name of the option for the option list.

Example 31–5 shows <option> nodes with dDocType values.

Example 31–5 Option Subnodes of an Optionlist Node

```
<idc:optionlist name="dDocType">
<idc:option>ADACCT</idc:option>
<idc:option>ADHR</idc:option>
<idc:option>ADSALES</idc:option>
</idc:optionlist>
```

31.3.1.3.6 Resultset Subnode The <resultset> subnode can be specified within a <document> or <user> node. This subnode contains ResultSet information in a request or response. The name attribute specifies the name of the ResultSet.

Example 31–6 specifies a <resultset> subnode for a ResultSet that contains a revision history.

Example 31–6 Resultset Subnode for a Revision History

```
<idc:resultset name="REVISION_HISTORY">
<idc:row dFormat="text/plain" dInDate="4/12/02 1:27 PM" dOutDate=""
dStatus="RELEASED" dProcessingState="Y" dRevLabel="1" dID="6" dDocName="stellent"
dRevisionID="1">
</idc:row>
</idc:resultset>
```

31.3.1.3.7 Row Subnode The <row> subnode is specified within a <resultset> subnode, which can have multiple <row> subnodes. Each <row> subnode specifies a row in the ResultSet.

Attributes that are valid are defined by your specific Content Server. Valid attributes are the same fields that can appear as attributes in a <document> or <user> node.

Example 31–7 specifies a row in a ResultSet of user attributes.

Example 31–7 Row Subnode of a Resultset Subnode

```
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="jsmith" AttributeInfo="role,contributor,15">
</idc:row>
</idc:resultset>
```

31.3.1.3.8 Field Subnode The <field> subnode can be specified within a <document>, <user>, or <row> node. The name attribute specifies the name of the field. A <field> subnode often represents data, such as refreshSubjects or dSubscriptionID.

A <field> subnode can represent document or user metadata that a user can configure, or it can represent custom metadata, such as xComments. Example 31–8 specifies a field subnode that represents subscription ID data.

Example 31–8 Field Node to Represent Metadata

```
<idc:field name="dSubscriptionID">
stellent
</idc:field>
```

Another use for a <field> subnode is to pass search result values for fields such as QueryText and OriginalQueryText, as Example 31–9 shows.

Example 31–9 Field Subnode to Pass a Value

```
<idc:field name="QueryText">
dDocType <Substring> "ADSALES&"
</idc:field>
```

31.3.2 Special Characters

When passing special characters, such as a left angle bracket (<) or right angle bracket (>), to WebCenter Content, you must use the XML-encoding format, which Table 31–2 shows.

Table 31–2 Special Character Formats

Standard Format	XML-Encoding Format
<	<
>	>
"	" ` (use back quotation mark if you are using universal query syntax)
&	&
\	'

Note: Some search result values, such as the QueryText and OriginalQueryText values, are URL-encoded in the response.

You can pass a string to Content Server for a content-item query (using universal query syntax) in either format. Example specifies a string in standard format

Example 31–10 Parameter with a Standard-Format String

```
QueryText=dDocType <Substring> "ADSALES"
```

Example specifies a string in XML-encoded format.

Example 31–11 Parameter with an XML-Encoded String:

```
<idc:field name="QueryText">
dDocType &lt;Substring&gt; `ADSALES`
</idc:field>
```

31.3.3 Sample Service Calls with SOAP Response/Request

Using service calls with SOAP response/request, you can execute Content Server services in a SOAP request. For a list of available services and the required parameters, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*

These IdcCommand services are used as SOAP request examples.

IdcCommand	Description
PING_SERVER	This service evaluates whether a connection to the server exists. See Section 31.3.3.1, "Ping the Server,".
ADD_USER	This service adds a new user to the system. See "Add a New User" on page 31-12.
EDIT_USER	This service edits an existing user. See "Edit Existing User" on page 31-15.
GET_USER_INFO	This service retrieves the user list. See "Get User Information" on page 31-18.
DELETE_USER	This service deletes an existing user. See "Delete User" on page 31-20.
CHECKIN_UNIVERSAL	This service performs a Content Server controlled check-in. See "Check In Content Item" on page 31-21.
CHECKOUT_BY_NAME	This service marks the latest revision of the specified content item as locked. See "Check out Content Item" on page 31-25.
UNDO_CHECKOUT_BY_NAME	This service reverses a content item checkout using the Content ID. See "Undo Content Item Checkout" on page 31-27.
DOC_INFO	This service retrieves content item revision information. See "Get Content Item Information" on page 31-29.
GET_FILE	This service retrieves a copy of a content item without performing a check out. See "Get File" on page 31-30.
GET_SEARCH_RESULTS	This service retrieves the search results for the passed query text. See "Get Search Results" on page 31-33.
GET_TABLE	This service exports the specified table from the WebCenter Content database. See "Get Table Data" on page 31-36.
GET_CRITERIA_WORKFLOWS_FOR_GROUP	This service returns criteria workflow information. See "Get Criteria Workflow Information" on page 31-37.

31.3.3.1 Ping the Server

The PING_SERVER service evaluates whether a connection to the server exists.

- This service returns status information for Content Server.
- If this service is unable to execute, this message is displayed to the user: `Unable to establish connection to the server.`

> **Tip:** Execute a PING_SERVER request before calling other services to ensure that there is a connection to Content Server and that you are logged in as a user authorized to execute commands.

31.3.3.1.1 Required Parameters These parameters must be specified.

Parameter	Description
IdcService	Must be set to PING_SERVER.

31.3.3.1.2 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="PING_SERVER">
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

31.3.3.1.3 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="PING_SERVER">
<idc:document>
<idc:field name="changedSubjects">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="StatusMessage">
You are logged in as &#39;sysadmin&#39;.
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.2 Add a New User

The ADD_USER service adds a new user to the system.

- Given a user name, the service determines if the user is in the system. If the user does not exist, the service will add the user.
- The most likely error is when the user name is not unique. If this service is unable to execute, an error message is displayed to the user.

31.3.3.2.1 Required Parameters These parameters must be specified.

Parameter	Description
dName	The unique name.
dUserAuthType	The user authorization type. This value must be set to either `LOCAL` or `GLOBAL`.
IdcService	Must be set to `ADD_USER`.

31.3.3.2.2 Optional Parameters These optional parameters may be specified.

Parameter	Description
dEmail	The email address for the user.
dFullName	The full name of the user.
dPassword	The password for the user.

31.3.3.2.3 Optional Attribute Information This optional data defines the user's attribute information, the roles the user belongs to, and the accounts the user has access to. Attribute information consists of a list of three comma-delimited strings. The first string indicates the type of attribute, the second the name of the attribute, and the third is the access number.

> **Important:** The user attribute information is not predefined. The user by default will belong to no roles or accounts, and will become a guest in the system.

Attribute Information	Description
Access Number	The access number determines the level of access or privileges assigned to the user
Attribute Name	The attribute name is the name of the *role* or *account* to be assigned. For example, *admin*, *contributor*, or *editor* may be assigned.
Attribute Type	The attribute types consists of *role* or *account*.

Access Number

These access numbers can be assigned to the user.

Access Level Flags	Description
1	Read only.
3	Read and write.
7	Read, write, and delete.
15	Administrative privileges.

Attribute Name

A user can belong to multiple roles and accounts, there may be multiple role and account information strings separated by commas in the attribute information column.

- If the user is to have the admin role, define the user attribute information as follows:

```
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="jsmith" AttributeInfo="role,contributor,15">
```

- If the user is to belong to both the contributor and editor roles and has read privilege on the account books, define the user attribute information as follows:

```
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="jsmith"
AttributeInfo="role,contributor,15,role,editor,15,account,books,1">
```

Attribute Type

When defining a role, the first string specifies that this is a role attribute, the second string is the name of the role, and the third is the default entry of 15.

When defining an account, the first string specifies that this is an account attribute, the second string is the name of the account, and the third is the access level.

- For an attribute role, the information is in this form:

```
role,contributor,15
```

- For an attribute account where the access level determines the user's rights to the named account, the information is in this form:

```
account,books,1
```

31.3.3.2.4 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="ADD_USER">
<idc:user dName="Jennifer" dFullName="Jennifer Anton" dPassword="password"
dEmail="email@example.com" dUserAuthType="local">
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="Jennifer" AttributeInfo="role,contributor,3">
</idc:row>
</idc:resultset>
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.2.5 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="ADD_USER">
<idc:document>
<idc:field name="refreshMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshSubjects">
```

```
</idc:field>
<idc:field name="isAdd">
1
</idc:field>
<idc:field name="copyAll">
1
</idc:field>
<idc:field name="alwaysSave">
1
</idc:field>
<idc:field name="dAttributeName">
contributor
</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="doAdminFields">
1
</idc:field>
<idc:field name="dAttributePrivilege">
3
</idc:field>
<idc:field name="dAttributeType">
role
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="changedSubjects">
userlist,1018884022874
</idc:field>
</idc:document>
<idc:user dUserAuthType="local" dEmail="email@example.com" dFullName="Jennifer
Anton" dUser="sysadmin" dPassword="password" dName="Jennifer">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.3 Edit Existing User

The `EDIT_USER` service edits the information for an existing user.

- Given a user name and user authorization type, the service determines if the user is in the system. If the user does not exist, the service fails. Otherwise the user information is updated and replaced.
- The most likely error is the user does not have the security level to perform this action. If this service is unable to execute, an error message is displayed to the user.

> **Note:** The user attribute information replaces the current attributes. It does not add to the list. Consequently, if the user attribute information is not defined, the user will become a guest in the system.

31.3.3.3.1 Required Parameters These parameters must be specified.

Parameter	Description
dName	The unique name.
dUserAuthType	The user authorization type. This value must be set to either `LOCAL` or `GLOBAL`.
IdcService	Must be set to `EDIT_USER`.

31.3.3.3.2 Optional Parameters These optional parameters can be specified.

Parameter	Description
dEmail	The email address of the user.
dFullName	The full name of the user.
dPassword	The password for the user.
dUserLocale	The locale designation, such as English-US, English-UK, Deutsch, Français, Español.
dUserType	The defined user type.

31.3.3.3.3 Optional Attribute Information A ResultSet containing the user's attribute information and referencing the roles to which the user belongs and the accounts to which the user has access. Attribute information consists of a list of three comma-delimited strings. The first string indicates the type of attribute, the second the name of the attribute, and the third is the access number.

Important: The user attribute information is not predefined. The user by default will belong to no roles or accounts, and will become a guest in the system.

Attribute Information	Description
Access Number	The access number determines the level of access or privileges assigned to the user
Attribute Name	The attribute name is the name of the *role* or *account* to be assigned. For example, *admin*, *contributor*, or *editor* may be assigned.
Attribute Type	The attribute types consist of *role* or *account*.

Access Number

These access numbers can be assigned to the user.

Access Level Flags	Description
1	Read only.
3	Read and write.
7	Read, write, and delete.
15	Administrative privileges.

A user can belong to multiple roles and accounts, there may be multiple role and account information strings separated by commas in the attribute information column.

- If the user is to have the admin role, define the user attribute information as follows:

```
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="jsmith" AttributeInfo="role,contribut
or,15">
```

- If the user is to belong to both the contributor and editor roles and has read privilege for the account books, define the user attribute information as follows:

```
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="jsmith"
AttributeInfo="role,contributor,15,role,editor,15,account,books,1">
```

Attribute Type

In the definition of a role, the first string specifies that this is a role attribute, the second string is the name of the role, and the third is the default entry of 15.

In the definition of an account, the first string specifies that this is an account attribute, the second string is the name of the account, and the third is the access level.

- For an attribute role, the information is in this form:

```
role,contributor,15
```

- For an attribute account where the access level determines the user's rights to the named account, the information is in this form:

```
account,books,1
```

31.3.3.3.4 SOAP Request <?xml version='1.0' ?>

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="EDIT_USER">
<idc:user dName="Jennifer" dFullName="Jennifer Anton" dPassword="password"
dEmail="jennifer@example.com" dUserAuthType="local">
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="Jennifer" AttributeInfo="role,guest,1">
</idc:row>
</idc:resultset>
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.3.5 Response <?xml version='1.0' ?>

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="EDIT_USER">
<idc:document>
<idc:field name="refreshMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshSubjects">
```

```
</idc:field>
<idc:field name="alwaysSave">
1
</idc:field>
<idc:field name="dAttributeName">
guest
</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="doAdminFields">
1
</idc:field>
<idc:field name="dAttributePrivilege">
1
</idc:field>
<idc:field name="dAttributeType">
role
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="changedSubjects">
userlist,1018884022877
</idc:field>
</idc:document>
<idc:user dUserAuthType="local" dEmail="jennifer@example.com" dFullName="Jennifer
Anton" dUser="sysadmin" dPassword="password" dName="Jennifer">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.4 Get User Information

The `GET_USER_INFO` service retrieves the user list.

- Given a defined user, the service retrieves the user list.
- If this service is unable to execute, this message is displayed to the user: *Unable to retrieve user list.*

31.3.3.4.1 Required Parameters These parameters must be specified.

Parameter	Description
dUser	The defined user.
IdcService	Must be set to `GET_USER_INFO`.

31.3.3.4.2 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_USER_
INFO">
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.4.3 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_USER_
INFO">
<idc:document>
<idc:field name="changedSubjects">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
<idc:optionlist name="Users_UserLocaleList">
<idc:option>
English-US
</idc:option>
</idc:optionlist>
</idc:document>
<idc:user dUser="sysadmin" dName="sysadmin">
<idc:resultset name="UserMetaDefinition">
<idc:row umdName="dFullName" umdType="BigText" umdCaption="apTitleFullName"
umdIsOptionList="0" umdOptionListType="0" umdOptionListKey="" umdIsAdminEdit="0"
umdOverrideBitFlag="1">
</idc:row>
<idc:row umdName="dEmail" umdType="BigText" umdCaption="apTitleEmailAddress"
umdIsOptionList="0" umdOptionListType="" umdOptionListKey="" umdIsAdminEdit="0"
umdOverrideBitFlag="2">
</idc:row>
<idc:row umdName="dUserType" umdType="Text" umdCaption="apTitleUserType"
umdIsOptionList="1" umdOptionListType="combo" umdOptionListKey="Users_
UserTypeList" umdIsAdminEdit="0" umdOverrideBitFlag="4">
</idc:row>
<idc:row umdName="dUserLocale" umdType="Text" umdCaption="apTitleUserLocale"
umdIsOptionList="1" umdOptionListType="choice,locale" umdOptionListKey="Users_
UserLocaleList" umdIsAdminEdit="0" umdOverrideBitFlag="8">
</idc:row>
</idc:resultset>
<idc:resultset name="USER_INFO">
<idc:row dName="sysadmin" dFullName="System Administrator" dEmail=""
dPasswordEncoding="" dPassword="-----" dUserType="" dUserAuthType="LOCAL"
dUserOrgPath="" dUserSourceOrgPath="" dUserSourceFlags="0" dUserArriveDate=""
dUserChangeDate="" dUserLocale="" dUserTimeZone="">
</idc:row>
</idc:resultset>
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.5 Delete User

The DELETE_USER service deletes an existing user.

- Given a user name, the service deletes the user from the system.
- The most likely error is when the user has been assigned to an alias.If this service is unable to execute, an error message is returned.

31.3.3.5.1 Required Parameters These parameters must be specified.

Parameter	Description
dName	The unique name.
IdcService	Must be set to DELETE_USER.

31.3.3.5.2 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="DELETE_
USER">
<idc:user dName="Jennifer" >
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.5.3 Response

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/"
IdcService="DELETE_USER">
<idc:document>
<idc:field name="changedSubjects">
userlist,1018884022876
</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="dUserName">
Jennifer
</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
</idc:document>
<idc:user dUser="sysadmin" dName="Jennifer">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.6 Check In Content Item

The CHECKIN_UNIVERSAL service performs a controlled check-in to Content Server:

- This service determines if the content item is new or already exists in the system by querying the database using the content ID (dDocName) as the key.
- If the content item exists in the system, the publish state (dPublishState) must be empty.
- If a revision label (dRevLabel) is specified, this service will check if the content revision exists in the system; an exception is thrown if the revision exists.
- This service will dispatch this request to one of these subservices:
  - CHECKIN_NEW_SUB - If the content item does not exist in the server.
  - CHECKIN_SEL_SUB - If the content item exists on the system and no valid revision was specified and the content item is checked out.
  - WORKFLOW_CHECKIN_SUB - If the content item exists and is part of a workflow.
- The most likely errors are mismatched parameters or when the content item was not successfully checked in. If this service is unable to execute, this message is displayed to the user: `Content item ''{dDocName}'' was not successfully checked in.`

The CHECKIN_UNIVERSAL service is a controlled check-in to Content Server. The check-in will fall into either a new, selected, or workflow check-in process and follow the same logic as a check-in through the browser or Repository Manager application. If the content item to be checked in already exists in the system, the content item must be checked out for the check in to succeed.

These are essentially the same subservices used during a controlled check-in to Content Server. However, these subservices are not called during a BatchLoad or Archive import. This service will check security to determine if the user has sufficient privilege to perform a check in on the content item and if the content item (if it exists) has been checked out. Also, it will determine if the content item matches a workflow criteria or belongs to an active basic workflow.

If the content item is not found the content item is checked in using the CHECKIN_NEW_SUB subservice. This subservice validates the check in data and determines if this content item belongs to a criteria workflow. If the content item already exists in the system and the content item does not belong to a workflow, the CHECKIN_SEL_SUB is used. Otherwise the content item exists and belongs to a workflow and the WORKFLOW_CHECKIN_SUB is used.

Note: All paths use the slash (/) as the file separator, because the backslash (\) is an escape character. For example, `primaryFile=d:/temp/myfile.txt` should point to the primary file to check in.

31.3.3.6.1 Required Parameters These parameters must be specified.

Parameter	Description
dDocAuthor	The content item author (contributor).
dDocName	The content item identifier (Content ID). ■ This field is optional if the system has been configured with `IsAutoNumber` set to `TRUE`. In this scenario, if the `dDocName` is not specified, the check in will always be new, and the system will generate a new name for the content item. ■ Otherwise, if `dDocName` is specified, the service will use this key to do a look up to determine what type of check in to perform.
dDocTitle	The content item title.
dDocType	The content item type.
doFileCopy	Set this flag to `TRUE` (1) or the file will be removed from your hard drive.
dSecurityGroup	The security group such as `PUBLIC` or `SECURE`.
IdcService	Must be set to `CHECKIN_UNIVERSAL`.
primaryFile	The absolute path to the location of the file as seen from the server. Use the slash as the file separator. A primary file must be specified unless checking in metadata only. If an alternate file is specified with the primary file, Oracle WebCenter Content: Inbound Refinery will convert the alternate file. Otherwise, the primary file will be converted. ■ If a primary file is not specified, a metafile can be used in its place. Only one metafile can exist though for each content item (that is, a primary AND alternate meta file cannot coexist). ■ If both a primary and alternate file is specified, their extensions must be different.

Important: Custom metadata fields that are defined must also be specified.

31.3.3.6.2 Additional Parameters This parameter may be required.

Parameter	Description
dDocAccount	The security account for the content item. If you have accounts enabled, you must pass this parameter.

31.3.3.6.3 Optional Parameters These optional parameters may be specified.

Parameter	Description
alternateFile	The alternate file for conversion. ■ Only one metafile can exist though for each content item (a primary AND alternate meta file cannot coexist.) ■ If an alternate file is specified with the primary file, Inbound Refinery will convert the alternate file. Otherwise, the primary file will be converted.
dCreateDate	The date the content item was created. By default, this is the current date.
dInDate	The content release date. The date the content item is to be released to the web. By default, this is the current date. If the content release date (dInDate) is not specified, the creation date (dCreateDate) is used. This value is auto generated if it is not supplied.
dOutDate	The content expiration date. By default, this is blank and does not specify an expiration date. If the content expiration date (dOutDate) is not entered, the value remains empty. This is a valid state.
dRevLabel	The revision label for the content item. If set, the label will be used to locate the specified revision.
isFinished	Set to `TRUE` (1) if this is a workflow check-in and you have finished editing it. See `WORKFLOW_CHECKIN` for additional information.

Note: Do not confuse the Content ID (`dDocName`) with the internal content item revision identifier (`dID`). The `dID` value is a generated reference to a specific rendition of a content item

31.3.3.6.4 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="CHECKIN_
UNIVERSAL">
<idc:document dDocName="SoapUpload2" dDocAuthor="sysadmin" dDocTitle="Soap Upload
2 Document" dDocType="ADACCT" dSecurityGroup="Public" dDocAccount="">
<idc:file name="primaryFile"
href="C:/stellent/custom/Soap/JavaSamples/SoapClientUpload/soaptest.doc">
</idc:file>
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.6.5 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="CHECKIN_
UNIVERSAL">
<idc:document dDocAuthor="sysadmin" dDocName="SoapUpload2" dExtension="doc"
dDocAccount="" dIsPrimary="1" dRevisionID="1" dPublishType="" dInDate="4/22/02
1:31PM" dReleaseState="N" dRevClassID="12" dCreateDate="4/22/02 1:31 PM"
dIsWebFormat="0" dPublishState="" dLocation="" dStatus="DONE"
```

```
dOriginalName="12.doc" dOutDate="" dDocID="24" dRevLabel="1" dProcessingState="Y"
dDocTitle="Soap Upload 2 Document" dID="12" dDocType="ADACCT"
dSecurityGroup="Public" dFileSize="19456" dFormat="application/msword">
<idc:field name="primaryFile:path">
c:/stellent/vault/~temp/1230750423.doc
</idc:field>
<idc:field name="dRawDocID">
23
</idc:field>
<idc:field name="changedSubjects">
documents,1019482656706
</idc:field>
<idc:field name="StatusCode">
0
</idc:field>
<idc:field name="soapFile:path">
c:/stellent/vault/~temp/1230750422.xml
</idc:field>
<idc:field name="xComments">

</idc:field>
<idc:field name="soapStartContentID">
SoapContent
</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="dActionDate">
4/22/02 1:31 PM
</idc:field>
<idc:field name="dActionMillis">
30263
</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="WebfilePath">
c:/stellent/weblayout/groups/public/documents/adacct/soapupload2~1.doc
</idc:field>
<idc:field name="StatusMessage">
Successfully checked in content item &#39;SoapUpload2&#39;.
</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="dConversion">
PASSTHRU
</idc:field>
<idc:field name="primaryFile">
C:/stellent/custom/Soap/JavaSamples/SoapClientUpload/soaptest.doc
</idc:field>
<idc:field name="dAction">
Checkin
</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
```

```
<idc:field name="VaultfilePath">
c:/stellent/vault/adacct/12.doc
</idc:field>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.7 Check out Content Item

The CHECKOUT_BY_NAME checks out the latest revision of the specified content item.

- Given a content item revision ID, this service attempts to locate the content item in the system and undo the checkout.
- The service fails if the content item does not exist in the system, if the content item is not checked out, or the user does not have sufficient privilege to undo the checkout.
- The most likely error is a content item name that does not exist. If this service is unable to execute, an error message is displayed to the user.

> **Note:** This service only marks the content item as locked. It does not perform a download.

31.3.3.7.1 Required Parameters These parameters must be specified.

Parameter	Description
dDocName	The content item identifier (Content ID).
IdcService	Must be set to CHECKOUT_BY_NAME.

> **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The dID value is a generated reference to a specific rendition of a content item.

31.3.3.7.2 Optional Parameters This optional parameter may be specified.

Parameter	Description
dDocTitle	The content item title.

31.3.3.7.3 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="CHECKOUT_
BY_NAME">
<idc:document dDocName="soap_sample">
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.7.4 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="CHECKOUT_
BY_NAME">
<idc:document dDocTitle="soap_sample" dID="10" dRevLabel="1" dDocAccount=""
dRevClassID="10" dDocName="soap_sample" dOriginalName="soap_sample.txt"
dSecurityGroup="Public">
<idc:field name="dActionMillis">
39964
</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
<idc:field name="dActionDate">
4/22/02 12:20 PM
</idc:field>
<idc:field name="latestID">
10
</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="CurRevID">
10
</idc:field>
<idc:field name="CurRevIsCheckedOut">
0
</idc:field>
<idc:field name="dAction">
Check out
</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="CurRevCheckoutUser">
sysadmin
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="changedSubjects">
documents,1019482656687
</idc:field>
<idc:resultset name="DOC_INFO">
<idc:row dID="10" dDocName="soap_sample" dDocType="ADACCT" dDocTitle="soap_sample"
dDocAuthor="sysadmin" dRevClassID="10" dRevisionID="1" dRevLabel="1"
dIsCheckedOut="1" dCheckoutUser="sysadmin" dSecurityGroup="Public"
dCreateDate="4/22/02 12:18 PM" dInDate="4/22/02 12:18 PM" dOutDate=""
dStatus="RELEASED" dReleaseState="Y" dFlag1="" dWebExtension="txt"
dProcessingState="Y" dMessage="" dDocAccount="" dReleaseDate="4/22/02 12:19 PM"
dRendition1="" dRendition2="" dIndexerState="" dPublishType="" dPublishState=""
dDocID="19" dIsPrimary="1" dIsWebFormat="0" dLocation="" dOriginalName="soap_
sample.txt" dFormat="text/plain" dExtension="txt" dFileSize="12">
<idc:field name="xComments">
```

```
</idc:field>
</idc:row>
</idc:resultset>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.8 Undo Content Item Checkout

The UNDO_CHECKOUT_BY_NAME service reverses a content item checkout using the Content ID.

- Given a content item name, this service attempts to locate the content item in the system and undo the checkout.
- The service fails if the content item does not exist in the system, if the content item is not checked out, or if the user does not have sufficient privilege to undo the checkout.
- This service is used by an applet or application.
- If this service is unable to execute, this message is displayed to the user: `Unable to undo checkout for ''{dDocName}''.`

31.3.3.8.1 Required Parameters These parameters must be specified.

Parameter	Description
dDocName	The content item identifier (Content ID).
IdcService	Must be set to UNDO_CHECKOUT_BY_NAME.

> **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The dID value is a generated reference to a specific rendition of a content item.

31.3.3.8.2 Optional Parameters This optional parameter may be specified.

Parameter	Description
dDocTitle	The content item title.

31.3.3.8.3 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="UNDO_
CHECKOUT_BY_NAME">
<idc:document dDocName="soap_sample">
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.8.4 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="UNDO_
CHECKOUT_BY_NAME">
<idc:document dCheckoutUser="sysadmin" dPublishState="" dDocTitle="soap_sample"
dID="10" dRevLabel="1" dDocAccount="" dDocName="soap_sample" dRevClassID="10"
dOriginalName="soap_sample.txt" dSecurityGroup="Public">
<idc:field name="dActionMillis">
5317
</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
<idc:field name="dActionDate">
4/22/02 12:23 PM
</idc:field>
<idc:field name="latestID">
10
</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="CurRevID">
10
</idc:field>
<idc:field name="CurRevIsCheckedOut">
1
</idc:field>
<idc:field name="dAction">
Undo Checkout
</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="CurRevCheckoutUser">
sysadmin
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="changedSubjects">
documents,1019482656689
</idc:field>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.9 Get Content Item Information

The DOC_INFO service retrieves content item revision information.

- Given a content item revision ID, the service retrieves content item revision information
- The most likely errors are when the content item no longer exists in the system or when the user does not have the security level to perform this action. If this service is unable to execute, an error message is displayed to the user.

31.3.3.9.1 Required Parameters These parameters must be specified.

Parameter	Description
dID	The generated content item revision ID.
IdcService	Must be set to DOC_INFO.

31.3.3.9.2 SOAP Request <?xml version='1.0' ?>

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="DOC_INFO">
<idc:document dID="6">
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.9.3 Response <?xml version='1.0' ?>

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="DOC_INFO">
<idc:document dStatus="RELEASED" dDocFormats="text/plain" dID="6"
DocUrl="HTTP://wharristest/stellent/groups/public/documents/adacct/stellent.txt"
dDocTitle="stellent">
<idc:field name="dSubscriptionAlias">
sysadmin
</idc:field>
<idc:field name="changedSubjects">

</idc:field>
<idc:field name="dSubscriptionID">
stellent
</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
<idc:field name="dSubscriptionType">
```

```
Basic
</idc:field>
<idc:resultset name="REVISION_HISTORY">
<idc:row dFormat="text/plain" dInDate="4/12/02 1:27 PM" dOutDate=""
dStatus="RELEASED" dProcessingState="Y" dRevLabel="1" dID="6" dDocName="stellent"
dRevisionID="1">
</idc:row>
</idc:resultset>
<idc:resultset name="WF_INFO">
</idc:resultset>
<idc:resultset name="DOC_INFO">
<idc:row dID="6" dDocName="stellent" dDocType="ADACCT" dDocTitle="stellent"
dDocAuthor="sysadmin" dRevClassID="6" dRevisionID="1" dRevLabel="1"
dIsCheckedOut="0" dCheckoutUser="" dSecurityGroup="Public" dCreateDate="4/12/02
1:27 PM" dInDate="4/12/02 1:27 PM" dOutDate="" dStatus="RELEASED"
dReleaseState="Y" dFlag1="" dWebExtension="txt" dProcessingState="Y" dMessage=""
dDocAccount="" dReleaseDate="4/12/02 1:27 PM" dRendition1="" dRendition2=""
dIndexerState="" dPublishType="" dPublishState="" dDocID="11" dIsPrimary="1"
dIsWebFormat="0" dLocation="" dOriginalName="stellent.txt" dFormat="text/plain"
dExtension="txt" dFileSize="8">
<idc:field name="xComments">
stellent
</idc:field>
</idc:row>
</idc:resultset>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.10 Get File

The `GET_FILE` service returns a specific rendition of a content item, the latest revision, or the latest released revision. A copy of the file is retrieved without performing a check out.

- This command computes the `dID` (content item revision ID) for the revision, and then determines the file name of a particular rendition of the revision with the computed `dID`. A specified `dID` or a `dDocName` (content item name) along with a `RevisionSelectionMethod` parameter can be used.
- Given a `dID` or a `dDocName` along with a `RevisionSelectionMethod` parameter, the service determines the file name of a particular rendition of the revision and returns that file to the client.
- The most likely errors are some form of mismatched parameters or a request for a revision or rendition that does not exist. If this service is unable to execute, an error message is displayed to the user.

> **Note:** Use *dDocName* in all requests for content items where the requester knows the *dDocName* value. Error messages in Content Server are based on the assumption that the *dDocName* value is present, as are other features, such as forms.

31.3.3.10.1 Required Parameters

Important: Either the content item revision ID (dID) must be specified or a content item name (dDocName) along with a RevisionSelectionMethod parameter must be defined.

Parameter	Description
dDocName	The content item identifier (Content ID). ■ If dDocName is not present, dID must be present, and RevisionSelectionMethod must not be present. ■ If RevisionSelectionMethod is present, a rendition of a revision of the content item with this name will be returned, if it exists. ■ If RevisionSelectionMethod is not present, dDocName can be used in error messages.
dID	The generated content item revision ID. ■ If dID is not specified, dDocName, and RevisionSelectionMethod must specified. ■ A rendition of the revision of the content item with this ID will be returned, if it exists, and the RevisionSelectionMethod parameter does not exist or has the value Specific.
RevisionSelectionMethod	The revision selection method. If present, dDocName must be present. The value of this variable is the method used to compute a dID from the specified dDocName. Its value can be Specific, Latest, or LatestReleased. ■ If the value is Specific, dDocName is ignored, and dID is required, and it is used to get a rendition. ■ If the value is Latest, the latest revision of the content item is used to compute the dID. ■ If the value is LatestReleased, the latest released revision of the content item is used to compute the dID.
IdcService	Must be set to GET_FILE.

31.3.3.10.2 Optional Parameter

This optional parameters may be specified.

Parameter	Description
Rendition	The content item rendition. This parameter specifies the rendition of the content item and can be set to *Primary*, *Web*, or *Alternate*. If Rendition is not present, it defaults to *Primary*. ■ If the value is *Primary*, the primary rendition of the selected revision is returned. ■ If the value is *Web*, the web viewable rendition of the selected revision is returned. ■ If the value is *Alternate*, the alternate rendition of the selected revision is returned.

Note: Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The dID value is a generated reference to a specific rendition of a content item.

31.3.3.10.3 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_FILE">
<idc:document dID="10">
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.10.4 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_FILE">
<idc:document dID="10">
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Receving response...
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Connection: keep-alive
Date: Mon, 29 Apr 2002 16:09:42 GMT
Content-type: Multipart/Related; boundary=----------------4002588859573015789;
type=text/xml; start="<SoapContent>"
Content-Length: 1717

------------------4002588859573015789
Content-Type: text/xml; charset=utf-8
Content-ID:  <SoapContent>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_FILE">
<idc:document dID="10" dExtension="txt">
<idc:field name="changedSubjects">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">
```

```
</idc:field>
<idc:resultset name="FILE_DOC_INFO">
<idc:row dID="10" dDocName="soap_sample" dDocType="ADACCT" dDocTitle="soap_sample"
dDocAuthor="sysadmin" dRevClassID="10" dRevisionID="1" dRevLabel="1"
dIsCheckedOut="0" dCheckoutUser="" dSecurityGroup="Public" dCreateDate="4/22/02
12:18PM" dInDate="4/22/02 12:18 PM" dOutDate="" dStatus="RELEASED"
dReleaseState="Y" dFlag1="" dWebExtension="txt" dProcessingState="Y" dMessage=""
dDocAccount="" dReleaseDate="4/22/02 12:19 PM" dRendition1="" dRendition2=""
dIndexerState="" dPublishType="" dPublishState="" dDocID="19" dIsPrimary="1"
dIsWebFormat="0" dLocation="" dOriginalName="soap_sample.txt" dFormat="text/plain"
dExtension="txt" dFileSize="12">
<idc:field name="xComments">

</idc:field>
</idc:row>
</idc:resultset>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

------------------4002588859573015789
Content-Type: text/html
Content-ID: <soap_sample.txt>

...File content...
------------------4002588859573015789--
```

31.3.3.11 Get Search Results

The `GET_SEARCH_RESULTS` service retrieves the search results for the passed query text.

- Used to display the search results to a user making a content item query.
- You can append values for `Title`, `Content ID`, and so on, in the `QueryText` parameter, to refine this service.

  The `QueryText` parameter defines the query. For use in a SOAP message, this query must be XML-encoded. This example passes a string submitted for a content item query in both standard format and XML-encoded format:

  – Parameter with standard formatted string:

  ```
  QueryText=dDocType <Substring> "ADSALES"
  ```

  – Parameter with XML-encoded string:

  ```
  <idc:field name="QueryText">
  dDocType &lt;Substring&gt; `ADSALES`
  </idc:field>
  ```

  For more information about formatting XML-encoded strings, see Section 31.3.2, "Special Characters."

- If this service is unable to execute, it displays the following message: `Unable to retrieve search results.`

31.3.3.11.1 Required Parameters These parameters must be specified.

Parameter	Description
IdcService	Must be set to `GET_SEARCH_RESULTS`.
QueryText	The user supplied text submitted for the content item query.

31.3.3.11.2 Optional Parameters These parameters may be specified.

Parameter	Description
resultCount	The number of results to return, defaults to 25.
sortField	The name of the metadata field to sort on. ■ Examples: dInDate, dDocTitle, Score. ■ Defaults to dInDate.
sortOrder	The sort order. Allowed values are ASC (ascending) and DES (descending).
startRow	The row to begin the search results. For example, if a result returns 200 rows, and resultCount is 25, set startRow to 26 to obtain the second set of results.

31.3.3.11.3 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_SEARCH_
RESULTS">
<idc:document>
<idc:field name="QueryText">
dDocType <Substring> "ADSALES"
</idc:field>
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.11.4 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_SEARCH_
RESULTS">
<idc:document StartRow="1" TotalDocsProcessed="6" TotalRows="0"
QueryText="dDocType+%3cSubstring%3e+%22ADSALES%22" EndRow="25"
SearchProviders="Master_on_wharristest" NumPages="0" PageNumber="1">
<idc:field name="refreshMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="EnterpriseSearchMaxRows">
4
</idc:field>
```

```
<idc:field name="FullRequest">
&QueryText=dDocType+%3cSubstring%3e+%22ADSALES%22
</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="changedSubjects">

</idc:field>
<idc:field name="Text2">
&lt;$dDocTitle$&gt;
</idc:field>
<idc:field name="Text1">
<$dDocName$>
</idc:field>
<idc:field name="OriginalQueryText">
dDocType+%3cSubstring%3e+%22ADSALES%22
</idc:field>
<idc:resultset name="SearchResults">
</idc:resultset>
<idc:resultset name="NavigationPages">
</idc:resultset>
<idc:resultset name="Master_on_wharristest">
</idc:resultset>
<idc:resultset name="EnterpriseSearchResults">
<idc:row ProviderName="Master_on_wharristest" IDC_Name="Master_on_wharristest"
TotalRows="0" TotalDocsProcessed="6">
<idc:field name="ProviderDescription">
!csProviderLocalContentServerLabel
</idc:field>
<idc:field name="InstanceMenuLabel">
Master_on_wharristest
</idc:field>
<idc:field name="InstanceDescription">
Master_on_wharristest
</idc:field>
<idc:field name="IntradocServerHostName">
wharristest
</idc:field>
<idc:field name="HttpRelativeWebRoot">
/stellent/
</idc:field>
<idc:field name="IsImplicitlySearched">

</idc:field>
<idc:field name="UserAccounts">
#all
</idc:field>
<idc:field name="IsLocalCollection">
true
</idc:field>
<idc:field name="Selected">

</idc:field>
<idc:field name="StatusMessage">
Success
</idc:field>
```

```
<idc:field name="ResultSetName">
Master_on_wharristest
</idc:field>
<idc:field name="SearchCgiWebUrl">
/idcplg/idc_cgi_isapi.dll/stellent/pxs
</idc:field>
</idc:row>
</idc:resultset>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.12 Get Table Data

The GET_TABLE service exports the specified table in the WebCenter Content database.

- Exports the specified table by creating a ResultSet and adding it to the serialized HDA file. If the table is not found, the service will fail. It is up to the calling program that is receiving the serialized HDA file to store this ResultSet for later use.
- The most likely error is a table name that does not exist. If this service is unable to execute, an error message is displayed to the user.

31.3.3.12.1 Required Parameters These parameters must be specified.

Parameter	Description
IdcService	Must be set to GET_TABLE.
tableName	The name of table to export.

31.3.3.12.2 SOAP Request

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_TABLE">
<idc:document>
<idc:field name="tableName">
DocTypes
</idc:field>
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.12.3 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_TABLE">
<idc:document>
<idc:field name="tableName">
DocTypes
</idc:field>
<idc:field name="changedSubjects">

</idc:field>
```

```
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
<idc:resultset name="DocTypes">
<idc:row dDocType="ADACCT" dDescription="Acme Accounting Department"
dGif="adacct.gif">
</idc:row>
<idc:row dDocType="ADCORP" dDescription="Acme Corporate Department"
dGif="adcorp.gif">
</idc:row>
<idc:row dDocType="ADENG" dDescription="Acme Engineering Department"
dGif="adeng.gif">
</idc:row>
<idc:row dDocType="ADHR" dDescription="Acme Human Resources Department"
dGif="adhr.gif">
</idc:row>
<idc:row dDocType="ADMFG" dDescription="Acme Manufacturing Department"
dGif="admfg.gif">
</idc:row>
<idc:row dDocType="ADMKT" dDescription="Acme Marketing Department"
dGif="admkt.gif">
</idc:row>
<idc:row dDocType="ADSALES" dDescription="Acme Sales Department"
dGif="adsales.gif">
</idc:row>
</idc:resultset>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.13 Get Criteria Workflow Information

The `GET_CRITERIA_WORKFLOWS_FOR_GROUP` service returns criteria workflow information.

- Given a named security group, this service returns a list of workflows and related steps.
- Returns the Resultset `WorkflowsForGroup` and `WorkflowStepsForGroup`:
  - `WorkflowsForGroup` lists all of the workflows for this group (`dWfID`, `dWfName`).
  - `WorkflowStepsForGroup` lists all of the steps in all of the workflows for this group (`dWfID`, `dWfName`, `dWfStepID`, `dWfStepName`).

- Criteria workflows and subworkflows can be added, edited, enabled, disabled, and deleted from the Criteria tab of the Workflow Admin administration applet.
- The most likely error is a named security group that does not exist or a user failing the security check. The service throws reasonable exceptions for display to the user in these situations.

31.3.3.13.1 Required Parameters

These parameters must be specified.

Parameter	Description
dSecurityGroup	The security group such as `PUBLIC` or `SECURE`.
IdcService	Must be set to `GET_CRITERIA_WORKFLOWS_FOR_GROUPS`.

31.3.3.13.2 SOAP Request

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_
CRITERIA_WORKFLOWS_FOR_GROUP">
<idc:document dSecurityGroup="Public" />
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.3.3.13.3 Response

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_
CRITERIA_WORKFLOWS_FOR_GROUP">
<idc:document dSecurityGroup="Public">
<idc:field name="changedSubjects">

</idc:field>
<idc:field name="refreshSubjects">

</idc:field>
<idc:field name="loadedUserAttributes">
1
</idc:field>
<idc:field name="changedMonikers">

</idc:field>
<idc:field name="refreshSubMonikers">

</idc:field>
<idc:field name="refreshMonikers">

</idc:field>
<idc:resultset name="WorkflowStepsForGroup">
<idc:row>
<idc:field name="dWfID">
1
</idc:field>
<idc:field name="dWfName">
TestWorkflow
</idc:field>
<idc:field name="dWfStepID">
1
</idc:field>
```

```
<idc:field name="dWfStepName">
contribution
</idc:field>
</idc:row>
<idc:row>
<idc:field name="dWfID">
1
</idc:field>
<idc:field name="dWfName">
TestWorkflow
</idc:field>
<idc:field name="dWfStepID">
2
</idc:field>
<idc:field name="dWfStepName">
StepOne
</idc:field>
</idc:row>
</idc:resultset>
<idc:resultset name="WorkflowsForGroup">
<idc:row>
<idc:field name="dWfID">
1
</idc:field>
<idc:field name="dWfName">
TestWorkflow
</idc:field>
</idc:row>
</idc:resultset>
</idc:document>
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.4 Using SOAP Packets in Active Server Pages

You can execute Content Server IdcCommand services from an Active Server Page by encapsulating a SOAP packet that defines the service to execute and the required parameters. You must have appropriate permissions to execute the commands. Some commands require administrative access, other commands may require only write permission.

31.4.1 Sample SOAP Request

An Active Server Page can call a service from Content Server. The following description of a sample service includes the required and optional parameters. It also provides an XML-formatted version of the embedded SOAP request.

For more information about service calls, including required and optional parameters, see Section 31.3.3, "Sample Service Calls with SOAP Response/Request."

In the following example, an XML-formatted SOAP request uses the GET_SEARCH_RESULTS service to retrieve the search results for the passed query text.

```
<?xml version='1.0' ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_SEARCH_RESULTS">
<idc:document>
<idc:field name="QueryText">
dDocType <Substring> "ADSALES"
</idc:field>
</idc:document>
</idc:service>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

31.4.2 Sample Active Server Page

The embedded SOAP request forms the basis of the Active Server Page. The following sample executes GET_SEARCH_RESULTS.

For more information about service calls and examples of SOAP response/request messages, see Section 31.3.3, "Sample Service Calls with SOAP Response/Request."

```
<%

' Sample ASP page of sending a DOC_INFO Soap request.

Option Explicit

Response.Write("Search Results")

%>
<br><br>
<%

' Construct the Soap request.
Dim strSoapRequest, strQueryText

strQueryText = Request.Form("QueryText")
strQueryText = Server.HtmlEncode(strQueryText)

strSoapRequest = "<?xml version='1.0' ?>" _
& "<SOAP-ENV:Envelope xmlns:SOAP-ENV=""http://schemas.xmlsoap.org/soap/envelope/"">" _
& "<SOAP-ENV:Body>" _
& "<idc:service xmlns:idc=""http://www.oracle.com/IdcService/"" IdcService=""GET_SEARCH_
RESULTS"">" _
& "<idc:document>" _
& "<idc:field name=""QueryText"">" & strQueryText & "</idc:field>" _
& "<idc:field name=""SortField"">" & Request.Form("SortField") & "</idc:field>" _
& "<idc:field name=""SortOrder"">" & Request.Form("SortOrder") & "</idc:field>" _
& "<idc:field name=""ResultCount"">" & Request.Form("ResultCount") & "</idc:field>" _
& "<idc:field name=""Auth"">Internet</idc:field>" _
& "</idc:document>" _
& "</idc:service>" _
& "</SOAP-ENV:Body>" _
& "</SOAP-ENV:Envelope>"

' Send the Soap request.
Dim objXmlHttp
Set objXmlHttp = Server.CreateObject("MSXML2.ServerXMLHTTP")
objXmlHttp.open "POST", "http://localhost/stellent/idcplg", False, "sysadmin", "idc"
objXmlHttp.setRequestHeader "Content-Type", "text/xml; charset=utf-8"
objXmlHttp.send(strSoapRequest)

' Parse the Soap response.
Dim objXmlDoc
Set objXmlDoc = Server.CreateObject("Msxml2.DOMDocument")
```

```
objXmlDoc.async = False
objXmlDoc.Load objXmlHttp.responseXml

‘ Check for errors.
Dim strResponseError
strResponseError = objXmlDoc.parseError.reason
If strResponseError <> "" Then
Response.Write(objXmlHttp.ResponseText)
DisplayBackButton()
Response.End
End If

‘ Check for a fault string.
Dim objXmlFaultNode
Set objXmlFaultNode =
objXmlDoc.documentElement.selectSingleNode("//SOAP-ENV:Fault/faultstring")
If (Not (objXmlFaultNode Is Nothing)) Then
Response.Write(objXmlFaultNode.Text)
DisplayBackButton()
Response.End
End If

‘ Check the status code.
Dim objXmlStatusCodeNode, objXmlStatusMessageNode, strStatusCode, nStatusCode,
strStatusMessage
Set objXmlStatusCodeNode =
objXmlDoc.documentElement.selectSingleNode("//idc:field[@name=’StatusCode’]")
If (Not objXmlStatusCodeNode Is Nothing) Then
nStatusCode = CInt(objXmlStatusCodeNode.Text)
If (nStatusCode < 0) Then
Response.Write(objXmlDoc.documentElement.selectSingleNode("//idc:field[@name=’StatusMessage’]
").Text)
DisplayBackButton()
Response.End
End If
End If

‘ Display search results
Dim strDocName, strDocTitle, strDocType, strInDate, strComments, nCurRow, nTotalRows
Dim objXmlResultNodeList, objXmlCommentNode

Set objXmlResultNodeList =
objXmlDoc.documentElement.selectNodes("//idc:resultset[@name=’SearchResults’]/idc:row")
nTotalRows = objXmlResultNodeList.Length

%>
<table>
<tr>
<td><b>Content ID</b></td>
<td> </td>
<td><b>Title</b></td>
<td> </td>
<td><b>Type</b></td>
<td> </td>
<td><b>Release Date</b></td>
<td> </td>
<td><b>Comments</b></td>
</tr>

<%
For nCurRow = 0 To (nTotalRows - 1)
strDocName = GetXmlNodeValue(objXmlResultNodeList.Item(nCurRow), "dDocName")
strDocTitle = GetXmlNodeValue(objXmlResultNodeList.Item(nCurRow), "dDocTitle")
strDocType = GetXmlNodeValue(objXmlResultNodeList.Item(nCurRow), "dDocType")
strInDate = GetXmlNodeValue(objXmlResultNodeList.Item(nCurRow), "dInDate")
strComments = GetXmlNodeValue(objXmlResultNodeList.Item(nCurRow), "xComments")
```

```
%>

<tr>
<td><%=strDocName%></td>
<td> </td>
<td><%=strDocTitle%></td>
<td> </td>
<td><%=strDocType%></td>
<td> </td>
<td><%=strInDate%></td>
<td> </td>
<td><%=strComments%></td>
</tr>
<%
Next
%>

</table>

<%

DisplayBackButton()
'----------------------------
Function GetXmlNodeValue(objXmlRowNode, strNodeName)
'----------------------------
Dim objXmlNode, objXmlNodeValue

Set objXmlNode = objXmlRowNode.selectSingleNode("@" & strNodeName)
If (objXmlNode Is Nothing) Then
Set objXmlNode = objXmlRowNode.selectSingleNode("idc:field[@name='" & strNodeName & "']")
End If

If (Not (objXmlNode Is Nothing)) Then
GetXmlNodeValue = objXmlNode.Text
End If
'----------------------------
End Function
'----------------------------

'----------------------------
Sub DisplayBackButton()
'----------------------------
%>
<form method=POST action="request.asp">
<table>
<tr>
<td><input type=submit value="Back"></td>
</tr>
</table>
</form>
<%
'----------------------------
End Sub
'----------------------------
%>
```

31.5 Generating WSDL Files to Access WebCenter Content

You can generate WSDL files for interfacing with WebCenter Content services.

31.5.1 Understanding WSDL Files

WSDL files provide the ability to pass data that can be understood by Content Server services, which enables access to the content and content management functions within WebCenter Content. The WSDL files provided with the component are stored in the *IntradocDir*/weblayout/groups/secure/wsdl/custom directory.

These WSDL files are provided with the WSDL Generator component:

- CheckIn.wsdl
- DocInfo.wsdl
- GetFile.wsdl
- MetaData.wsdl
- PortalInfo.wsdl
- Search.wsdl
- Subscription.wsdl
- Workflow.wsdl

Additional WSDL files can be generated using the Soap Custom WSDL administrative pages. See Section 31.5.2, "Sample WSDL File," for additional information.

31.5.1.1 WSDL File Structure

WSDL files are formally structured with elements that contain a description of the data to be passed to the web service. This structure enables both the sending application and the receiving application to interpret the data being exchanged.

WSDL elements contain a description of the operation to perform on the data and a binding to a protocol or transport. This permits the receiving application to both process the data and interpret how to respond or return data. Additional subelements may be contained within each WSDL element.

The WSDL file structure includes these major elements:

- **Data Types**: Generally in the form of XML schema to be used in the messages.
- **Message**: The definition of the data in the form of a message either as a complete document or as arguments to be mapped to a method invocation.
- **Port Type**: A set of operations mapped to an address. This defines a collection of operations for a binding.
- **Binding**: The actual protocol and data formats for the operations and messages defined for a particular port type.
- **Service and Port**: The service maps the binding to the port and the port is the combination of a binding and the network address for the communication exchange.

Note: The following code fragments are from the DocInfo.wsdl file provided with the WSDL Generator component. For a complete WSDL file, see Section 31.5.2, "Sample WSDL File."

31.5.1.1.1 Data Type The Data Type <types> defines the complex types and associated elements. Web services supports both simple data types (such as string, integer, or boolean) and complex data types. A complex type is a structured XML document that contains several simple types or an array of subelements.

The following code fragment for the ContentInfo set defines the Name, Title, Author, and Group elements and specifies that they are strings.

```
<s:complexType name="ContentInfo">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dDocName" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocTitle" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocType" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocAuthor" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dSecurityGroup" type="s:string"/>
</s:sequence>
</s:complexType>
```

31.5.1.1.2 Message The Message <message> defines the data as arguments to be mapped to a method invocation.

```
<message name="DocInfoByIDSoapIn">
<part name="parameters" element="s0:DocInfoByID" />
</message>
<message name="DocInfoByIDSoapOut">
<part name="parameters" element="s0:DocInfobyIDResponse" />
</message>
```

31.5.1.1.3 Port Type The Port Type <portType> defines a collection of operations for a binding. The DocInfo.wsdl file provides the DocInfoSoap and the DocInfo operation name (method name) with I/O information for processing the message.

```
<portType name="DocInfoSoap">
<operation name="DocInfoByID">
<input message="s0:DocInfoByIDSoapIn" />
<output message="s0:DocInfoByIDSoapOut" />
</operation>
</portType>
```

Note: While a port type is a collection of operations (like classes in Java), WSDL is an independent data abstraction that provides more functionality than simply mapping to .NET, EJB, or CORBA objects.

31.5.1.1.4 Binding The binding <binding> defines the actual protocol and data formats for the operations and messages for the particular port type.

```
<binding name="DocInfoSoap" type="s0:DocInfoSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="DocInfoByID">
<soap:operation soapAction="http://wwww.oracle.com/Soap/DocInfo/" style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
</binding>
```

31.5.1.1.5 Service and Port The service <service> maps the binding to the port. The port is the combination of a binding and the network address for the communication exchange. The port is used to expose a set of port types (operations) on the defined transport.

```
<service name="DocInfo">
<port name="DocInfoSoap" binding="s0:DocInfoSoap">
<soap:address location="http://myhost.example.com:16200/_dav/cs/idcplg" />
</port>
</service>
```

Tip: You can add `&IsSoap=1` to the URL of a Content Server browser window to view the underlying SOAP code for that page.

31.5.2 Sample WSDL File

This sample code presents the complete DocInfo.wsdl file. This file and the CheckIn.wsdl, GetFile.wsdl, and Search.wsdl files are found in the *IntradocDir*/weblayout/groups/secure/wsdl/custom directory for the Content Server instance.

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://wwww.oracle.com/DocInfo/"
targetNamespace="http://wwww.oracle.com/DocInfo/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<s:schema elementFormDefault="qualified" targetNamespace="http://www.oracle.com/DocInfo/">
<s:element name="DocInfoByID">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="extraProps" type="s0:IdcPropertyList" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="DocInfoByIDResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="DocInfoByIDResult" type="s0:DocInfoByIDResult"
/>
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="DocInfoByIDResult">
<s:sequence>
<s:element minOccurs="0" maxOccurs="unbounded" name="ContentInfo" type="s0:ContentInfo" />
<s:element minOccurs="0" maxOccurs="unbounded" name="Revisions" type="s0:Revisions" />
<s:element minOccurs="0" maxOccurs="unbounded" name="WorkflowInfo" type="s0:WorkflowInfo" />
<s:element minOccurs="0" maxOccurs="1" name="StatusInfo" type="s0:StatusInfo" />
</s:sequence>
</s:complexType>
<s:element name="DocInfoByName">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dDocName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="extraProps" type="s0:IdcPropertyList" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="DocInfoByNameResponse">
```

```
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="DocInfoByNameResult"
type="s0:DocInfoByNameResult" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="DocInfoByNameResult">
<s:sequence>
<s:element minOccurs="0" maxOccurs="unbounded" name="ContentInfo" type="s0:ContentInfo" />
<s:element minOccurs="0" maxOccurs="unbounded" name="Revisions" type="s0:Revisions" />
<s:element minOccurs="0" maxOccurs="unbounded" name="WorkflowInfo" type="s0:WorkflowInfo" />
<s:element minOccurs="0" maxOccurs="1" name="StatusInfo" type="s0:StatusInfo" />
</s:sequence>
</s:complexType>
<s:complexType name="ContentInfo">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dDocName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dDocTitle" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dDocType" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dDocAuthor" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dSecurityGroup" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dDocAccount" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dRevClassID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dRevisionID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dRevLabel" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dIsCheckedOut" type="s:boolean" />
<s:element minOccurs="0" maxOccurs="1" name="dCheckoutUser" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dCreateDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dInDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dOutDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dStatus" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dReleaseState" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dFlag1" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWebExtension" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dProcessingState" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dMessage" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dReleaseDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dRendition1" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dRendition2" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dIndexerState" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dPublishType" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dPublishState" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dDocID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dIsPrimary" type="s:boolean" />
<s:element minOccurs="0" maxOccurs="1" name="dIsWebFormat" type="s:boolean" />
<s:element minOccurs="0" maxOccurs="1" name="dLocation" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dOriginalName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dFormat" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dExtension" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dFileSize" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="CustomDocMetaData" type="s0:IdcPropertyList" />
</s:sequence>
</s:complexType>
<s:complexType name="Revisions">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dFormat" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dInDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dOutDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dStatus" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dProcessingState" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dRevLabel" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dDocName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dRevisionID" type="s:int" />
```

```
</s:sequence>
</s:complexType>
<s:complexType name="WorkflowInfo">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dWfID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dDocName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfDocState" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfComputed" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfCurrentStepID" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="dWfDirectory" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dClbraName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfName" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfDescription" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dCompletionDate" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dSecurityGroup" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfStatus" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dWfType" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dProjectID" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="dIsCollaboration" type="s:boolean" />
</s:sequence>
</s:complexType>
<s:complexType name="StatusInfo">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="statusCode" type="s:int" />
<s:element minOccurs="0" maxOccurs="1" name="statusMessage" type="s:string" />
</s:sequence>
</s:complexType>
<s:complexType name="IdcPropertyList">
<s:sequence>
<s:element minOccurs="0" maxOccurs="unbounded" name="property" type="s0:IdcProperty" />
</s:sequence>
</s:complexType>
<s:complexType name="IdcProperty">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="name" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="value" type="s:string" />
</s:sequence>
</s:complexType>
</s:schema>
</types>
<message name="DocInfoByIDSoapIn">
<part name="parameters" element="s0:DocInfoByID" />
</message>
<message name="DocInfoByIDSoapOut">
<part name="parameters" element="s0:DocInfoByIDResponse" />
</message>
<message name="DocInfoByNameSoapIn">
<part name="parameters" element="s0:DocInfoByName" />
</message>
<message name="DocInfoByNameSoapOut">
<part name="parameters" element="s0:DocInfoByNameResponse" />
</message>
<portType name="DocInfoSoap">
<operation name="DocInfoByID">
<input message="s0:DocInfoByIDSoapIn" />
<output message="s0:DocInfoByIDSoapOut" />
</operation>
<operation name="DocInfoByName">
<input message="s0:DocInfoByNameSoapIn" />
<output message="s0:DocInfoByNameSoapOut" />
</operation>
</portType>
<binding name="DocInfoSoap" type="s0:DocInfoSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="DocInfoByID">
<soap:operation soapAction="http://www.oracle.com/DocInfo/" style="document" />
```

```
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
<operation name="DocInfoByName">
<soap:operation soapAction="http://www.oracle.com/DocInfo/" style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
</binding>
<service name="DocInfo">
<port name="DocInfoSoap" binding="s0:DocInfoSoap">
<soap:address location="http://myhost.example.com:16200/_dav/cs/idcplg/idc_cgi_isapi.dll" />
</port>
</service>
</definitions>
```

31.5.3 Generating WSDL Files

When the WSDL Generator component is installed and enabled during Oracle WebCenter Content installation, several folders and related HDA files are generated that expose several services as web services. Two directories are created in the *IntradocDir*/data/soap directory. The generic directory contains a generic.hda file, and the custom directory contains a wsdl_custom.hda file. Administrators can customize or add WSDL files using the Soap Wsdl administration pages. These pages are accessed by clicking the **Soap WSDL** link from the Administration section of the Admin Applet page.

Note: The WSDL Generator component must be enabled to generate WSDL files.

For step-by-step instructions on creating and editing a custom WSDL using the Soap Custom Wsdl administration pages, see Section 31.6, "Customizing WSDL Files."

31.5.4 Generating Proxy Class from WSDL Files

Using the WSDL files, developers may choose to create proxy classes to plug into a development tool. A number of software products and tool kits are available for converting WSDL files to programming class files in languages such as Java, Visual Basic, and C#. For example, Apache AXIS provides a SOAP to Java toolkit, and Microsoft .NET Development Environment provides functionality to convert WSDL files to C#.

If you are using Microsoft .NET, you can use utilitywsdl.exe to generate the proxy classes:

```
wsdl /l:CS DocInfo.wsdl
```

This utility generates the file DocInfoService.cs (C# class) which contains the class DocInfoService and the function DocInfo with the parameters specified. The return value is the DocInfoSet class, which is all the response parameters specified, along with ErrorCode and ErrorMessage values. If the ErrorCode is less than zero, an error

has occurred in the service call, and you can see the specifics of it in the value of ErrorMessage.

> **Note:** In addition to the WSDL files provided with the WSDL Generator component, you can generate WSDL files for any WebCenter Content service. For more information, see Section 31.5.3, "Generating WSDL Files."

31.6 Customizing WSDL Files

The Soap Custom Wsdl administration pages provide an administrator with the ability to edit and customize WSDL files. This chapter provides an administrative tutorial that gives step-by-step instructions on creating and editing a custom WSDL.

The WSDL Generator component must be enabled to generate WSDL files. In addition to the WSDL files provided with the WSDL Generator component, you can generate additional WSDL files for any WebCenter Content service. See Section 31.5.3, "Generating WSDL Files," for additional information.

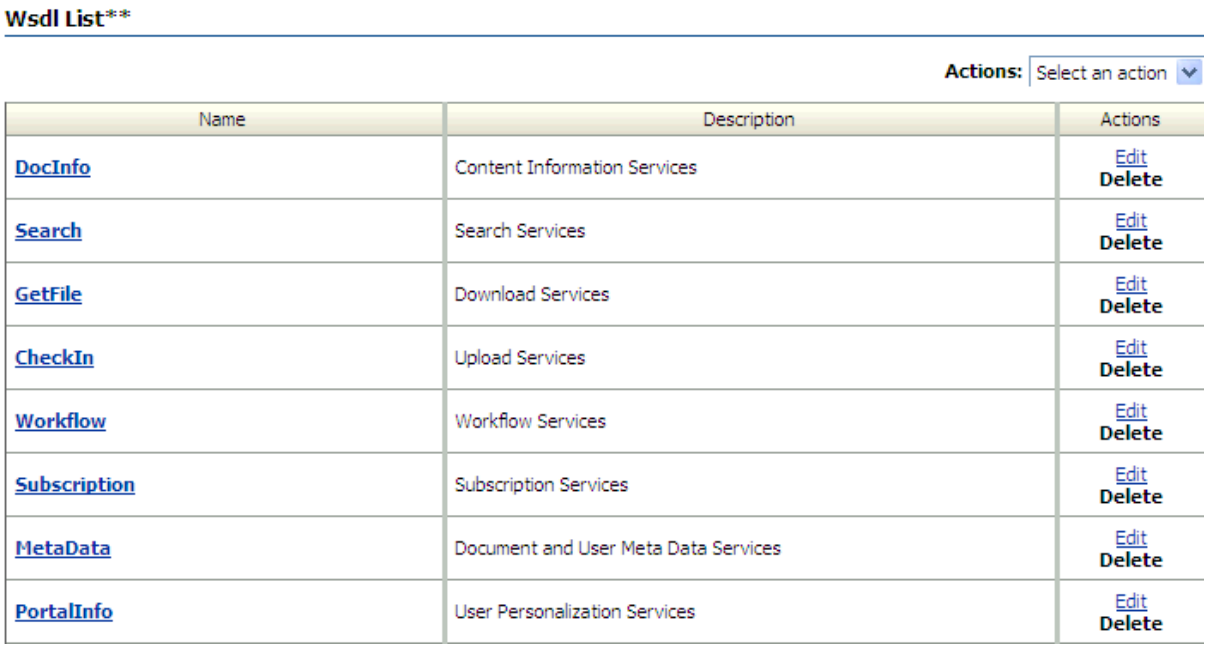
For a list of available services and the required parameters, see the *Oracle Fusion Middleware Services Reference for Oracle WebCenter Content*.

To create and edit a custom WSDL file with the Soap Custom WSDL administration pages:

1. In a web browser, log in to Oracle WebCenter Content Server as an administrator.
2. From the **Administration** tray or menu, choose **Soap Wsdls**.

   This option displays the Wsdl List page, which Figure 31–3 shows.

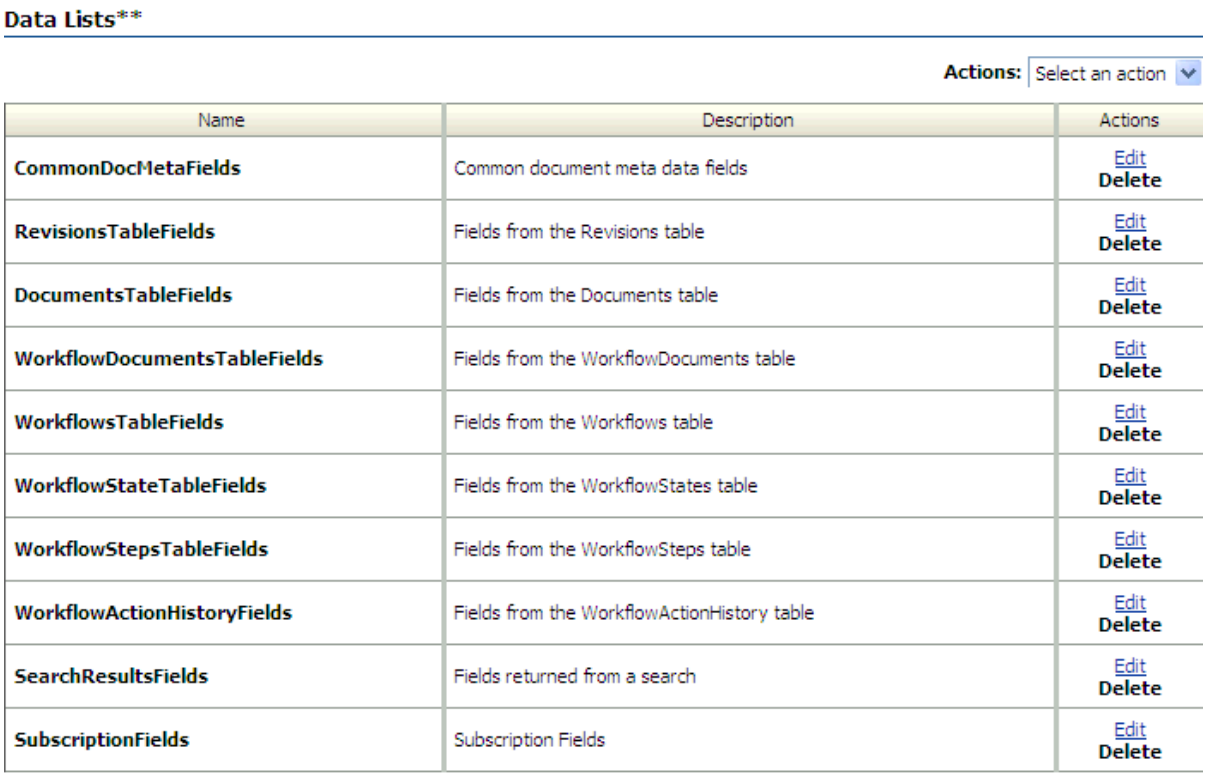
Figure 31–3 Wsdl List Page



3. From the **Actions** menu, choose **Data Lists**.

   This option displays the Data Lists page, which Figure 31–4 shows.

Figure 31–4 Data Lists Page



Data Lists**

Actions: Select an action

Name	Description	Actions
CommonDocMetaFields	Common document meta data fields	Edit Delete
RevisionsTableFields	Fields from the Revisions table	Edit Delete
DocumentsTableFields	Fields from the Documents table	Edit Delete
WorkflowDocumentsTableFields	Fields from the WorkflowDocuments table	Edit Delete
WorkflowsTableFields	Fields from the Workflows table	Edit Delete
WorkflowStateTableFields	Fields from the WorkflowStates table	Edit Delete
WorkflowStepsTableFields	Fields from the WorkflowSteps table	Edit Delete
WorkflowActionHistoryFields	Fields from the WorkflowActionHistory table	Edit Delete
SearchResultsFields	Fields returned from a search	Edit Delete
SubscriptionFields	Subscription Fields	Edit Delete

Note: System-specific WSDLs cannot be deleted. You can, however, edit the WSDL and enable or disable the complex type elements for that WSDL.

Data Lists are global lists of data that can be used with complex types, service parameters, or other Data Lists. When a Data List is specified as a parameter or a subtype of a complex type, all the subtypes of the Data List will appear as data types. Data Lists are defined once but can be referenced multiple times with different WSDLs and services. All the Data Lists have a prefix of "d:" in the data type list.

4. Choose **Add Data List** from the **Actions** menu.

   The Add Data List page is displayed.

5. Enter the following information:

   Name: `UserMetaFields`

   Description: `User Metadata Fields`

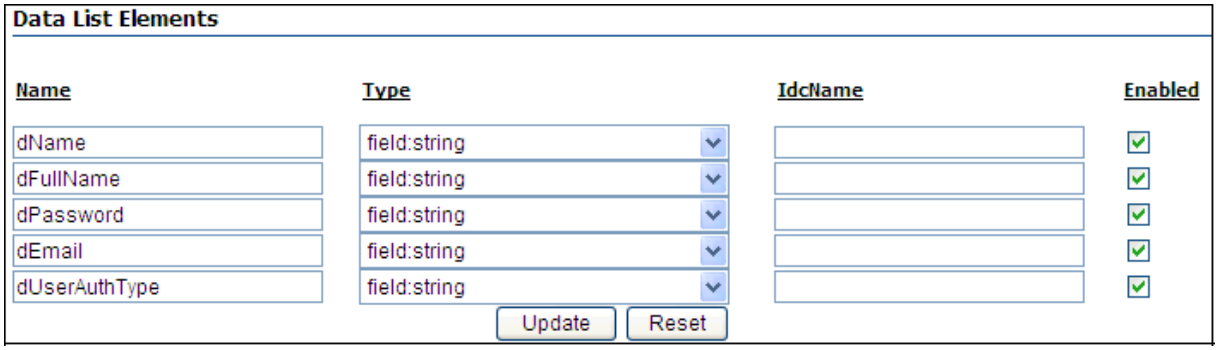
6. Click **Add**.

7. In the Data List Elements **Name** column, enter the following names for user metadata fields:

   - `dName`
   - `dFullName`
   - `dPassword`
   - `dEmail`
   - `dUserAuthType`

   For each name, choose `field:string` from the menu in the **Type** column, and make sure **Enabled** is selected, as Figure 31–5 shows.

Figure 31–5 Data List Elements



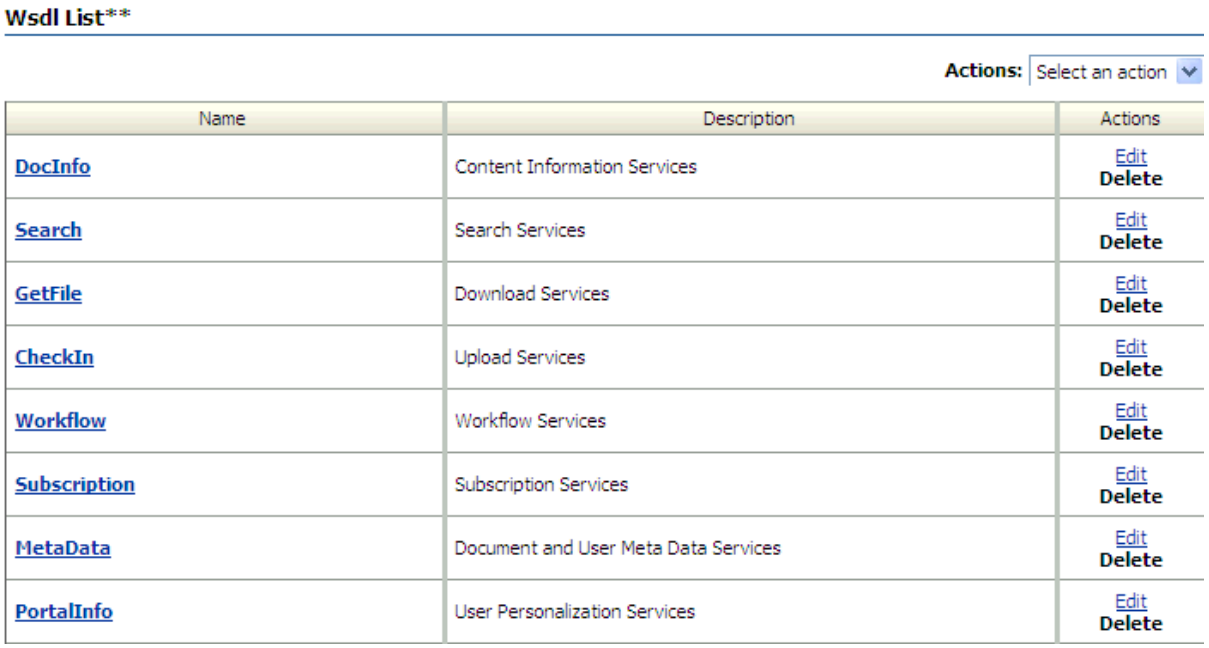
8. Click **Update**.

   You are returned to the updated Data Lists page. Note that `UserMetaFields` now appears at the bottom of the list.

9. Choose **Wsdl List** from the **Actions** menu.

   The Wsdl List page is displayed again, as Figure 31–6 shows.

Figure 31–6 Wsdl List Page Redisplayed



10. Choose **Add Wsdl** from the **Actions** menu.

    The Add Wsdl page is displayed.

11. Enter the following information:

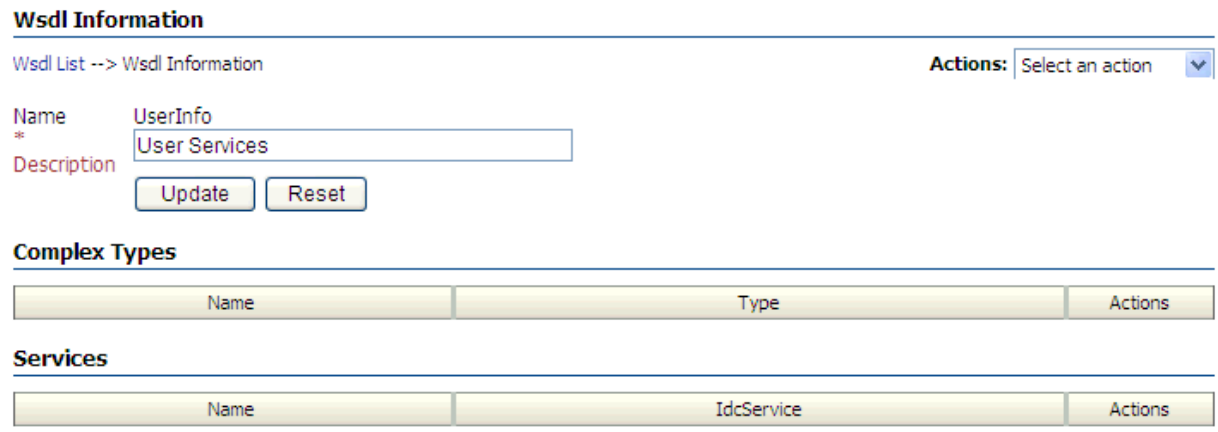
    Name: **UserInfo**

    Description: **User Services**

12. Click **Add**.

    The Wsdl Information page is displayed, as Figure 31–7 shows.

Figure 31–7 Wsdl Information Page



13. Choose **Add Complex Type** from the **Actions** menu.

    The Add Complex Type page is displayed.

> **Note:** Complex types contain other data types as subtypes. After these are created, any service in the WSDL can use these complex types as parameters.

14. Enter the following Complex Type information:

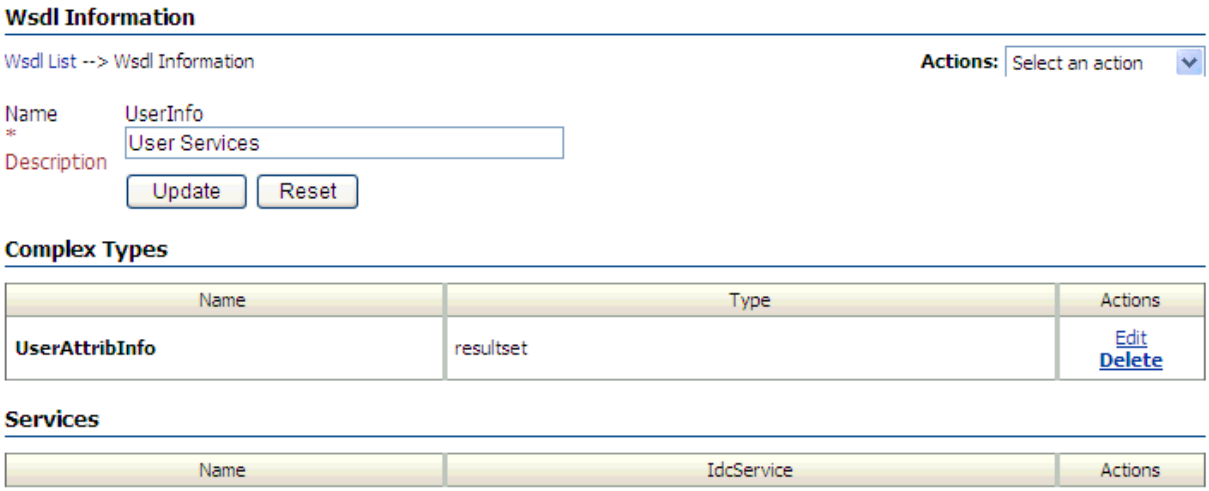
    Name: **UserAttribInfo**

    Type: Choose **resultset** from the menu.

15. Click **Add**.

    The Wsdl Information page is displayed again, as Figure 31–8 shows.

Figure 31–8 Wsdl Information Page Redisplayed



16. Click **Edit** on the **UserAttribInfo** line.

    The Complex Type Information/Complex Type Elements page opens.

17. Enter the following Complex Type Elements, and choose the **Type** value for each one from the menu.

Name	Type	Idc Name
`dUserName`	`field:string`	
`AttributeInfo`	`field:string`	

18. Click **Update** in the Complex Type Elements section.

    You are returned to the updated Wsdl Information page. Note that `User AttribInfo` now appears as a complex type.

19. Choose **Add Service** from the Actions menu.

    The Add Service page opens.

20. Enter the following information:

    Name: **AddUser**

    IdcService: **ADD_USER**

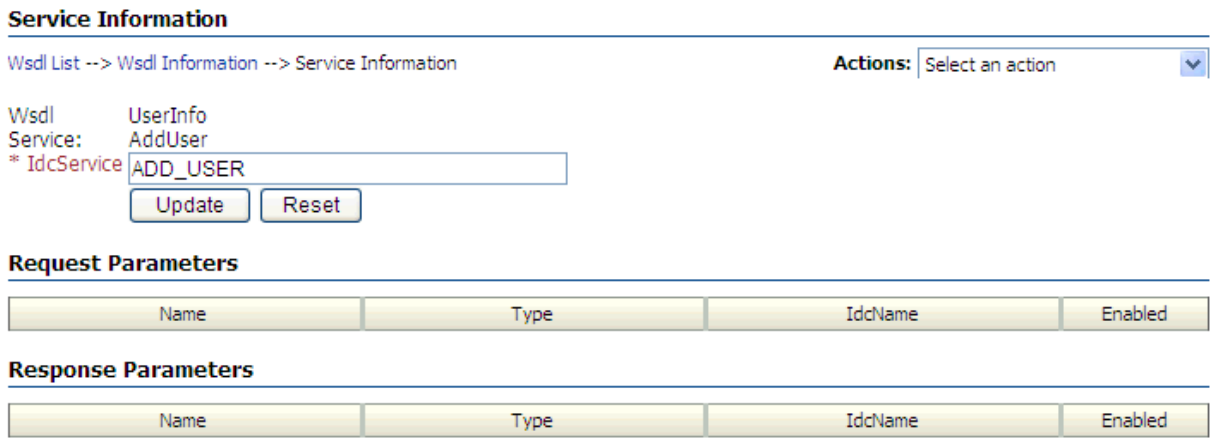
21. Click **Add**.

    The Wsdl Information page opens.

22. Choose **Edit** for the **AddUser** service.

    This option displays the Service Information page, which Figure 31–9 shows.

Figure 31–9 Service Information Page



Note: When you create a WSDL, you create services that correspond to the IdcServices feature of Content Server. You also specify the request and response parameters that you want the service to pass and receive from the Web Service call.

23. Choose **Update Request Parameters** from the **Actions** menu.

    The Request Parameters page is displayed.

24. Enter the following information, selecting the Type from the menu.

Name	Type	Idc Name
DataList	d:UserMetaFields	
CustomUserData	propertylist:CustomUserMeta	

25. Click **Update**.

    You are returned to the updated Service Information page. Note that `DataList` and `CustomUserData` now appear in the Request Parameters section.

26. Click **Update**.

    You are returned to the updated Wsdl Information page, showing the service that you just added.

27. Click **Update** again.

    You are returned to the updated Wsdl List page. UserInfo appears at the bottom of the list.

28. Choose **Generate Wsdls** from the **Actions** menu.

    A confirmation message displays after the Wsdls are generated successfully.

29. Click **Back**.

    You are returned to the Wsdl List page.

30. Click the **UserInfo** link in the Name column.

    The source code for the generated Wsdl file is displayed (a portion is shown in Example 31–12).

Example 31–12 Partial Source Code, Wsdl File

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmins:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.smlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://www.example.com/UserInfo/"
    targetNamespace="http://www.example.com/UserInfo/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified"
        targetNamespace="http://www.example.com/UserInfo/">
      - <s:element name="AddUser">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="dName"
                type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="dFullName"
                type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="dPassword"
                type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="dEmail"
                type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="dUserAuthType"
                type="s:string: />
              <s:element minOccurs="0" maxOccurs="1" name="CustomUserData"
                type="s0:IdcPropertyList" />
              <s:element minOccurs="0" maxOccurs="1" name="extraProps"
                type="s0:IdcPropertyList" />
            </s:sequence>
          </s:complexType>
        </s:element>
```

31. Click the browser **Back** button.

    You are returned to the Soap Custom Wsdl page.

Tip: You can right click **View** and save the WSDL file to your desktop (for use with .NET, and so on). However, be sure to save the file with the `.wsdl` file extension rather than the default `.xml` file extension.

32

Customizing the DesktopTag Component

This chapter describes how to customize the DesktopTag component of Oracle WebCenter Content Server to specify properties for checked out versions of Microsoft Word, Excel, and PowerPoint files.

This chapter includes the following sections:

32.1 About the DesktopTag Component

DesktopTag is a Content Server component that manages custom properties in files created using the default formats of Microsoft Office applications (2002 or later versions). The component adds custom properties to Word documents (DOC, DOCX, and DOT files), Excel spreadsheets (XLS, XLSX, and XLT files), and PowerPoint presentations (PPT and PPTX files) when they are checked out of Content Server, and removes this information when they are checked in again.

The properties to be added to the Microsoft Office files are specified in the DesktopTag configuration file. For more information, see Section 32.5, "Configuring the DesktopTag Component."

The custom properties provide information about where a content item resides in Content Server so that the file can be checked in to the right location, with the right content management parameters, and so on. This is particularly useful if the content item is processed outside of Content Server after check-out; for example, in an external workflow (that is, one that is not managed by Content Server). Also, the information can be exposed to users; for example, in the task area of Microsoft Office applications.

DesktopTag uses the Oracle Clean Content technology to add custom properties to and remove them from Microsoft Office files.

32.2 Enabling the DesktopTag and OracleCleanContent Components

The DesktopTag component is included with Content Server 11*g*R1. It must be enabled on Content Server because it is not enabled by default. The DesktopTag component requires that the OracleCleanContent component is enabled as well. The OracleCleanContent component is enabled with typical Content Server installations.

You can enable components using Component Manager, which is launched from the Content Admin Server page. For more information about enabling components, see "Enabling and Disabling a Component" in *Oracle Fusion Middleware Administering Oracle WebCenter Content*.

DesktopTag can add custom properties to the following Microsoft Office applications:

- Microsoft Word 2002 (XP) and later versions
- Microsoft Excel 2002 (XP) and later versions
- Microsoft PowerPoint 2002 (XP) and later versions

32.3 Checking Out and Checking In Content Items with DesktopTag

The DesktopTag component modifies the check-out (*file get*) and check-in operations for Content Server.

32.3.1 File Get Operation

The DesktopTag component installs a service handler override for the `createFileName` method, which should be called for all file get operations that go through the server (native URL requests do not call this method). If the file type is supported by the configuration, a set of custom properties are added to the file. These custom properties are used in various ways by the DesktopIntegrationSuite component and are made available to other components.

32.3.2 File Check-In Operation

The DesktopTag component installs an extension filter that hooks the validateCheckinData filter, which is part of the DesktopIntegrationSuite component. It removes the custom properties that were added by a file get operation before the data is checked in to the server.

The ResultSet returned for this operation includes the properties that would be added to the Microsoft Office file in a subsequent file get operation. This is provided to allow the client to modify the file rather than having to get a new copy. This method calls the desktopTagGetFilter extension filter, just like the file get operation.

32.4 Adding Properties to Checked-Out Content Items

The functionality offered by the DesktopTag component is provided entirely in the background. There is no direct user interaction. It is typically used for content tracking purposes, although the information can be exposed to users.

The properties that are added to the Microsoft Office files depend on the settings in the DesktopTag configuration file (see Section 32.5, "Configuring the DesktopTag Component"). In Figure 32–2, the content ID (`dDocName`), user name (`dUser`), and unique content item identifier (`dID`) are added to the Word document. The `DISProperties` custom property is always added. It lists all custom properties added by DesktopTag (as specified in the configuration file), and is used to ensure that the correct custom properties are deleted when a file is checked into Content Server again.

Figure 32–1 shows an example of a Word 2003 document without any custom properties that DesktopTag would add.

Figure 32–1 Word 2003 Document Without Custom Properties Added by DesktopTag

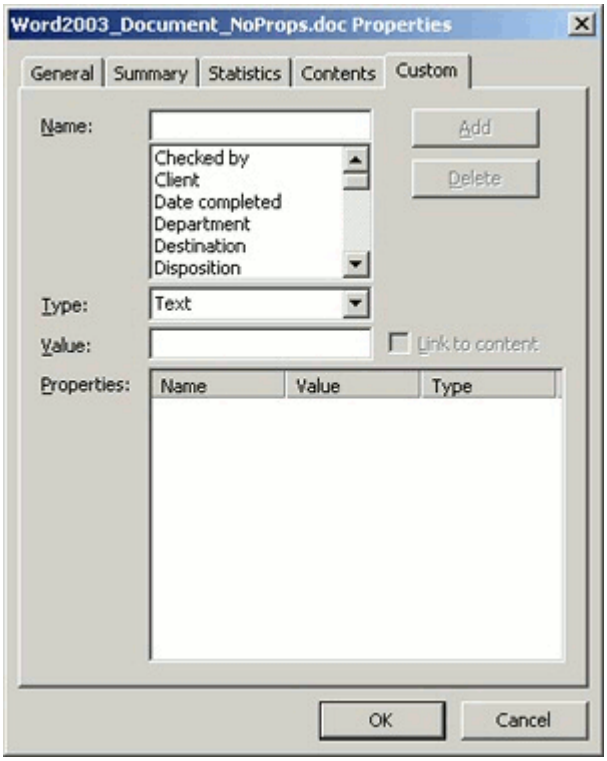
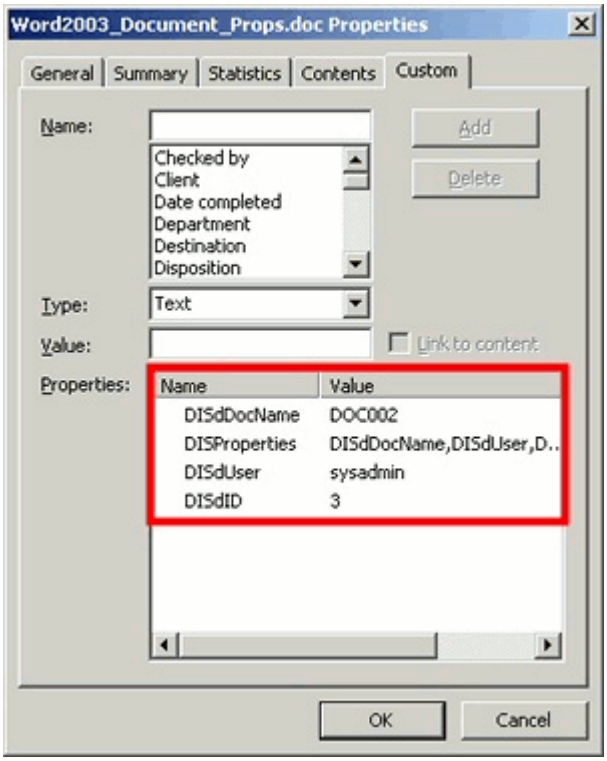


Figure 32–2 shows a number of custom properties added to a Word 2003 document.

Figure 32–2 Word 2003 Document with Custom Properties Added by DesktopTag



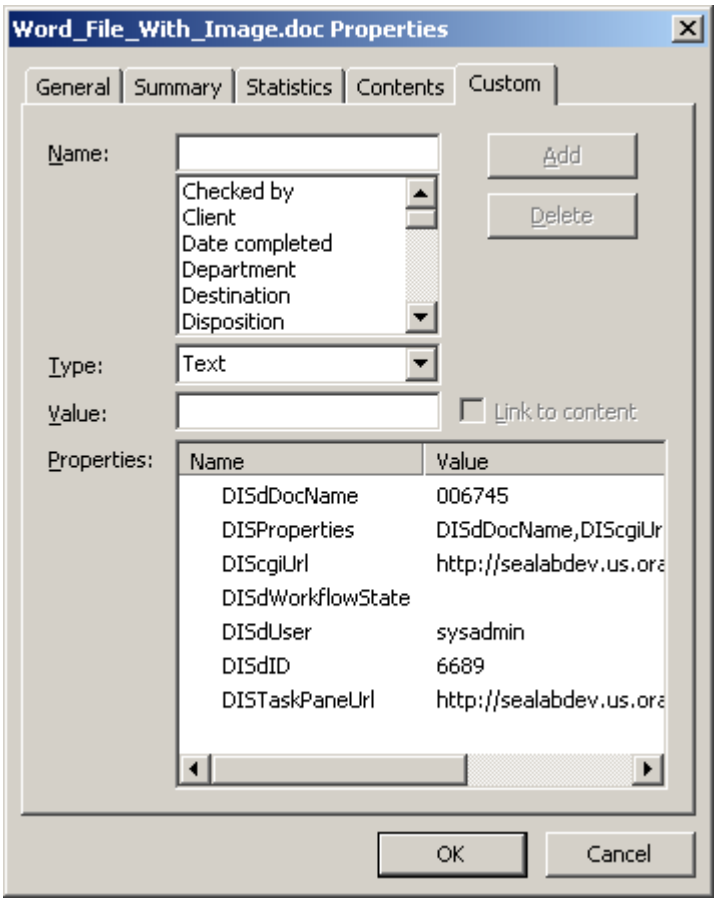
32.4.1 Viewing Custom Properties

Users can view the custom properties of a Microsoft Office file as follows:

- **Microsoft Office XP (2002) and 2003:** Choose **File**, then **Properties**, and then click the **Custom** tab.
- **Microsoft Office 2007:** Click the **Office** button in the application, then choose **Prepare**, then **Properties**, then **Document Properties**, then **Advanced Properties**, and then click the **Custom** tab.
- **Microsoft Office 2010:** Open the **File** panel, then click **Info**, then **Properties**, then **Advanced Properties**, and then click the **Custom** tab.

Figure 32–3 shows the custom properties that DesktopTag has added to a Word 2003 document.

Figure 32–3 Custom Document Properties (Microsoft Word 2003)



Screenshot of the Custom tab of the file properties dialog in Microsoft Word 2003, which shows the custom properties added to the file by Oracle WebCenter Content: Desktop. The custom properties are explained in the text surrounding this image.

****

32.4.2 Checking In Documents from Outside Content Server

These custom document properties enable Oracle Webcenter Content: Desktop to keep track of where a managed file resides in Content Server. This, in turn, enables users to check a Microsoft Office document back in to Content Server even from outside a content management integration context.

To check in a document, the user must have a connection to the server set up. The Office add-in looks at the *CGI URL*, *Server*, and *IDCNAME* properties to try to match the document to a server, so the user must be on the same network and have access to the server.

This feature can be useful in a number of situations; for example:

- A user receives a managed Word document from someone else, as an attachment to an e-mail.
- A user copies a managed Word document from a server in the integration hierarchy to a folder outside that hierarchy.

In either case, users can open the file in Microsoft Word, make changes, and then check the file back in to the server using the Oracle WebCenter Content menu or ribbon in Word. Desktop checks the custom properties embedded in the Word document to find out where to upload the file to.

32.5 Configuring the DesktopTag Component

The DesktopTag component is configured using a configuration file, desktoptag_environment.cfg, which is located in the component installation directory. This is a plain-text file that you can edit in any text editor. The component installation directory is *MW_HOME*/*ECM_ORACLE_HOME*/ucm/idc/components/DesktopTag.

> **Note:** Make sure that you restart Content Server after making changes to the DesktopTag configuration file.

The following properties can be set in the configuration file:

- `DesktopTagFormats`
- `DesktopTagPrefix`
- `DesktopTagFields`
- `DesktopTagPrefixCustom`
- `DesktopTagFieldsCustom`
- `DesktopTagPrefixExtended`
- `DesktopTagFieldsExtended`
- `DefaultTaskPaneUrl`
- `DesktopTagLog`
- `DesktopTagFormatsExclude`

32.5.1 DesktopTagFormats Property

The value of the `DesktopTagFormats` property is a comma-separated list of MIME data types that are processed for tagging. If the data type is not in the list, it is not processed. If this parameter is commented out (using `#`), empty, or not included in the configuration file at all, then all supported data types are processed.

Example:
`DesktopTagFormats=application/msword,application/ms-excel`

If you include a nonsupported MIME data type in the list, DesktopTag will attempt to process the file, and an error event is included in the log file if logging is enabled.

32.5.2 DesktopTagPrefix Property

The value of the `DesktopTagPrefix` property is the prefix added to the names of all standard Content Server metadata fields in the list of standard DesktopTag fields (see Section 32.5.3, "DesktopTagFields Property"). This prefix is not added if a specific property name is defined.

If this parameter is commented out (using `#`), empty, or not included in the configuration file at all, then `DIS` is used as the default.

Example: `DesktopTagPrefix=STD`

32.5.3 DesktopTagFields Property

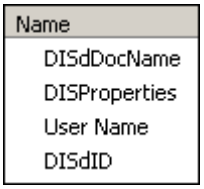
The value of the `DesktopTagFields` property is a comma-separated list of all standard Content Server metadata fields that are added to Microsoft Office files as custom properties. You should use the server-internal field names (for example, `dDocName` for the content ID). For information about the internal field names of the standard metadata field, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

You can set a specific property name for a metadata field by adding it in parentheses after the field name. This is especially useful if the property name will be exposed to end users (for example, in the task area in Microsoft Office 2007 applications).

Example: `DesktopTagFields=dID,dDocName,dUser(User Name)`

Figure 32–4 shows the result of the preceding `DesktopTagFields` definition (assuming the default `DIS` prefix is used).

Figure 32–4 Example of Property Names



Note: The `DISProperties` custom property is always added. Its value is a list of all properties added by DesktopTag.

32.5.4 DesktopTagPrefixCustom Property

The value of the `DesktopTagPrefixCustom` property is the prefix added to the names of all custom Content Server metadata fields in the list of custom DesktopTag fields (see Section 32.5.4, "DesktopTagPrefixCustom Property"). This prefix is not added if a specific property name is defined.

If this parameter is commented out (using `#`), empty, or not included in the configuration file at all, then `DISC` is used as the default.

Example: `DesktopTagPrefixCustom=CST`

32.5.5 DesktopTagFieldsCustom Property

The value of the `DesktopTagFieldsCustom` property is a comma-separated list of all custom Content Server metadata fields that will be added to Microsoft Office files as custom properties. You define these fields in exactly the same manner as standard metadata fields (see Section 32.5.3, "DesktopTagFields Property").

Example: `DesktopTagFieldsCustom=xComments(Extra Info),xArchiveStatus`

Note: The standard and custom Content Server metadata fields are processed exactly the same by DesktopTag. The separate configuration entries are there only to make it easier to distinguish between these fields.

32.5.6 DesktopTagPrefixExtended Property

The value of the `DesktopTagPrefixExtended` property is the prefix added to the names of all custom Content Server metadata fields in the list of extended DesktopTag fields (see Section 32.5.7, "DesktopTagFieldsExtended Property"). This prefix is not added if a specific property name is defined.

If this parameter is commented out (using `#`), empty, or not included in the configuration file at all, then `DISX` is used as the default.

Example: `DesktopTagPrefixExtended=EXT`

32.5.7 DesktopTagFieldsExtended Property

The value of the `DesktopTagFieldsExtended` property is a comma-separated list of property definitions that come from the ExtendedUserAttributes component. The general form of a property definition is *`type`*`/`*`key`*`/`*`subkey`*`(`*`name`*`)`. The *`type`*, *`key`*, and *`subkey`* values are the parameters used by the EC_GET_PROPERTY service. If any of these values begins with the character `@`, then the parameter value is taken from the specified Content Server metadata field (see the following example).

You can set a specific property name for a metadata field by adding it in parentheses after the field name.

Example: `DesktopTagFieldsExtended=account/@dSecurityGroup/WCTPUrl(DIS_Task_Pane_Url)`

This example specifies that the property will be named `DIS_Task_Pane_Url`, and its value will be the `ExtendedUserAttributes` item with the type `account`, the key value specified by the `dSecurityGroup` metadata field (the security group of the content item), and the subKey `WCTPUrl`.

32.5.8 DefaultTaskPaneUrl Property

The value of the `DefaultTaskPaneUrl` property is a string that defines the default URL to use in setting the `DISTaskPaneUrl` property, which is required to display a web page for a file in the task area of Microsoft Office applications. Any words beginning with the character `@` are replaced by the values from the binder or by other means (currently, this applies only to `@cgiUrl`).

Example: `DefaultTaskPaneUrl=@cgiUrl?IdcService=GET_TASK_PANE &dID=@dID`

In this example, `@cgiUrl` would be replaced by the Content Server Cgi URL value, and `@dID` would be replaced by the value of the server-internal, unique content item identifier (`dID`).

As another example, if there is an extended user attribute called `WebCenterUrl`, then adding the string `"WebCenterUrl(DISTaskPaneUrl)"` will set the `DISTaskPaneUrl` property to the value of the extended user attribute called `WebCenterUrl`.

32.5.9 DesktopTagLog Property

The value of the `DesktopTagLog` property is a Boolean value that indicates whether or not to log the operations and results of the DesktopTag component (`1` = yes, `0` = no).

If this parameter is commented out (using `#`), empty, or not included in the configuration file at all, then the component operations and results are not logged. The DesktopTag log information is included in the standard Content Server log files (accessible from the server's administration pages), as Figure 32–5 shows.

Figure 32–5 DesktopTag Event in Content Server Log File

Info	1/23/08 1:36 PM	Document 'Word2003_Document.doc', format 'application/msword': Successfully tagged

32.5.10 DesktopTagFormatsExclude Property

The value of the `DesktopTagFormatsExclude` property is a comma-separated list of MIME data types that are not processed for tagging. If the data type is not in the list, it is processed.

Example: `DesktopTagFormatsExclude=application/ms-excel`

There is no reason to use both `DesktopTagFormats` and `DesktopTagFormatsExclude`.

Part VIII

Appendices

This part includes appendices that provide additional information.

Part VIII contains the following appendix:

- Appendix A, "Idoc Script Functions and Variables"
- Appendix B, "Building a Website"
- Appendix C, "Troubleshooting"

A

Idoc Script Functions and Variables

This appendix provides information about Idoc Script functions and variables, which you can use for customizing Oracle WebCenter Content.

Idoc Script has many built-in global functions. Functions perform actions, including string comparison and manipulation routines, date formatting, and ResultSet manipulation. Some functions also return results, such as the results of calculations or comparisons.

Information is passed to functions by enclosing the information in parentheses after the name of the function. Pieces of information that are passed to a function are called parameters. Some functions do not take parameters; some functions take one parameter; some take several. There are also functions for which the number of parameters depends on how the function is being used.

In addition to the built-in global functions, you can define new global functions, including custom classes, with Java code. For more information, see Chapter 11, "Getting Started with Content Server Components," and Chapter 17, "Creating Custom Components."

Along with built-in functions, Idoc Script uses a range of variables. Variables which are used within Idoc scripts include dynamic variables, conditional dynamic variables, and page display variables. Many of these variables can be used both within scripts and specified individually in the WebCenter Content `config.cfg` and `intradoc.cfg` files, or used in a web browser URL.

For details about variables that are used only in `.cfg` files or in a web browser URL, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

abortToErrorPage()

Aborts the current page and displays an error message.

- This function evaluates the StatusCode variable, and if a negative numeric value (`-1`) is returned, substitutes the display of the current page with an error page.
- The StatusMessage variable can be used as the error message string.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.1.5.1, "Page Display Variables"

Parameters

The only parameter is the error message string.

Output

Returns the error message string on an error page.

Example

Aborts the current page and displays `Access Denied` as an error message:

```
<$abortToErrorPage("Access Denied")$>
```

Aborts the current page and displays the value of the StatusMessage variable as an error message:

```
<$abortToErrorPage("<$StatusMessage$>")$>
```

See Also

- executeService()
- IsRequestError
- StatusCode
- StatusMessage

addEmptyOption

Specifies that a metadata field option list has blank value as the first option in the list.

Type and Usage

- Section 4.1.5.2, "Field Display Variables"
- Section 4.1.5.2.3, "Other Field Display Variables"

Output

- Returns TRUE if the first value in the option is blank.
- Returns FALSE if the first value in the option list is not blank.

Example

Specifies that the first value in the option list is blank:

```
<$if ForceDocTypeChoice and isTrue(ForceDocTypeChoice)$>
    <$addEmptyOption = 1$>
<$endif$>
```

See Also

- fieldIsOptionList

AdminAtLeastOneGroup

Checks if the current user has the *admin* role for at least one security group.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user is an administrator for at least one security group.
- Returns FALSE if the user is not assigned the *admin* role.

Example

Can be used to do an optional presentation for an administrator:

```
<$if (AdminAtLeastOneGroup)$>
    <a href="<$redirect$>">
<$endif$>
```

See Also

- UserAppRights
- UserIsAdmin

AfterLogin

Specifies whether the current page was created immediately after a login.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.1.5.1, "Page Display Variables"

Output

- Returns TRUE if the page was created immediately after a login.
- Returns FALSE if the page was not created immediately after a login.

Example

Displays an alternate URL if the page was not created immediately after a login:

```
<$if not AfterLogin$>
    <a href="<$redirect$>">
<$endif$>
```

AllowCheckin

Checks if the current user has checkin permission for the content item's security group.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user has checkin permission.
- Returns FALSE if the user does not have checkin permission.

Example

Can be used to do an optional presentation for a user with checkin permission:

```
<$if (AllowCheckin)$>
    <a href="<$redirect$>">
<$endif$>
```

AllowCheckout

Checks whether current user has checkout permission for the content item's security group.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user has checkout permission.
- Returns FALSE if the user does not have checkout permission.

Example

Can be used to do an optional presentation for a user with checkout permission:

```
<$if (AllowCheckout)$>
    <a href="<$redirect$>">
<$endif$>
```

AllowReview

Checks if the current user is allowed to approve or reject the current workflow item.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.1.8, "Workflows"

Output

- Returns TRUE if the user is a reviewer for the current workflow step.
- Returns FALSE if the user is not a reviewer for the current workflow step.

Example

Displays Approve and Reject buttons if the user is a reviewer:

```
<$if AllowReview$>
    <$include workflow_doc_action_buttons$>
<$endif$>
```

AuthorAddress

Specifies the e-mail address of the author of a content item.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.24, "Users"

Output

Returns a string or Boolean value depending on use.

- Standard use: Returns the e-mail address of the content item's author as a string.
- Used in a conditional statement:
  - Returns TRUE if the content item author has a defined e-mail address.
  - Returns FALSE if the content item author has no e-mail address.

Example

Can be used to alert the content item author through e-mail when a revision is made.

```
<$AuthorAddress$>
```

AuthorDelete

Enables authors of content items to delete their own revisions without having the Delete privilege for the security group.

- When set to `TRUE` and Content Server is configured to use Folders (enabled by the `FrameworkFolders` component), authors can delete their own revisions as long as they have the Read privilege, where normally they would need the Delete privilege for the security group.
- When set to `TRUE` and Content Server is configured to use Contribution Folders (enabled by the `Folders_g` component), authors can delete their own revisions as long as they have the Read privilege, where normally they would need the Delete privilege for the security group.
- Default is an empty string.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.21.1, "Internal Security"

Location

- System Properties, Content Security tab, Allow author to delete revision
- Admin Server, Content Security option, Allow author to delete revision
- *IntradocDir*`/config/config.cfg`

Example

As a configuration entry:

```
AuthorDelete=true
```

As Idoc Script:

```
<$if AuthorDelete$>
    <$AuthorDelete$>
<$else$>
    false
<$endif$>
```

AutoNumberPrefix

Defines the prefix that will be added to all automatically numbered Content IDs.

- Returns the automatic numbering prefix (returns value in configuration settings).
- Returns a string.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.3, "Content Items"

Location

- System Properties, Options tab, Auto Name Prefix
- Admin Server, General Configuration, Auto Number Prefix
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting, defines the automatic numbering prefix:

```
AutoNumberPrefix=HR
```

As Idoc Script, returns the value of the configuration setting:

```
<$AutoNumberPrefix$>
```

See Also

- "IsAutoNumber" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

BatchLoaderPath

Defines the path to the default batch load text file.

Returns the file path as a string.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.1, "Batch Loader"

Location

DomainHome/ucm/cs/bin/intradoc.cfg

Example

- As a Windows configuration entry:

  ```
  BatchLoaderPath=c:/domain/BatchLoader/batchfile.txt
  ```

- As a Solaris/UNIX configuration entry:

  ```
  BatchLoaderPath=/u1/intradoc3/batLd/batchfile
  ```

- As Idoc Script, returns the file path as a string:

  ```
  <$BatchLoaderPath$>
  ```

break()

Often used to terminate a loop.

- The break instruction causes the innermost loop to be exited.
- Control resumes with the first statement following the end of the loop.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

BrowserVersionNumber

Retrieves the version number of the client browser.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.2, "Clients"

Output

Returns the browser version number as a string.

Example

Can be used to ensure that the user has a browser version compatible with Content Server.

```
<$BrowserVersionNumber$>
```

C

Specifies a comment in Idoc Script code.

Type and Usage

- Section 3.2.2, "Idoc Script Comments"
- Section 4.2.13, "Idoc Script"

Example

Places a comment in the code:

```
<$c = "Sets the variables to empty strings."$>
```

cacheInclude()

This function acts similar to the `inc()` Idoc Script function. It will evaluate the dynamic html include corresponding to `includeName` and display it on the page. The difference is that if possible, it will pull the rendered html from a cache, instead of evaluating it again.

For more information, see Section 3.4.1, "Keywords Versus Functions."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

Takes three required parameters and two optional parameters. For example:

```
cacheInclude(includeName, scope, lifeSpan [, cacheName, key])
```

Parameter	Description
includeName	The name of the dynamic html element to evaluate.
scope	Set to 'session' if each user should see different html, or 'application' if all users will see the same thing.
lifeSpan	The lifespan of this include in the cache, in seconds.
cacheName	Optional: if you want to place this data into a named cache instead of the default cache. If an empty string is passed, it will cache the include into the default cache for the session.
key	Optional: if you do not want automatic name-scoping of your cache to prevent conflicts, you can specify a unique key here.

Example

This will cache the `std_page_begin` include for each user for ten minutes. This is about 10k per user in the cache.

```
<$cacheInclude("std_page_begin", "session", 600)$>
```

See Also

- inc()
- forceExpire()
- setExpires()
- setHttpHeader()
- setMaxAge()

captionEntryWidth

Specifies the width of a metadata field, in percent.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the width of the current metadata field in percent.

Example

Used as script:

```
<$if isInfo$>
    <$captionFieldWidth="30%"$>
    <$captionEntryWidth="70%"$>
<$elseif isEditMode$>
    <$captionFieldWidth="20%"$>
    <$captionEntryWidth="80%"$>
<$endif$>
```

See Also

- captionFieldWidth

captionFieldWidth

Specifies the width of a metadata field caption, in percent.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the width of the current metadata field caption in percent.

Example

Used as script:

```
<$if isInfo$>
    <$captionFieldWidth="30%"$>
    <$captionEntryWidth="70%"$>
<$elseif isEditMode$>
    <$captionFieldWidth="20%"$>
    <$captionEntryWidth="80%"$>
<$endif$>
```

See Also

- fieldCaptionStyle
- captionEntryWidth

clearSchemaData()

Clears the data from a schema ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

This function can take zero, one, or two parameters.

- If passed zero arguments, it clears the data binder. Returns no value.
- If passed one argument, the argument is the name of the ResultSet to clear the values from the current row. Returns no value.
- If passed two arguments, the first argument is the ResultSet name and the second argument is the key identifying the data object to clear. Returns `0` if the data does not exist or `1` if it does exist. The use is `True()` or `False()` to conditionally execute scripts based on the return value.

Example

```
<$clearSchemaData()$>
```

See Also

- loadSchemaData()

ClientControlled

Checks if the page was accessed from the ODMA Client.

This value is passed by the string for controlling the update process that is provided by the client.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.2, "Clients"

Output

- Returns TRUE if the page was accessed from the ODMA Client.
- Returns FALSE if the page was not accessed from the ODMA Client.

Example

Checks if the page was accessed from the ODMA Client:

```
<$ClientControlled$>
```

See Also

- HasLocalCopy
- IsNotLatestRev

computeDocUrl()

This function computes the URL to a content item based on the data on the page.

For example, this function can be called to generate a URL to an item when looping over a ResultSet of items.

The following information must be present on the page:

- `dDocAccount` (optional)
- `dDocName`
- `dDocType`
- `dProcessingState`
- `dRevLabel`
- `dSecurityGroup`
- `dWebExtension`

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.3, "Content Items"

Parameters

The only parameter indicates if the URL is relative, set to `TRUE` (`1`) or `FALSE` (`0`).

computeRenditionUrl()

Returns the URL of a given rendition.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.3, "Content Items"

Parameters

Takes three parameters:

- The first parameter is the URL of the content item.
- The second parameter is the `dRevLabel` value.
- The third parameter is the `dRendition1` value. Possible values of `dRendition1` include:
  - **T** = Thumbnail rendition
  - **X** = XML rendition

Output

Returns the complete URL of the rendition as a string.

Example

Returns the URL of the rendition as a string.

```
<$computeRenditionUrl(url, dRevLabel, dRendition1)$>
```

CONTENT_LENGTH

Retrieves the length in bytes of the requested content item as supplied by the client.

This variable is specific to the current gateway program request.

> **Important:** This setting is obsolete for Content Server version 7.0 and later. The web server filter no longer sends this information.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.25, "Web Servers"

Output

Returns the content length in bytes.

Example

As information output on a page or to a log:

```
CONTENT_LENGTH=0
```

As part of an Idoc Script statement or evaluation:

```
<$if CONTENT_LENGTH$>
<!--statement-->
```

coreContentOnly

Set this variable in the URL of a service request to display only the form. The `std_page_begin` and `std_page_end` include files will not be displayed.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.16, "Page Display"

Output

None.

CURRENT_DATE

Returns the current date and time.

This variable is similar to dateCurrent, which is used more frequently.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.8, "Date and Time"

Output

Returns a string formatted according to the user locale.

Example

Used as script:

```
<$CURRENT_DATE$>
```

See Also

- dateCurrent()

CURRENT_ROW

Evaluates which row of a ResultSet you are in.

The first row in a ResultSet is row zero (0).

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.18, "ResultSets"

Output

Returns the row number.

Example

Used as script:

```
<$CURRENT_ROW$>
```

dateCurrent()

Returns the current date and time.

Can be used to return the current date and time to the user or to create commands using date evaluations.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only optional parameter is an integer, which adjusts the date by the specified number of days relative to the current date.

Output

Returns a date formatted according to the user locale.

Example

In the following examples, dates are formatted according to the default English-US locale:

```
m/d/yy h:mm XM
```

The following returns the current date and the current time (for example, 8/12/01 1:55 PM):

```
<$dateCurrent()$>
```

Returns the date ten days in the future and the current time (for example, 8/22/01 1:55 PM):

```
<$dateCurrent(10)$>
```

Returns the date ten days in the past and the current time (for example, 8/2/01 1:55 PM):

```
<$dateCurrent(-10)$>
```

See Also

- CURRENT_DATE

dcShowExportLink

This function verifies if the Dynamic Converter has been configured to convert a content item. The returned value is based on the value for `dFormat` for the item.

This function is typically used on Search Result pages to conditionally display a Dynamic Converter link.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.6.2, "Dynamic Converter"

Output

- Returns `TRUE` if the Dynamic Converter is configured to convert the content item.
- Returns `FALSE` if the Dynamic Converter is not configured.

Example

```
<$QueryText = "dDocTitle <substring> `test`"$>
<$executeService("GET_SEARCH_RESULTS")$>
<$loop SearchResults$>
    <$if dcShowExportLink()$>
        <a href="<$HttpCgiPath$>?IdcService=
        GET_DYNAMIC_CONVERSION&dID=<$dID$>">
        HTML Conversion of <$dDocTitle$></a>
    <$endif$>
<$endloop$>
```

ddAppendIndexedColumnResultSet()

This function loads a dynamicdata table into a ResultSet. It is very similar to `ddLoadIndexedColumnResultSet`. The main difference is that if the Idoc Script ResultSet already exists, the new ResultSet created from the dynamicdata table is appended to it. Any fields found in the dynamicdata table, but not in the target ResultSet, are automatically added.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for this function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be appended.
`indexColumn`	The name of an indexed column in the dynamicdata table.
`indexValue`	The value to use to select a section of the original table. This value will be checked in a case-insensitive manner against the `indexColumn` value in each row. If it matches, that row is loaded into the ResultSet; otherwise, it is skipped.
`mappingTableName`	The name of a dynamicdata table used to rename the columns of the final ResultSet. The renaming is done by mapping the column names in the specified mapping table to the values of the first row in the same table. This is done exactly the same in `ddLoadResultSet`.

Output

If the dynamicdata table is successfully loaded, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
<?commatable indexedColumns="fieldA"?>
fieldA, fieldB, fieldC
1,      2,       3
4,      5,       6
1,      3,       9
<@end@>


<$exec rsCreateResultSet("MyResultSet", "fieldA,fieldB")$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.fieldA = "first value"$>
<$MyResultSet.fieldB = "second value"$>
<$exec ddLoadIndexedColumnResultSet("MyDataTable", "MyResultSet", "fieldA", "1")$>
<$exec rsFirst("MyResultSet")$>
<$foo = MyResultSet.fieldB$>    [[% (foo == 'second value') %]]<$exec
rsNext("MyResultSet")$>
<$bar = MyResultSet.fieldC$>    [[% (bar == '3') %]]
```

See Also

- ddLoadIndexedColumnResultSet()

ddAppendResultSet()

This function loads a dynamicdata table into a ResultSet. It is very similar to `ddLoadResultSet`. The main difference is that if the Idoc Script ResultSet already exists, the new ResultSet created from the dynamicdata table is appended to it. Any fields found in the dynamicdata table, but not in the target ResultSet, are automatically added.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be appended.
`mappingTableName`	The name of a dynamicdata table used to rename the columns of the final ResultSet. The renaming is done by mapping the column names in the specified mapping table to the values of the first row in the same table. This is done exactly the same in `ddLoadResultSet`.

Output

If the dynamicdata table is successfully loaded, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
field1, field2, field3
a,      b,       c
d,      e,       f
<@end@>


<$exec rsCreateResultSet("MyResultSet", "field1,field2")$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.field1 = "first value"$>
<$MyResultSet.field2 = "second value"$>
<$exec ddAppendResultSet("MyDataTable", "MyResultSet")$>
<$exec rsFirst("MyResultSet")$>
<$foo = MyResultSet.field1$>    [[% (foo == 'first value') %]]
<$exec rsNext("MyResultSet")$>
<$bar = MyResultSet.field3$>    [[% (bar == 'c') %]]
```

See Also

- ddLoadResultSet()
- ddAppendIndexedColumnResultSet()

ddApplyTableSortToResultSet()

This function sorts an existing ResultSet using the rules defined in a particular dynamicdata table.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to use for its sorting rules.
`resultSetName`	The name of the ResultSet into which the dynamicdata table should be appended.

Output

If the sorting is successfully performed, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
<?commatable sortcolumn="order" sortType="int" sortOrder="asc"?>
user,    service,            order
jane,    DOC_INFO,           10
bob,     GET_SEARCH_RESULTS, 20
annette, CHECKIN_NEW,        30
<@end@>


<$exec rsCreate("MyResultSet")$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.user = "james"$>
<$MyResultSet.service = "GET_FILE"$>
<$MyResultSet.order = 75$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.user = "zoe"$>
<$MyResultSet.service = "DOC_INFO_BY_NAME"$>
<$MyResultSet.order = 20$>
<$exec ddApplyTableSortToResultSet("MyDataTable", "MyResultSet")$>
```

ddGetFieldList()

This function takes a dynamicdata table and returns a comma-separated string containing the names of the columns in the table. It is expected that many dynamicdata tables consist only of field names without any rows just to supply comma-separated lists of values to code in the Content Server system.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

This function has one parameter, `dataTableName`, which is the name of the dynamicdata table to load.

Output

If the dynamictable exists, it returns True.

Example

```
<@dynamicdata MyDataTable@>
foo,bar,baz
<@end@>


<$fieldList = ddGetFieldList("MyDataTable")$>
<$trace(fieldList, "#console")$> [[% Outputs: foo,bar,baz %]]
```

ddIncludePreserveValues()

This function executes a resource include, but protects values specified by a dynamicdata table from being changed. The column names in the data table are used as the list of variables names that must be protected. These variables are protected by temporarily caching them, calling the include, and then resetting those variables back to the cached values. If a variable was null, it is set to blank.

If one of the column names in the table starts with a dollar symbol ($), then the string that follows is assumed to be the name of a ResultSet. In that case, it is the pointer to the ResultSet that is temporarily cached in memory and then replaced after the resource include has finished executing. If the ResultSet did not exist at the time of caching, then any ResultSet that exists with that key at the end are removed. If the ResultSet is active at the time it has its pointer cached, new ResultSets of the same name can be created during the call of the include and the previously active ResultSet will be recovered appropriately without disturbing the loop. The one side effect is that if a new ResultSet is created that temporarily replaces the active ResultSet, then the algorithm for variable substitution that retrieves values first from active ResultSets can find values for a variable from the cached active ResultSet (assuming the variable is not found as a field in an active ResultSet with higher precedence). The cached active ResultSet maintains its place in the active ResultSet stack,

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
includeName	The name of the resource include to execute.
dataTableName	The name of the dynamicdata table to use for preserving local data and ResultSets.

Output

A string representing the output of the executed resource include.

Example

```
<@dynamicdata MyPreservedFields@>
foo, bar, $baz
<@end@>


<@dynamichtml my_include@>
    <$foo = "tempValue1"$>
    <$bar = "tempValue2"$>
<@end@>


<$foo = 5$>
<$ddIncludePreserveValues("my_include", "MyPreservedFields")$>
```

```
<$trace(foo, "#console")$>   [[% Outputs: 5 %]]
```

ddLoadIndexedColumnResultSet()

This function loads a dynamicdata table into a ResultSet. This function is similar to `ddLoadResultSet` except that it only loads those rows whose values for `indexColumn` match `indexValue`. The comparisons made on this column are case-insensitive.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be loaded.
`indexColumn`	The name of an indexed column in the dynamicdata table.
`indexValue`	The value to use to select a section of the original table. This value will be checked in a case-insensitive manner against the *indexColumn* in each row. If it matches, that row is loaded into the ResultSet; otherwise, it is skipped.
`mappingTableName`	The name of a dynamicdata table used to rename the columns of the final ResultSet. The renaming is done by mapping the column names in the specified mapping table to the values of the first row in the same table. This is done exactly the same in `ddLoadResultSet`.

Output

If the dynamicdata table is successfully loaded, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
<?commatable indexedColumns="fieldA"?>
fieldA, fieldB, fieldC
1,      2,       3
4,      5,       6
1,      3,       9
<@end@>


<$exec ddLoadIndexedColumnResultSet("MyDataTable", "MyResultSet", "fieldA", "1")$>
<$exec rsFirst("MyResultSet")$>
<$foo = MyResultSet.fieldB$>    [[% (foo == '2') %]]
<$exec rsNext("MyResultSet")$>
<$bar = MyResultSet.fieldC$>    [[% (bar == '9') %]]
```

See Also

- ddLoadResultSet()

ddLoadResultSet()

This function loads a dynamicdata table into a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be loaded.
`mappingTableName`	The name of a dynamicdata table used to rename the columns of the final ResultSet. The renaming is done by mapping the column names in the specified mapping table to the values of the first row in the same table. This is done exactly the same in `ddLoadIndexedColumnResultSet`.

Output

If the dynamicdata table is successfully loaded, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
fieldA, fieldB, fieldC
1,      2,       3
4,      5,       6
1,      3,       9
<@end@>


<$exec ddLoadResultSet("MyDataTable", "MyResultSet")$>
<$exec ddLoadResultSet("MyDataTable", "MyAliasedResultSet")$>
<$exec rsFirst("MyResultSet")$>
<$exec rsFirst("MyAliasedResultSet")$>
<$foo = MyResultSet.field1$>         [[% (foo == 'a') %]]
<$bar = MyAliasedResultSet.alias2$>  [[% (bar == 'c') %]]
```

See Also

- ddMergeIndexedColumnResultSet()

ddMergeIndexedColumnResultSet()

This function merges a dynamicdata table into a ResultSet. This function is similar to `ddMergeResultSet` except that it only merges those rows whose values for `indexColumn` match `indexValue`. The comparisons made on this column are case-insensitive.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be merged.
`indexColumn`	The name of an indexed column in the dynamicdata table.
`indexValue`	The value to use to select a section of the original table. This value is checked in a case-insensitive manner against the indexColumn in each row. If it matches, that row is loaded into the ResultSet; otherwise it is skipped.
`mergeType`	The type of merge to perform. Set this to `replace` to prevent the merge from appending any new rows. Any other value for this parameter is treated as `append`.
`mappingTableName`	The name of a dynamicdata table which should be used to rename the columns of the final ResultSet. This renaming is done by mapping the column names in this mapping table to the values of the first row in this same table. This is done exactly as it is done in `ddLoadResultSet`.

Output

If the dynamicdata table is successfully merged, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
fieldA, fieldB, fieldC
1,      2,       3
4,      5,       6
1,      3,       9
<@end@>


<$exec rsCreateResultSet("MyResultSet", "fieldA,fieldB,fieldC")$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.fieldA = "4"$>
<$MyResultSet.fieldB = "8"$>
<$MyResultSet.fieldB = "23"$>
<$exec rsAppendNewRow("MyResultSet")$>
```

```
<$MyResultSet.fieldA = "8"$>
<$MyResultSet.fieldB = "21"$>
<$MyResultSet.fieldB = "59"$>
<$exec ddMergeIndexedColumnResultSet("MyDataTable", "MyResultSet", "fieldA", "4",
"replace")$>
<$exec rsFirst("MyResultSet")$>
<$foo = MyResultSet.fieldB$>    [[% (foo == '5') %]]
<$exec rsNext("MyResultSet")$>
<$bar = MyResultSet.fieldC$>    [[% (bar == '59') %]]
```

See Also

- ddMergeResultSet()

ddMergeResultSet()

This function merges a dynamicdata table into an existing ResultSet. If the target ResultSet does not exist, then this function acts exactly like `ddLoadResultSet` and the target ResultSet is simply created. Otherwise, the merge is performed using the first column of the new ResultSet as the merge key. By default, the merge appends any rows from the new ResultSet that do not match any rows in the existing Idoc Script table, unless the `mergeType` parameter is set to replace, in which case no new rows are added to the Idoc Script ResultSet.

Note that this merge does not replace all rows matched by a particular row of the new ResultSet, just the first one that it finds. The matching is case-sensitive. The mergeType parameter is optional and defaults to null.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be merged.
`mergeType`	The type of merge to perform. Set this to `replace` to prevent the merge from appending any new rows. Any other value for this parameter is treated as `append`.
`mappingTableName`	The name of a dynamicdata table which should be used to rename the columns of the final ResultSet. This renaming is done by mapping the column names in this mapping table to the values of the first row in this same table. This is done exactly as it is done in `ddLoadResultSet`.

Output

If the dynamicdata table is successfully merged, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
fieldA, fieldB, fieldC
1,      2,       3
4,      5,       6
<@end@>


<$exec rsCreateResultSet("MyResultSet", "fieldA,fieldB,fieldC")$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.fieldA = "4"$>
<$MyResultSet.fieldB = "52"$>
<$MyResultSet.fieldC = "18"$>
<$exec ddMergeResultSet("MyDataTable", "MyResultSet", "replace")$>
```

```
<$exec rsFirst("MyResultSet")$>
<$foo = MyResultSet.fieldB$>     [[% (foo == 52) %]]
<$bar = MyResultSet.#numRows$>   [[% (bar == 1) %]]
```

See Also

- ddLoadResultSet()

ddMergeUsingIndexedKey()

This function pulls selective rows from a dynamicdata table and use them to replace particular row values in a target ResultSet. The values of a particular column in the target ResultSet are used as values for an index look up into the dynamicdata table. The subtable retrieved is used to replace column values in the target ResultSet that have matching column names. The dynamicdata table is assumed to have only one row in the subtable selected out by the index value. If it has more than one row, only the first row is used to merge in values into the target ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`idocTableName`	The name of the ResultSet into which the dynamicdata table should be merged.
`indexColumn`	The name of an indexed column in the dynamicdata table and the name of the column in the target ResultSet from which values are retrieved to select out subtables in the dynamicdata table.
`mappingTableName`	The name of a dynamicdata table which should be used to rename the columns of the final ResultSet. This renaming is done by mapping the column names in this mapping table to the values of the first row in this same table. This is done exactly as it is done in `ddLoadResultSet`.

Output

If the dynamicdata table is successfully merged, it returns `TRUE`.

Example

```
<@dynamicdata MyDataTable@>
<?commatable indexedColumns="fieldA"?>
fieldA, fieldB, fieldC
1,      2,       3
4,      5,       6
7,      8,       9
<@end@>


<$exec rsCreateResultSet("MyResultSet", "fieldA,fieldB,fieldC")$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.fieldA = "4"$>
<$MyResultSet.fieldB = "8"$>
<$MyResultSet.fieldB = "23"$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.fieldA = "7"$>
<$MyResultSet.fieldB = "20"$>
```

```
<$MyResultSet.fieldB = "41"$>
<$exec rsAppendNewRow("MyResultSet")$>
<$MyResultSet.fieldA = "8"$>
<$MyResultSet.fieldB = "21"$>
<$MyResultSet.fieldB = "59"$>
<$exec ddMergeIndexedColumnResultSet("MyDataTable", "MyResultSet", "fieldA")$>
<$exec rsFirst("MyResultSet")$>
<$foo = MyResultSet.fieldB$>    [[% (foo == '5') %]]
<$exec rsNext("MyResultSet")$>
<$bar = MyResultSet.fieldC$>    [[% (bar == '9') %]]
<$exec rsNext("MyResultSet")$>
<$bar = MyResultSet.fieldC$>    [[% (bar == '59') %]]
```

ddSetLocal()

This function takes a dynamicdata table and iterates over it, setting local data values for each row. The first column of the table is used as the list of keys, while the second column is used as the list of values.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

This function has one parameter, `dataTableName`, which is the name of the dynamicdata table to load.

Output

If the dynamicdata table exists, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
key, value
foo, 15
bar, 23
baz, 77
<@end@>


<$exec ddSetLocal("MyDataTable")$>
<$tmp1 = #local.foo$>   [[% (tmp1 == '15') %]]
<$tmp2 = #local.bar$>   [[% (tmp2 == '23') %]]
<$tmp3 = #local.baz$>   [[% (tmp3 == '77') %]]
```

See Also

- ddSetLocalByColumnsFromFirstRow()
- ddSetLocalByColumnsFromFirstRowIndexed()

ddSetLocalByColumnsFromFirstRow()

This function takes the first row of a dynamicdata table and, using the column names as keys, sets a local value for each column and its corresponding value. If there is no first row, then this function does nothing. This can be a useful method for quickly setting a lot of local values. The `filterInclude` and `includeColumns` properties of the dynamicdata table are ignored.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

This function has one parameter, `dataTableName`, which is the name of the dynamicdata table to load.

Output

If the dynamicdata table was successfully loaded, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
fieldA, fieldB, fieldC
foo,    bar,    baz
<@end@>


<$exec ddSetLocalByColumnsFromFirstRow("MyDataTable")$>
<$tmpStr1 = #local.fieldA$>   [[% (tmpStr1 == 'foo') %]]
<$tmpStr2 = #local.fieldC$>   [[% (tmpStr2 == 'baz') %]]
```

See Also

- ddSetLocal()
- ddSetLocalByColumnsFromFirstRowIndexed()

ddSetLocalByColumnsFromFirstRowIndexed()

This function takes the first row of a dynamicdata table, and, using the column names as keys, sets a local value for each column and its corresponding value. If there is no first row, then this function does nothing. This function is almost identical to ddSetLocalByColumnsFromFirstRow. The only difference is that this function uses the first row given an indexed column and value instead of the very first row of the dynamicdata table. If no indexed row is found then this function does nothing.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

The following table lists parameters for the function.

Parameters	Description
`dataTableName`	The name of the dynamicdata table to load.
`indexColumn`	The name of an indexed column in the dynamicdata table.
`indexValue`	The value to use to select a row of the original table. This value will be checked in a case-insensitive manner against the indexColumn in each row. If it matches, that row will be used and the rest of the table will be ignored.

Output

If the dynamicdata table is successfully loaded, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
<?commatable indexedColumns="fieldB"?>
fieldA, fieldB, fieldC
1,      2,      3
4,      5,      6
7,      8,      9
3,      5,      2
<@end@>


<$exec ddSetLocalByColumnsFromFirstRowIndexed("MyDataTable", "fieldB", 5)$>
<$tmpStr1 = #local.fieldA$>   [[% (tmpStr1 == '4') %]]
<$tmpStr2 = #local.fieldC$>   [[% (tmpStr2 == '6') %]]
```

See Also

- ddSetLocal()
- ddSetLocalByColumnsFromFirstRow()

ddSetLocalEmpty()

This function takes a dynamicdata table and iterates over it, clearing local data. The first column is used as the keys to clear.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

This function has one parameter, `dataTableName`, which is the name of the dynamicdata table to use.

Output

If the dynamicdata table exists, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
key
foo
bar
baz
<@end@>


<$foo = 1$>
<$bar = 2$>
<$baz = 3$>
<$exec ddSetLocalEmpty("MyDataTable")$>
<$if foo or bar or baz$>
    [[% This will not be executed as foo, bar, and baz are all empty. %]]
<$endif$>
```

See Also

- ddSetLocal()

ddSetLocalEmptyByColumns()

This function takes a dynamicdata table and uses the columns to clear values in local data.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

This function has one parameter, `dataTableName`, which is the name of the dynamicdata table to use.

Output

If the dynamicdata table exists, it returns TRUE.

Example

```
<@dynamicdata MyDataTable@>
foo, bar, baz
<@end@>


<$bar = "asdf"$>
<$exec ddSetLocalEmptyByColumns("MyDataTable")$>
<$if bar$>
    [[% This will not execute as bar will be empty. %]]
<$endif$>
```

See Also

- ddSetLocal()
- ddSetLocalEmpty()

DefaultAccounts

Defines the default accounts for anonymous users.

- This must be a comma-delimited list of accounts.
- Permissions for each account must be specified in parentheses after the account name.
- The `#none` entry grants privileges to content items that have no account assigned. The `#all` entry grants privileges to all accounts.
- Default is `#none(RWDA)`.
- Returns the list of accounts as a string.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Location

IntradocDir`/config/config.cfg`

Example

As a configuration setting, defines default account information:

```
DefaultAccounts=BOS(R),SEA(RW),MSP/Gen(RWD)
```

As Idoc Script, returns the account information as a string:

```
<$DefaultAccounts$>
```

See Also

- ExternalUserAccounts
- SelfRegisteredAccounts

defaultFieldInclude

Specifies the include to use to display the metadata field.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Example

Sets the default field-display include for a hidden field on a query page to `std_query_hidden_field`:

```
<$if isFieldHidden$>
    <$if isQuery and not (fieldType like "Date") and not (fieldType like "Int")$>
        <$defaultFieldInclude = "std_query_hidden_field"$>
    <$endif$>
```

See Also

- fieldCaptionInclude
- fieldEntryInclude

defaultOptionListScript

Defines a piece of Idoc Script that displays a standard option list field.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

None.

Example

Generates an option list using the `optList` function:

```
<$if optionsAllowPreselect and fieldValue$>
    <$defaultOptionListScript = "<$optList " & optionListName & ":fieldValue$>"$>
<$else$>
    <$defaultOptionListScript = "<$optList " & optionListName & ":noselected$>"$>
<$endif$>
```

Loops on the current option list ResultSet to generate an option list:

```
<@dynamichtml compute_resultset_option_list_script@>
<$if not optionListKey$>
    <$optionListKey = fieldName$>
<$endif$>
<$defaultOptionListScript = "<$loop " & optionListResultSet & "$>" & "<$inc('std_
resultset_option_list_item')$>" & "<$endloop$>"$>
<@end@>
```

See Also

- optionListScript
- optList()

DelimitedUserRoles

Retrieves a comma-delimited, colon-delimited list of roles the current user belongs to.

Type and Usage

- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

Returns the user role list as a string.

Example

Returns a list of roles formatted as follows:

```
:guest:,:PublicContributor:,:ClassifiedConsumer:
```

See Also

- UserRoles

docLoadResourceIncludes()

Loads all the includes in a specified content item for use in the display of the current page.

- The content item specified must have the file extension *idoc*.
- This function sets StatusCode as a side effect. Use the abortToErrorPage function if the specified file must successfully load for the page to correctly display.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.16, "Page Display"
- Section 4.2.13, "Idoc Script"

Parameters

Takes a CGI-encoded parameter list that specifies a content item that is checked into the Content Server system. The parameter options are listed in the following table.

Optional Parameters	Description
dID	If *dID* is not present, *dDocName* and *RevisionSelectionMethod* must be present. A rendition of the revision of the content item with this ID will be returned, if it exists, and the *RevisionSelectionMethod* parameter does not exist or has the value *Specific*.
dDocName	It is recommended that *dDocName* be present in all requests for content items where the *dDocName* is known. Error messages assume that it is present, as do other features such as forms. ■ If *dDocName* is not present, *dID* must be present and *RevisionSelectionMethod* must not be present. ■ If *RevisionSelectionMethod* is present, a rendition of a revision of the content item with this name with be returned, if it exists. ■ If *RevisionSelectionMethod* is not present, *dDocName* may be used in error messages.
RevisionSelectionMethod	If present, *dDocName* must be present. The value of this variable is the method used to compute a *dID* from the specified *dDocName*. The value may be *Specific*, *Latest*, or *LatestReleased*. ■ **Specific:** The *dDocName* is ignored, *dID* is required and is used to get a specific revision. ■ **Latest:** The latest revision of the content item (including revisions in a workflow) is used to compute the *dID*. ■ **LatestReleased:** The latest released revision of the content item is used to compute the *dID*.
Rendition	■ If not present, *Rendition* defaults to *Primary*. This parameter specifies the rendition of the content item. ■ If the value is *Primary*, *Web*, or *Alternate*, the primary, web-viewable, or alternate rendition of the selected revision is returned.

Note: When used in HCSP pages, the ampersand (&) character in the CGI-encoded parameter list must be changed to the & character.

Output

None.

Example

Loads the resource includes in the primary vault rendition of the latest revision of *mydoc*.

```
<$docLoadResourceIncludes("dDocName=mydoc&RevisionSelectionMethod=Latest")$>
```

See Also

- abortToErrorPage()

docRootFilename()

Retrieves the file name of a file without the extension or directory path.

- This function is typically used to extract the Content ID (*dDocName*) part of a static URL controlled by Content Server.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.10, "Directories and Paths"

Parameters

Takes one parameter:

- The only parameter is a path and file name.

Output

Returns the file name as a string.

Example

Returns the value *mydoc*:

```
docRootFilename("/groups/public/documents/adacct/mydoc.pdf")
```

DocTypeSelected

Evaluates whether the Type of the current content item matches the Type in the active ResultSet.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

- Returns TRUE if the content item Types match.
- Returns FALSE if the content item Types do not match.

Example

Returns value based on whether the content item type matches the type for the ResultSet.

```
<$DocTypeSelected$>
```

DocUrl

Retrieves the URL of the file in the `weblayout/` directory. This variable is evaluated once per content item, not once per service call.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

Returns the URL of the file as a string.

Example

Used to build URL links to content items.

```
<$if HasUrl$>
    <a href="<$DocUrl$>"><$dDocName$></a>
<$else$>
    <$dDocName$>
<$endif$>
```

docUrlAllowDisclosure()

Evaluates whether a URL can be disclosed to the current user.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"

Parameters

The only parameter is an absolute path, such as the following:

```
http://mycomputer/domain/groups/.../documents/mydoc.pdf
```

A full relative path can be used, such as the following:

```
/oracle/domain/.../documents/mydoc.pdf).
```

Output

Returns a Boolean value.

- Returns TRUE if the URL can be disclosed.
- Returns FALSE if the URL is restricted.

Example

Determines if the user can view the URL of the `mydoc.pdf` document.

```
<$docUrlAllowDisclosure("/domain/groups/documents/mydoc.pdf")$>
```

DownloadApplet

Enables the multiple file Download Applet.

- When set to `TRUE`, the Download Applet is enabled so that multiple files can be downloaded from a search results page.
- When set to `FALSE`, the Download Applet is disabled.
- Default is `FALSE`.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"
- Section 4.2.2, "Clients"

Location

- System Properties, Options tab, **Enable download applet**
- Admin Server, General Configuration, **Enable download applet**
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting, enables the Download Applet:

```
DownloadApplet=true
```

As script, evaluates the condition of the Download Applet:

```
<$DownloadApplet$>
```

See Also

- MultiUpload
- UploadApplet

DownloadSuggestedName

Retrieves the default path and suggested name for a file being downloaded.

The suggested name is based on the original file name of the content item.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.10, "Directories and Paths"

Output

Returns the path and suggested name for the downloaded file as a string.

Example

Returns the path and suggested name for the downloaded file:

```
<$DownloadSuggestedName$>
```

dpGet()

Function used to return the value set by the `dpSet` function.

Both `dpGet` and `dpSet` are used to control the logic of Content Profile rules that are to be displayed. Commonly, a user may create an activation condition that sets a value for a specified key. Even if the condition evaluates to false (that is, the rule fails to fire), the key is set. Another rule can now access the key and retrieve the value when evaluating its activation condition. By using these two functions, a user can create a dependency between rules. For example, a user may want a rule to be evaluated only when a preceding rule is evaluated.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

Takes one parameter:

- Key: The designation for the value to be returned.

Example

```
<$myValue = dpGet("myKey")$>
```

See Also

- dpSet()

dpPromote()

Function used to 'promote' values from a rule evaluation context into the request context. After the evaluation of all rules, the key and value specified by this function are pushed into the local data.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

Takes two parameters:

- The first parameter is the designation for the value to be returned.
- The second parameter is the value to be returned.

Example

This example changes the value for AutoNumberPrefix for one request to be the value for dDocType:

```
<$dpPromote("AutoNumberPrefix", dDocType)$>
```

This example causes the Title field to be hidden on checkin pages:

```
<$dpPromote("dDocTitle:isHidden", "1")$>
```

See Also

- dpPromoteRs()

dpPromoteRs()

Similar to `dpPromote`, this function allows a ResultSet that is generated in a profile rule to be promoted to the request context. The ResultSet is pushed into the DataBinder after the evaluation of all rules. The values are not evaluated during rule activation or manipulation but are available for page presentation.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

This function can take two parameters:

- The first parameter is the name of the ResultSet to push into the data.
- Optional. The second parameter is an alternate name for the ResultSet.

See Also

- dpPromote()

dpSet()

Function used to set temporary values during the evaluation of activation conditions or rule values anywhere Idoc Script is evaluated.

The values are set into a working area and can be retrieved by the `dpGet` function call. The key and value that is set does not persist, but is globally available for the profile being evaluated. All key/value pairs can be accessed by the rules of the profile.

Both `dpGet` and `dpSet` are used to control the logic of Content Profile rules that are to be displayed. Commonly, a user may create an activation condition that sets a value for a specified key. Even if the condition evaluates to false (that is, the rule fails to fire), the key is set. Another rule can now access the key and retrieve the value when evaluating its activation condition. By using these two functions, a user can create a dependency between rules. For example, a user may want a rule to be evaluated only when a preceding rule is evaluated.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

This function takes two parameters:

- Key: The designation for the value to be returned.
- Value: Value associated with the key.

Example

```
<$dpSet("myKey", "1")$>
```

See Also

- dpGet()

dWfName

Retrieves the name of the workflow.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the name of the workflow as a string.

Example

The following code in the *IdcHomeDir*/resources/core/templates/workflow_info.htm template page is used to display the workflow name on the Workflow Step Information page:

```
<td align=right><span class=infoLabel><$lc("wwLabelWorkflow")$></span></td>
<td><span class=tableEntry><$dWfName$></span></td>
```

dWfStepName

Retrieves the name of the current step in the workflow.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the name of the current step as a string.

Example

The following code in the *IdcHomeDir*/resources/core/templates/workflow_info.htm template page is used to display the current step name on the Workflow Step Information page:

```
<td align=right><span class=infoLabel><$lc("wwCurrentStep")$>
</span></td>
<td><span class=tableEntry>
    <$loop WorkflowStep$>
    <$dWfStepName$>
    <$exec RequiredUsers = dWfStepWeight$>
    <$if isTrue(dWfStepIsAll)$><$exec RequiredUsers = 0$><$endif$>
    <$endloop$></span>
</td>
```

EmptyAccountCheckinAllowed

Checks if an account must be specified on the checkin page.

Used on the Standard Page Resources page to display an error message if an account is not specified.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"

Output

- Returns TRUE if an Account value is required.
- Returns FALSE if an Account value is not required.

Example

Evaluates whether an account number is required and displays an error message.

```
<$if not EmptyAccountCheckinAllowed$>
    <$isRequired = 1, requiredMsg = "Please specify an account."$>
<$endif$>
```

EnableDocumentHighlight

Enables highlighting of full-text search terms in PDF, text, and HTML files.

- When set to TRUE, search term highlighting is enabled.
- When set to FALSE, search term highlighting is disabled.
- Default is FALSE after installation of Content Server.
- Default is TRUE after installation of PDF Converter.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.20, "Searching"

Location

- System Properties, Options tab, Enable search keyword highlighting
- Admin Server, General Configuration, Enable search keyword highlighting
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting:

```
EnableDocumentHighlight=false
```

As script, returns the value of the configuration setting:

```
<$EnableDocumentHighlight$>
```

See Also

- UseHtmlOrTextHighlightInfo

encodeHtml()

Idoc function used to filter data input for illegal or corruptive HTML constructs. This function is used by default in Threaded Discussions, a component available during installation.

All input data received by Content Server when using the `unsafe` value for the `rule` parameter applies only to well-known unsafe script tags. This functionality can be altered by using the `HtmlDataInputFilterLevel` configuration variable to change the filtering that is done.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21, "Security"

Parameters

Takes two required and one optional parameter:

- The first parameter is the string to encode.
- The second parameter is the rule to apply when encoding HTML constructs. The following values are allowed:
  - `none`: no conversion is done to HTML constructs.
  - `unsafe`: only well-known unsafe script tags are encoded.
  - `exceptsafe`: well-known safe script tags are not encoded.
  - `lfexceptsafe`: recommended for use when extended comments are entered by users and line breaks in the original text must be preserved.
- An optional parameter is also available that specifies if long strings without space characters are to be broken up and what maximum word size to apply. Specify either `wordbreak` or `nowordbreak`.

Output

Returns the encoded string.

See Also

- "HtmlDataInputFilterLevel" in *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

entryCount

Retrieves the number of times the current workflow step has been entered by the current revision.

- This variable can be used to create conditional statements, but it should not be hard-coded or altered.
- This variable is localized in the companion file and maintained in the key:

  `<step_name>@<workflow_name>.entryCount`

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the entry count as an integer.

Example

The following code defines a jump called `MaxEntry`, which exits to the parent workflow and notifies the reviewers if the last time the step was entered was more than one week ago:

```
<$if wfCurrentGet("entryCount")==2$>
    <$wfSet("WfJumpName","MaxEntry")$>
    <$wfSet("WfJumpTargetStep",wfExit(0,0))$>
    <$wfSet("WfJumpEntryNotifyOff","0")$>
<$endif$>
```

eval()

Evaluates a variable definition as if it were Idoc Script. Can be used to recursively evaluate a literal string.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is a variable that defines a string to be evaluated as Idoc Script.

Output

Returns the parameter string, with any Idoc Script in the variable definition resolved.

Example

Variable one is assigned the string `CompanyName`, and variable two is assigned a string that includes variable one within Idoc Script delimiters. On a page, variable one presents the string `CompanyName`, variable two presents the string `Welcome to <$one$>` and `eval(two)` presents the string `Welcome to CompanyName`.

```
<$one="CompanyName"$>
<$two="Welcome to <$one$>"$>
<$one$><br>
<$two$><br>
<$eval(two)$>
```

Say you wanted to exclude all metadata fields that have the prefix `xPrefix` in their names. You could use the `eval()` function to dynamically write and evaluate Idoc Script for this purpose:

```
<$loop DocMetaDefinition$>
<$if strIndexOf("xPrefix", dName) >= 0$>
    <$myScript = "<$" & dName & ":isExcluded=1$>"$>
    <$eval(myScript)$>
<$endif$>
<$endloop$>
```

See Also

- Section 3.4.1, "Keywords Versus Functions"
- exec
- setResourceInclude()

ExclusiveCheckout

Determines whether users can check out content that was authored by another user.

- When set to TRUE, only the author or a user with Admin permission to the security group can check out a content item.
- When set to FALSE, users with Write permission to the security group can check out content that was authored by another user.
- Default is FALSE.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Location

- System Properties, Content Security tab, Allow only original contributor to check out
- Admin Server, Content Security, Allow only original contributor to check out
- *IntradocDir*/config/config.cfg

Example

As a configuration setting, only original contributor can check out a content item.

```
ExclusiveCheckout=true
```

As Idoc Script, returns the value of the configuration setting:

```
<$ExclusiveCheckout$>
```

See Also

- isUserOverrideSet()

exec

Executes an Idoc Script expression and suppresses the output (does not display the expression on the page).

- In earlier versions of Idoc Script, the *exec* keyword was required to suppress the *value* of any variable from appearing in the output file. In the current version, the *exec* keyword is needed only to suppress an *expression* from appearing in the output.

  For example, the first line below is equivalent to the last two lines:

  ```
  <$varA="stringA", varB ="stringB"$>
  <$exec varA="stringA"$>
  <$exec varB="stringB"$>
  ```

- The *exec* keyword is typically used to evaluate behind-the-scenes code, such as specifying an include to be used later in the page.

Type and Usage

- Section 3.4, "Special Keywords"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is an Idoc Script expression.

Output

Returns the value of the expression, but does not display the expression on the page.

Example

Evaluates the `inc` global function to incorporate the specified includes without displaying their code on the page:

```
<@dynamichtml std_definitions@>
    <$exec inc("std_page_variable_definitions")$>
    <$exec inc("define_image_files")$>
<@end@>
```

See Also

- Section 3.4.1, "Keywords Versus Functions"
- eval()

executeService()

Executes a Content Server service.

- This function allows the specified service to be executed while the page is being constructed. Generally, services are executed using a tool such as IdcCommand or the CGI URL on the browser.
- Used with dynamic server pages.
- Services that can be called with the `executeService` function must be scriptable, meaning that they do not require parameter input. Scriptable services have an access level of 32 or more.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is the name of the service to be executed. The live data in the DataBinder is used as parameters to the specified service.

Output

- Returns a StatusCode value of `0` if the service was executed successfully.
- Returns a StatusCode value of `-1` if an error in the service execution occurred.
- All output of the executed service is suppressed, but any ResultSets and loaded values are available.

Example

Executes a service when given a service name:

```
<$executeService("servicename")$>
```

See Also

- StatusCode

ExternalUserAccounts

Retrieves the default roles for users who are defined by an external user base (NTLM, Active Directory, or LDAP).

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.21.2, "External Security"
- Section 4.2.24, "Users"

Output

- Returns a comma-delimited list of accounts, with permissions for each account specified in parentheses after the account name.
- The `#none` entry grants privileges to content items that have no account assigned. The `#all` entry grants privileges to all accounts.

Example

As script, returns the default account information as a string:

```
<$ExternalUserAccounts$>
```

See Also

- ExternalUserRoles
- UserAccounts
- DefaultAccounts

ExternalUserRoles

Retrieves the default roles for users who are defined by an external user base (NTLM, Active Directory, or LDAP).

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.21.2, "External Security"
- Section 4.2.24, "Users"

Output

- Returns the list of roles as a string.

Example

As script, returns the default role information as a string:

```
<$ExternalUserRoles$>
```

See Also

- ExternalUserAccounts
- UserRoles

fieldCaption

Specifies the caption label for a metadata field.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the caption of the metadata field as a string.

Example

Defines the caption for the `dDocTitle` metadata field as the `wwTitle` localized string:

```
<$fieldName = "dDocTitle", fieldCaption = lc("wwTitle"), isRequired = 1, fieldType
= "BigText", requiredMsg = lc("wwSpecifyTitle")$>
```

fieldCaptionInclude

Specifies the include to use to display the caption for the metadata field.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

None.

Example

Sets the caption display include to `std_field_caption`:

```
<$fieldCaptionInclude = "std_field_caption"$>
```

See Also

- defaultFieldInclude
- fieldEntryInclude
- fieldCaptionStyle

fieldCaptionStyle

Specifies the style of the caption for the metadata field.

The following SPAN styles are typically used for captions. They are defined in the `std_style_declaration` include in the *IdcHomeDir*`/resources/core/std_page.htm` resource file:

- `searchLabel`
- `infoLabel`
- `tableEntry`
- `requiredField`

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the name of the caption style.

Example

Sets the caption style to *requiredField*:

```
<$if isRequired and not suppressCaptionHighlighting$>
    <$fieldCaptionStyle = "requiredField"$>
<$endif$>
Generates the standard field caption:
<@dynamichtml std_field_caption@>
    <span class=<$fieldCaptionStyle$>><$fieldCaption$><$if not
    isFormSubmit$><$":"$><$endif$></span>
<@end@>
```

See Also

- fieldCaption
- fieldCaptionInclude
- fieldValueStyle
- "NotationForRequiredFields" and "StyleForRequiredFields" in *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

fieldDefault

Specifies the default value for a metadata field.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the default value of the metadata field as a string.

Example

Defines the default value for the *dDocTitle* metadata field:

```
<$fieldName = "dDocTitle", fieldCaption = lc("wwTitle"), isRequired = 1,
     fieldType = "BigText", requiredMsg = lc("wwSpecifyTitle"),
     fieldDefault="Enter a Title"$>
```

fieldEditWidth

Specifies the character width of the metadata input field on the HTML page. It is set in `compute_namevalue_edit_widths`, which is included in the resource `compute_std_field_includes`.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

None.

fieldEntryInclude

Specifies the include to use to display the value for the metadata field.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Example

Sets the value display include to `std_checkbox_entry`:

```
<$fieldEntryInclude = "std_checkbox_entry"$>
```

See Also

- defaultFieldInclude
- fieldCaptionInclude
- fieldValueStyle

fieldExtraScriptInclude

Specifies the name of the include file containing additional JavaScript validation for a specific metadata field. This is set in the resource `compute_std_field_includes`.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

fieldInclude

Specifies the name of the include to display a metadata field. The include file is usually `std_namevalue_field`, but it can be set to another include.

This is usually set at the top of the page with `xFieldName:include` but can also be set by overriding the resource `compute_std_field_includes`.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

fieldIsOptionList

Specifies whether a metadata field has an option list.

- When set to TRUE, the field has an option list.
- When set to FALSE, the field does not have an option list.
- Default is FALSE.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field has an option list.
- Returns FALSE if the field does not have an option list.

Example

Defines the *dDocAccount* metadata field as an option list if predefined accounts exist:

```
<$if HasPredefinedAccounts$>
    <$fieldIsOptionList = 1, optionListName = "docAccounts",
    fieldOptionListType = "combo"$>
<$endif$>
```

See Also

- hasOptionList
- fieldOptionListType
- optionListName

fieldMaxLength

Specifies the maximum length of the metadata field. This is not the size of the field but the maximum number of characters that the field can contain. It is usually set at the top of the page with `xFieldName:include` but can also be set by overriding the resource `compute_std_field_includes`.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

fieldName

Specifies the internal name of a metadata field.

- Predefined metadata fields begin with a lowercase `d` (for example, `dDocName`).
- Custom metadata fields begin with a lowercase `x` (for example, `xComments`).

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the name of the metadata field as a string.

Example

Defines the *dDocTitle* metadata field:

```
<$fieldName = "dDocTitle", fieldCaption = lc("wwTitle"), isRequired = 1, fieldType
= "BigText", requiredMsg = lc("wwSpecifyTitle")$>
```

fieldOptionListType

Specifies the type of option list for a metadata field.

Possible values for this variable are:

fieldOptionListType Value	Corresponding Option in Configuration Manager
strict	Select List Validated or Select List Not Validated
combo	Edit and Select List
multi	Edit and Multiselect List
access	None (Used in Collaboration Server for member access lists)

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the option list type as a string.

Example

Defines the *dDocAccount* option list type as *combo* (Edit and Select List):

```
<$if HasPredefinedAccounts$>
    <$fieldIsOptionList = 1, optionListName = "docAccounts",
    fieldOptionListType = "combo"$>
<$endif$>
```

See Also

- fieldIsOptionList
- optionListName

fieldType

Specifies the type of metadata field.

Possible values for this variable are:

fieldType Value	Corresponding Option in Configuration Manager
Text	Text
BigText	Long Text
Int	Integer
Date	Date
Memo	Memo

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the metadata field type as a string.

Example

Defines the *dDocTitle* metadata field as a `BigText` (`Long Text`) field:

```
<$fieldName = "dDocTitle", fieldCaption = lc("wwTitle"), isRequired = 1,
    fieldType = "BigText", requiredMsg = lc("wwSpecifyTitle")$>
```

fieldValue

Specifies the value for a metadata field.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the value of the current metadata field.

Example

Generates the standard field value:

```
<@dynamichtml std_value_label@>
    <span class="<$fieldValueStyle$>"><$fieldValue$></span><!--""-->
<@end@>
```

See Also

- fieldEntryInclude
- fieldCaptionStyle
- fieldValueStyle
- optionListName

fieldValueStyle

Specifies the style of the value for the metadata field.

The following SPAN styles are typically used for values. They are defined in the `std_style_declaration` include in the *IdcHomeDir*`/resources/core/std_page.htm` resource file:

- `tableEntry`
- `xxsmall`
- `strongHighlight`

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the name of the value style.

Example

Sets the value style:

```
<$if isFieldInfoOnly$>
    <$if valueStyle$>
<$fieldValueStyle = valueStyle$>
    <$else$>
<$fieldValueStyle = "tableEntry"$>
    <$endif$>
<$endif$>
```

Generates the standard field value:

```
<@dynamichtml std_value_label@>
    <span class="<$fieldValueStyle$>"><$fieldValue$></span><!--'"-->
<@end@>
```

See Also

- fieldEntryInclude
- fieldCaptionStyle
- valueStyle

fieldWidth

Specifies the width of a metadata field, in characters.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the width of the current metadata field.

Example

Generates the *Extension* field with a width of 10 characters:

```
<$fieldName = "dExtension", fieldCaption = lc("wwNativeExtension"),
fieldWidth = 10$>
```

See Also

- fieldCaptionStyle

fileUrl

Retrieves the relative URL of the current dynamic server page (HCSP or HCST).

This variable is typically used in self-referencing pages, such as a form that posts back to itself.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

Returns the URL as a string.

Example

Returns the relative URL of the current dynamic server page:

```
<$fileUrl$>
```

FIRSTREV

Returns the first revision label for the current revision label sequence.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

Returns the first revision label as a string.

Example

Returns the first revision label (default is 1):

```
<$FIRSTREV$>
```

ForcedConversionRules

Defines the Dynamic Converter rules that will apply forced conversions upon file checkin.

- This is a comma-delimited list.
- If a content item matches any of the specified conversion rules upon checkin, the file will be converted according to that rule. Each file can be converted into multiple renditions.
- There is no default.

> **Note:** The forced conversion process does not apply the same Dynamic Converter template twice. For example, if you set `ForcedConversionRules=RuleA,RuleB`, but RuleA and RuleB specify the same template with a different layout, the conversion according to RuleB will not occur. (Best practice in this case would be to merge the layouts and use Idoc Script to dynamically select the appropriate layout elements.)

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.6.2, "Dynamic Converter"

Location

IntradocDir`/config/config.cfg`

Example

Used as a configuration entry:

```
ForcedConversionRules=Rule1,Rule2,Rule3
```

See Also

- incDynamicConversionByRule()
- "DisableForcedConversions" and *"rule*:IsFragmentOnly" in *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

forceExpire()

This function can be used to force the cache for a particular include to expire. This is useful in the case of a change in the page display, such as adding new metadata fields, or if the user has saved a new query, or altered which links are viewable.

If the value for 'includeName' is null, or an empty string, then the entire cache will be eliminated. This is useful when all includes for a particular page are placed into the same cache, and need to be expired at the same time.

For more information about keywords versus functions, see Section 3.4.1, "Keywords Versus Functions."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

Takes two required parameters and two optional parameters. For example:

```
forceExpire(includeName, scope [, cacheName, key])
```

Parameter	Description
includeName	The name of the dynamic html element to evaluate.
scope	Set to 'session' if each user should see different html, or 'application' if all users will see the same thing.
cacheName	Optional: if you want to place this data into a named cache instead of the default cache. If an empty string is passed, it will cache the include into the default cache for the session.
key	Optional: if you do not want automatic name-scoping of your cache to prevent conflicts, you can specify a unique key here.

Example

This can be used as a replacement for the 'std_page_begin' include. It will verify if the user executed one of the standard service calls to modify the links on the standard left navigation bar (saving a query, altering the portal design). If so, the cached html for the standard page begin will be invalidated. Then, the next 'cacheInclude' function will reevaluate the include, and place it back into the cache.

```
<$if strEquals(IdcService, "PNE_SAVE_QUERY") or strEquals(IdcService, "PNE_UPDATE_
PORTAL_INFO")$>
<$forceExpire("std_main_page_begin", "session", "", "std_main_page_begin")$>
<$endif$>
<$cacheInclude("std_main_page_begin", "session", 600, "", "std_main_page_begin")$>
```

See Also

- cacheInclude()
- inc()
- setExpires()
- setHttpHeader()

- setMaxAge()

formatDate()

Reformats a date/time to the default date/time format.

Database-formatted dates cannot be evaluated (for example, 2001-06-15).

Long-formatted dates cannot be evaluated (for example, June 15, 2001).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string that specifies a date/time.

Output

- Returns the date/time in the format used by dateCurrent (for example, 6/15/01 1:55 PM).
- Returns null if the parameter cannot be evaluated.
- If a time is not provided, returns a default time of 12:00 AM.

Example

The following example formats the date and time and displays it as 12/14/99 2:00 PM:

```
<$formatDate("12/14/1999 02:00 PM")$>
```

The following example formats the date, assigns the default time, and displays it as 9/15/03 12:00 AM:

```
<$formatDate("09/15/2003")$>
```

The following script formats and displays a specified date and time. Line one evaluates an alternate date and time format and assigns it to a custom variable. Line two displays this date to a user.

```
<$my_customDateTime = formatDate("06/14/2004 15:05:34")$>
Final Approval: <$my_customDateTime$>
```

Output:

Final Approval: 6/14/04 3:05 PM

formatDateDatabase()

Formats the date and time in preparation for an SQL query. Long-formatted dates cannot be evaluated (for example, May 22, 2000).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string or variable that specifies a date and time.

Output

- Returns an ODBC-formatted date and time:

  ```
  [ts 'yyyy-mm-dd hh:mm:ss']
  ```

- Returns null if the parameter cannot be evaluated.
- If a time is not provided, returns a default time of 00:00:00.

Example

Formats the current date and time for an SQL query:

```
<$formatDateDatabase(dateCurrent())$>
```

Formats the date and time and displays as 2001-03-19 15:32:00:

```
<$formatDateDatabase("03/19/2001 3:32 PM")$>
```

Formats the date and time and displays as 1999-04-03 00:00:00:

```
<$formatDateDatabase("4/3/99")$>
```

formatDateDisplay()

Reformats a date/time to a date/time format for display to the user. Uses the "Display Date Format" in System Properties to format the date.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string that specifies a date/time.

Output

- Returns the date/time in the format used.
- Returns null if the parameter cannot be evaluated.

Example

```
<$formatDateDisplay$>
```

See Also

- formatDateOnlyDisplay()

formatDateOnly()

Reformats a date/time to the default date format and strips out the time.

Database-formatted dates cannot be evaluated (for example, 2000-02-02).

Long-formatted dates cannot be evaluated (for example, June 12, 2001).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string that specifies a date/time.

Output

- Returns the date only in the format used by dateCurrent (for example, 7/12/00).
- Returns null if the parameter cannot be evaluated.

Example

Returns the current date only (for example, 9/12/01):

```
<$formatDateOnly(dateCurrent())$>
```

Returns the date ten days in the future (for example, 9/22/01):

```
<$formatDateOnly(dateCurrent(10))$>
```

Formats the date and time and displays the date only as 1/17/00:

```
<$formatDateOnly("01/17/2000 2:00 PM")$>
```

This script displays the current date and a date 100 days in the future. Line one assigns the current date only to a custom variable. Line two assigns a date 100 days in the future to a second custom variable. Line three displays these dates to a user (for example, Start Date: 10/12/01 and End Date: 1/20/02):

```
<$my_startDate = formatDateOnly(dateCurrent())$>
<$my_endDate = formatDateOnly(dateCurrent(100))$>
Start Date: <$my_startDate$> and End Date: <$my_endDate$>
```

formatDateOnlyDisplay()

Reformats a date to a date format for display to the user. Uses the "Display Date Format" in System Properties to format the date.

Similar to formatDateDisplay but only formats the date.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string that specifies a date.

Output

- Returns the date in the format used.
- Returns null if the parameter cannot be evaluated.

Example

```
<$formatDateOnlyDisplay$>
```

See Also

- formatDateDisplay()

formatDateOnlyFull()

Reformats a date/time to the long date format and strips out the time. Database-formatted dates cannot be evaluated (for example, 2000-02-02).

This setting is deprecated for version 5.1 and later. Use formatDateWithPattern.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string that specifies a date/time.

Output

- Returns a long-formatted date:

  ```
  month d, yyyy
  ```

- Returns null if the parameter cannot be evaluated.

Example

Returns the current date in long format:

```
<$formatDateOnlyFull(dateCurrent())$>
```

Returns the date 365 days in the future in long format (for example, September 12, 2002):

```
<$formatDateOnlyFull(dateCurrent(365))$>
```

Formats the date only and displays as June 12, 2001:

```
<$formatDateOnlyFull("6/12/01 3:00 PM")$>
```

formatDateWithPattern()

Reformats a date/time to a specified date/time pattern.

When using this variable in a program (instead of dynamic page), add `!rfc` to the end of the date pattern. This provides a date format that conforms to the `rfc` standard. If `!rfc` is not added, a program will try to use the locale to create the date string, but in the case of a timed update event, no locale is specified. Consequently, the output uses integers instead of strings for the day and month.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

Takes two parameters:

- The first parameter is a date string used by the Content Server instance, or a date object created with the `parseDate` or `dateCurrent` functions.
- The second parameter is the date/time pattern, such as `MM/dd/yyyy`.

  The capital letter `Z` denotes the use of a UTC time zone for the entry. The lowercase `zzzz` denotes the time offset (`HHMM`) from the UTC time, preceded by a plus (+) or minus (-) sign to indicate the offset.

Output

- Returns the date/time in the format specified by the pattern parameter.
- Returns null if the parameter cannot be evaluated.

Example

Displays `Wed, 5 Jul 2006 12:08:56 -0700`:

```
<$formatDateWithPattern(dateCurrent(),"EEE, d MMM yyyy HH:mm:ss zzzz")$>
```

Displays 2006-07-05 14:30:33Z:

```
<$formatDateWithPattern(dateCurrent(),"yyyy-MM-dd HH:mm:ssZ")$>
```

formatTimeOnly()

Reformats a date/time to the default time format and strips out the date.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string or variable that specifies a date/time.

Output

- Returns a time only in the format used by dateCurrent (for example, 1:15 PM).
- Returns null if the parameter cannot be evaluated.

Example

Returns the current time only:

```
<$formatTimeOnly(dateCurrent())$>
```

Formats the time only and displays as 5:00 PM:

```
<$formatTimeOnly("2/2/99 5:00 PM")$>
```

Formats the time only and displays as 6:14 PM:

```
<$formatTimeOnly("04/21/2001 18:14:00")$>
```

formatTimeOnlyDisplay()

Reformats a time to a time format for display to the user. Uses the "Display Date Format" in System Properties to format the time.

Similar to formatDateDisplay but only formats the time.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is a string that specifies the time.

Output

- Returns the time in the format used.
- Returns null if the parameter cannot be evaluated.

Example

```
<$formatTimeOnlyDisplay$>
```

See Also

- formatDateDisplay()

GATEWAY_INTERFACE

Retrieves the revision level of the CGI specification to which this server complies. This variable is not request-specific; it is set for all requests.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the revision level as a string in the format `CGI/revision`.

Example

As information output on a page or to a log:

```
GATEWAY_INTERFACE=CGI/1.1
```

As part of an Idoc Script statement or evaluation:

```
<$if GATEWAY_INTERFACE$>
<!--statement-->
```

generateUniqueId

This function returns the unique ID for an HTML page. This is used with the `id` attribute in an HTML element to guarantee a unique ID.

Type and Usage

- Section 4.1.5, "Page Variables"
- Section 4.2.16, "Page Display"

Parameters

The only parameter is the field name for which the ID will be generated.

Output

Returns the ID for an HTML page.

getCookie

Obtains a cookie from a browser.

This is useful for tracking user sessions when the information does not need to be stored in the personalization.

Can be used to track the last pages navigated to, or the last searches which were run.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.25, "Web Servers"

Parameters

Takes one parameter, the name of the cookie.

Example

```
<$myCounter=getCookie("myCounter")$>
<$if not myCounter$>
    <$myCounter=1$>
<$endif$>
Num times loaded: <$myCounter$>
<$setCookie("myCounter", myCounter+1, dateCurrent(1))$>
```

See Also

- setCookie

GetCopyAccess

Determines what permission a user must have to get a copy of a content item.

When set to TRUE, users can get a copy of a content item for which they have only Read permission.

When set to FALSE, users must have Write permission to a content item to be able to get a copy.

Default is FALSE.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.21.1, "Internal Security"

Location

- System Properties, Content Security tab, Allow get copy for user with read privilege
- Admin Server, Content Security, Allow get copy for user with read privilege
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting:

```
GetCopyAccess=true
```

As Idoc Script, returns the value of the configuration setting:

```
<$GetCopyAccess$>
```

getDebugTrace()

Retrieves the output of the debug trace.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.9, "Debugging"

Output

- Returns the output of the accumulated debug trace for the page being constructed.
- Returns an empty string if `IsPageDebug` has not been set.

Example

Retrieves the output of the debug trace and outputs the information to a page:

```
<$getDebugTrace()$>
```

See Also

- IsPageDebug

getErrorTrace()

Retrieves the output of the error trace.

Error trace output is encoded for display in HTML pages. For example, the < and > delimiters are HTML-escaped and carriage returns are converted to <*br*> tags.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.9, "Debugging"

Output

- Returns the output of the accumulated error trace for the page being constructed.
- The function returns an empty string if `IsPageDebug` has not been set.

Example

Retrieves the output of the error trace and outputs the information to a page:

```
<$getErrorTrace()$>
```

See Also

- IsPageDebug

getFieldConfigValue

This function returns a configuration flag for a specific field. If the field does not exist, the default is returned instead.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.5, "Content Server"

Parameters

Takes three parameters:

- The name of a metadata field.
- The flag to return for the field when it is found.
- A default value to return if the field does not exist.

Output

Returns the specified configuration flag.

Example

```
<$caption = getFieldConfigValue("dDocTitle", "fieldCaption", lc("wwTitle"))$>
```

getFieldViewDisplayValue()

Returns the display value for an item in a Schema option list.

For example, assume an option list exists for a custom metadata field named `Customer`. In the database there is a schema table with the `CustomerName` column and a unique `CustomerID` column. On checkin the `CustomerName` value is visible to the user, but the `CustomerID` value is stored in the database. The `getFieldViewDisplayValue` function extracts the human-readable `CustomerName` value based on the `CustomerID` value.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

Takes three parameters:

- The first parameter is the name of the field to be used for the option list.
- The second parameter is the name of the schema view used for the current field.
- The third parameter is the value assigned to the field.

Output

Returns the display value.

Example

```
<$customerName = getFieldViewDisplayValue("xCustomer", "Customer_View", "1234")$>
```

getFieldViewValue()

This function is used when defining a content rule for a field. The value of a field for a content profile can be made dependent on a view by using this function. Because a field can have both a default and a derived value, the view can be used to create an interdependency between fields.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

Takes three parameters:

- Field. The name of a metadata field. This field must have an associated view.
- Value. A lookup key for value.
- Column name. A column in the view's table.

Output

Returns the value in the column specified using the field value as a lookup.

Example

In the following example, the view table for the field `xEmployee` has a column called `type`. Using the value as specified in the `xEmployee` metadata field, this function provides a lookup into this table and returns the `type` column value.

```
getFieldViewValue("xEmployee", #active.xEmployee, "type")
```

getFreeMemory()

This function returns the amount of free memory in the Java Virtual Machine (JVM). This is a performance auditing function used on the System Audit Information page.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.9, "Debugging"

Output

The amount of free JVM memory in megabytes.

Example

```
<$freeMem = getFreeMemory()$>
```

See Also

- getTotalMemory()

getHelpPage

This function returns a relative URL to a help page based on the name of the page.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.16, "Page Display"

Parameters

The name of the page used.

Example

```
<input type=Button value="<$lc("wwQuickHelp")$>"
    onClick="QuickHelp('<$getHelpPage("QH_AdvancedSearch")$>', 'Search')">
```

getOptionListSize

This function returns the size of an option list. It is used to determine whether to display custom user metadata fields on the User Profile page.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.16, "Page Display"

Parameters

The only parameter is the option list to be sized.

Output

Returns the size of the specified option list.

getParentValue()

This function returns a parent value from a Schema view. This is needed before generating the dependent option list for the current field.

In most cases this function returns the same value as the internal parent value parameter because most option lists trigger their dependency on the unique key for the parent field. However, this is not a requirement for complex dependent choice lists.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

Takes four parameters:

- The name of the schema view used for the current field.
- The name of the schema relation between this field and the parent field.
- The name of the parent field.
- The internal value for the parent field. This may be different than the value displayed on the page. This is often a unique number stored internally.

Example

```
<$parentValue=getParentValue("xState", "Country_State", "xCountry", "123")$>
```

getRequiredMsg()

This function is used on presentation pages to populate the Java script with the desired error message. The error message is set from the content rule user interface for a field that is designated as required. It returns a required message based on the following algorithm:

1. The function looks for the `fieldname:requiredMsg` value.
2. If the value exists, it is localized.
3. If the value is undefined or the message is empty, the function returns the default message and does not localize it. To localize the message, the caller of the function must use the `lc` function on the default message before passing it to this function (`getRequiredMsg`).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

Takes two parameters:

- `Fieldname`: Field associated with the error message.
- `Message`: Default message to be used.

See Also

- lc()

getTextFile()

Gets the web version of a text file and returns its contents to a string.

- This can be used only with files that have a format containing text, such as `text/html`, `text/plain`, or `text/xml`. Specifically, the `dFormat` field of the content item must start with `text`. For example, if an HCSP file is checked in, Formats: `text/hcsp` is displayed in the `Content Info` display; therefore, the content is displayed by the `getTextFile()` function.
- There must be enough information to determine the content item's web URL. The `dID` value is the only required metadata field.

  The following information must be present:

  – `dDocName`, `dWebExtension`, `dSecurityGroup`, `dRevLabel`, and `dDocType`.

  – If Accounts are enabled, `dDocAccount` must also be specified.

  This information is available automatically after a `GET_SEARCH_RESULTS` call or after a `DOC_INFO` call.
- This function should be wrapped with a `cacheInclude` call for greater performance. That can, however, create a security issue if all docs go to the same cache.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

This function does not take parameters but uses variables on the page to determine the document's URL.

Example

```
<$loop SearchResults$>
    <$fullDocument=getTextFile()$>
    <h2><$dDocTitle$></h2>
    <div><$fullDocument$></div>
<$endloop$>
```

See Also

- cacheInclude()

getTotalMemory()

This function returns the amount of total memory in the Java Virtual Machine (JVM). This is a performance auditing function used on the System Audit Information page to determine how much memory the server is using.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.9, "Debugging"

Output

The total JVM memory in megabytes.

Example

```
<$totalMem = getTotalMemory()$>
```

See Also

- getFreeMemory()

getUserValue()

Retrieves the value of a user metadata field for the current user.

The parameter must refer to a column in the `Users` database table. Unlike the user personalization functions that have no support for global reference, information assigned to the user in the `Users` table can be available to the Content Server instance.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"
- Section 4.2.24, "Users"

Parameters

The only parameter is a user-related variable.

Output

- Returns the value of the metadata field for the current user.
- Evaluates to `TRUE` if the user value was retrieved.
- Evaluates to `FALSE` if an error in retrieval occurred.
- Returns a StatusCode value of `-1` if the value is unspecified or unknown.

Example

Returns the user type of the currently logged on user:

```
<$getUserValue('dUserType')$>
```

getValue()

This function has multiple uses:

- Retrieves the value of a particular metadata field from local, active, or environment data.
- Retrieves the value of a particular column from a specific ResultSet.
- Retrieves information about ResultSet rows.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is either the type of data or the name of a ResultSet.
- The second parameter is the name of a metadata field, the column name, or a ResultSet row parameter.

Variations

You can also use a shorthand format that returns results identical to the `getValue(arg1,arg2)` function. The format uses the form `<$arg1.arg2$>`, where `arg1` and `arg2` are the literal string arguments "`arg1`" and "`arg2`" to the `getValue` function.

Note: Data types (`local`, `active`, and `env`) and ResultSet parameters must start with the # symbol.

The following parameter combinations can be used.

getValue and Shorthand Formats	Description
getValue("#local",*fieldName*) <$#local.*fieldName*$>	Retrieves the value of the specified metadata field from the local data.
getValue("#active",*fieldName*) <$#active.*fieldName*$>	Attempts to retrieve the value of the specified metadata field from the data in the following order: 1. Local data 2. Active ResultSets 3. All other ResultSets 4. Environment settings
getValue("#env",*fieldName*) <$#env.*fieldName*$>	Retrieves the value of the specified metadata field from the environment settings.
getValue(*ResultSetName*,*fieldName*) <$*ResultSetName*.*fieldName*$>	Retrieves the value of the specified metadata field from the current row of the specified ResultSet.

getValue and Shorthand Formats	Description
getValue(*ResultSetName*,*columnName*) <$*ResultSetName*.*columnName*$>	Retrieves the value of the specified column from the current row of the specified ResultSet.
getValue(*columnName*) <$*columnName*$>	Retrieves the value of the specified column from the current row of the current ResultSet.
getValue(*ResultSetName*,"#row") <$*ResultSetName*.#row$>	Retrieves the number of the current row in the specified ResultSet. The first row is 0.
getValue(*ResultSetName*,"#numRows") <$*ResultSetName*.#numRows$>	Retrieves the total number of rows in the specified ResultSet.
getValue(*ResultSetName*,"#isRowPresent") <$*ResultSetName*.#isRowPresent$>	Checks if at least one row is present in the specified ResultSet. This is useful when looping manually with rsNext.
getValue(*ResultSetName*,"#isEmpty") <$*ResultSetName*.#isEmpty$>	Checks if the specified ResultSet does not contain any rows.

Output

- For output values, see the preceding subsection, Variations.
- Returns an empty string if a value is not found.

Example

- Gets the Content ID from the ResultSet named *DOC_INFO*:

```
<$name = getValue("DOC_INFO","dDocName")$>
```

or

```
<$name = <$DOC_INFO.dDocName$>
```

- Checks to see if the passed parameter *dDocType* (which is in the local data) equals the value in the active ResultSet:

```
<$loop DocTypes$>
<$if strEquals(#active.dDocType, getValue("#local","dDocType"))$>
<!--do special HTML for selected document type-->
<$endif$>
<!-- additional statement-->
<$endloop$>
```

See Also

- Section 3.4, "Special Keywords"

getValueForSpecifiedUser()

Retrieves the value of a user attribute for a specific user. This function can be useful for defining extended workflow functionality.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.24, "Users"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters:

- The first parameter is the user name.
- The second parameter is the name of a column in the `Users` database table that specifies a user attribute (such as `dFullName`, `dUserType`, or `dEmail`).

Output

- Returns the value of the user attribute for the specified user.
- Returns an empty string if the value is not found.

Example

Retrieves the full name for *mjones*.

```
<$getValueForSpecifiedUser('mjones', 'dFullName')$>
```

getViewValue()

Returns the display value for an item in a Schema option list.

For example, assume an option list exists for a custom metadata field named `Customer`. In the database there is a schema table with the `CustomerName` and a unique `CustomerID`. On checkin the `CustomerName` is visible to the user but the `CustomerID` is what is stored in the database. The `getFieldViewDisplayValue` function extracts the human-readable `CustomerName` based on the `CustomerID`.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

This function takes three parameters:

- The name of the schema view used for the current field.
- The value stored in the database for the ID.
- The column name associated with the view.

Output

Returns the display value.

Example

```
<$custName = getViewValue("Customer_View", "1234", "CustomerName")$>
<$custRegion = getViewValue("Customer_View", "1234", "CustomerRegion")$>
```

See Also

- getFieldViewDisplayValue()

getViewValueResultSet()

This function loads a schema table and places it on the page as a ResultSet named SchemaData. The column names in the ResultSet are the same as the names in the database table.

This function is most useful to obtain a list of dependent choices based on a parent value.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

Takes three parameters:

- The name of the schema view used.
- The relation for the schema and the view.
- The value for the schema parent.

Example

Assume you have a schema table and view for a list of countries. The view is named `Country_view`.

To output the contents of that table to the page, use the following code.

```
<$getViewValuesResultSet("Country_View", "", "")$>
<$loop SchemaData$>
<$count = 0, num = rsNumFields("SchemaData")$>
    <$loopwhile count < num$>
        <$fieldName=rsFieldByIndex("SchemaData", count)$>
        <$fieldName$> = <$getValue("SchemaData", fieldName)$>
        <$count = count + 1$>
    <$endloop$>
<$endloop$>
```

This will output the table even if you do not know the column name.

In the following example, assume you have a DCL for the fields `Country` and `State`. The list of `States` depends on which `Country` is selected. To obtain the list of `States` when the `Country` is USA, use this code:

```
<$getViewValuesResultSet("State_View", "Country_State", "USA")$>
```

hasAppRights()

Checks if the current user has rights to an administrative application.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"

Parameters

The only parameter is one of the following administrative applications:

- UserAdmin
- WebLayout
- RepoMan
- Workflow
- ConfigMan
- Archiver

Output

- Returns TRUE if the user has rights to the specified application.
- Returns FALSE if the user does not have rights to the specified application.
- Returns a StatusCode value of `-1` if the value is unspecified or unknown.

Example

Evaluates whether the current user has rights to the specified application.

```
<$hasAppRights('RepoMan')$>
```

HasExternalUsers

Indicates to Content Server that an external user database is present.

When set to `TRUE`, the system recognizes external users. If a custom component has been written to support an external user integration (such as LDAP), this should be set to `TRUE`.

Default is `FALSE`.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"
- Section 4.2.21.2, "External Security"

Location

IntradocDir`/config/config.cfg`

Example

As a configuration setting:

```
HasExternalUsers=true
```

As Idoc Script, returns the value of the configuration setting:

```
<$HasExternalUsers$>
```

See Also

- "NtlmSecurityEnabled" in *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

HasLocalCopy

Checks whether the client computer has a copy of the requested content item in the download target directory.

Generally used to query the user whether to overwrite when downloading. For example, this variable is used by the Oracle ODMA Client.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.2, "Clients"

Output

- Returns TRUE if a local copy is detected.
- Returns FALSE if a local copy is not detected.

Example

Checks for a local copy of the content item:

```
<$HasLocalCopy$>
```

See Also

- ClientControlled
- IsNotLatestRev

hasOptionList

Specifies that the metadata field has an option list.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field has an option list.
- Returns FALSE if the field does not have an option list.

Example

Specifies that the field has an option list:

```
<$hasOptionList=1$>
```

Generates the option list values if the field has an option list:

```
<$if hasOptionList$>
    <$if isQuery$></td><td><$endif$>
    <$include std_option_list_entry$>
<$endif$>
```

See Also

- fieldIsOptionList

HasOriginal

Checks if an original file exists for a revision.

This variable checks for a `dOriginalName` value.

It is possible for a content item to be checked in but to not have an original file (for example, when a Basic workflow has been started but files have not been checked in at the initial contribution step).

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

- Returns TRUE if an original file exists.
- Returns FALSE if no original file exists.

Example

Checks for an original file:

```
<$if HasOriginal and not isNew$>
```

HasPredefinedAccounts

Checks if the current user has permission to any predefined accounts. Predefined accounts are those that are created in the User Admin utility.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user is assigned to any predefined accounts.
- Returns FALSE if the user is not assigned to any predefined accounts.

Example

Displays the Accounts option list if the user is assigned to any predefined accounts:

```
<$if HasPredefinedAccounts$>
    <$fieldIsOptionList = 1, optionListName = "docAccounts",
    fieldOptionListType = "combo"$>
<$endif$>
```

HasUrl

Checks if a file exists in the `weblayout` directory for the current content item.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

- Returns `TRUE` if a `weblayout` file exists.
- Returns `FALSE` if no `weblayout` file exists.

Example

Checks for a `weblayout` file:

```
<$if HasUrl$>
    <$include doc_url_field$>
<$endif$>
```

HeavyClient

Checks if the checkin is through the Content Server ODMA client or Upload applet.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.2, "Clients"

Output

- Returns `TRUE` if the ODMA client or `Upload` applet is being used for checkin.
- Returns `FALSE` if the ODMA client and `Upload` applet are not being used for checkin.

Example

Checks for check in method:

```
<$HeavyClient$>
```

htmlRefreshTimeout

Similar to `DefaultHtmlRefreshTimeoutInSeconds`. Defines the time, in seconds, that a Work In Progress page, My Checked-Out Content page or My Workflow Assignments page refreshes.

`htmlRefreshTimeout` can be set in the URL or the service's databinder. This can be used to set a different refresh time for different pages. If that is needed, do not set `DefaultHtmlRefreshTimeoutInSeconds` in the `config.cfg` file, but instead set `htmlRefreshTimeout` in the URL or databinder.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.16, "Page Display"

Example

In the page's URL:

```
&htmlRefreshTimeout=90
```

See Also

- "DefaultHtmlRefreshTimeoutInSeconds" in *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- htmlRefreshUrl

htmlRefreshUrl

Used to set the URL of the page to load after a Work In Progress page, My Checked-Out Content page or My Workflow Assignments page is refreshed. The default is the current page. This variable can be set in the URL of the page or the databinder of the service.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.16, "Page Display"

Example

In the page's URL:

```
&htmlRefreshUrl=http://www.home.com
```

See Also

- "DefaultHtmlRefreshTimeoutInSeconds" in *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- htmlRefreshUrl

HttpAbsoluteCgiPath

Retrieves the Content Server CGI path as a complete URL.

This variable cannot be modified directly; to change the path, use the `HttpAbsoluteCgiRoot` configuration setting, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the path as a string.

Example

Returned for a Content Server *domain*:

```
HttpAbsoluteCgiPath=http://localhost/domain/idcplg/
```

See Also

- HttpBrowserFullCgiPath
- HttpCgiPath
- "HttpAbsoluteCgiRoot" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

HttpAdminCgiPath

Retrieves the Admin Server CGI path as a relative URL.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the Admin Server CGI path as a string.

Example

Returned for a Content Server instance:

```
HttpAdminCgiPath=/idcplg/idc_cgi_isapi-instance.dll/cs-admin/pxs
```

HttpBrowserFullCgiPath

This variable is used to set explicit control over the CGI path construction made for applets and the Admin Server.

Content Server evaluates the HTTP address in the address bar of the browser. If the `HttpIgnoreWebServerInternalPortNumber` configuration entry is not sufficient to assist the evaluation, `HttpBrowserFullCgiPath` can be set with an explicit answer.

Specify a relative path to the Content Server instance (`idc1` in the example that follows).

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"
- Section 4.2.2, "Clients"

Example

```
HttpBrowserFullCgiPath=http://localhost/idc1/idcplg
```

See Also

- "HttpIgnoreWebServerInternalPortNumber" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

HttpCgiPath

Retrieves the Content Server CGI path.

The following configuration settings are used to determine the value of this variable:

Variable	Description
UseSSL	When set to TRUE, the secure sockets layer (SSL) is used (*https* instead of *http*).
isAbsoluteCgi	Defines whether the complete URL is used instead of a relative path. This is an internal flag set by the Content Server instance and is not intended for user configuration.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the path as a string.

Example

Returned for a Content Server instance:

```
HttpCgiPath=/domain/idcplg
```

See Also

- HttpAbsoluteCgiPath
- HttpEnterpriseCgiPath
- UseSSL

HttpCommonRoot

Retrieves the URL of the *common* directory.

Multiple Content Server instances can share resources from one Content Server installation. This variable defines the URL path to the `DomainHome/ucm/cs/common/` directory of the Content Server instance whose resources are being shared. For example, the *HttpCommonRoot* defines the prefix to use for accessing the shared common directory where web applets are located.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the relative URL as a string. If the URL is external, the complete URL is returned rather than the relative path.

Example

Returned for a Content Server instance:

```
HttpCommonRoot=/domain/common/
```

See Also

- HttpHelpRoot
- HttpImagesRoot
- HttpWebRoot

HttpEnterpriseCgiPath

Retrieves the CGI path of a Content Server instance as a relative URL.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the Content Server CGI path as a string.

Example

Returned for a Content Server:

```
HttpEnterpriseCgiPath=/idcplg/idc_cgi_isapi-instance.dll
```

See Also

- HttpCgiPath

HttpHelpRoot

Retrieves the URL of the *help* directory.

Multiple Content Server instances can share online help files from one Content Server installation. This variable defines the URL path to the *DomainHome*/ucm/cs/weblayout/help/ directory of the Content Server instance whose help files are being shared.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the relative URL as a string. If the URL is external, the complete URL is returned rather than the relative path.

Example

Returned for a Content Server instance:

```
HttpHelpRoot=/domain/help/
```

See Also

- HttpCommonRoot
- HttpImagesRoot
- HttpWebRoot

HttpImagesRoot

Retrieves the URL of the *images* directory.

Multiple Content Server instances can share user interface images from one Content Server installation. This variable defines the URL path to the *DomainHome*/ucm/cs/weblayout/images/ directory of the Content Server instance whose image files are being shared.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the relative URL as a string. If the URL is external, the complete URL is returned rather than the relative path.

Example

Returned for a Content Server:

```
HttpImagesRoot=/domain/images/
```

See Also

- HttpCommonRoot
- HttpHelpRoot
- HttpWebRoot

HttpLayoutRoot

Retrieves the URL of the folder containing the current layout files. This is useful if you have additional JavaScript resources specific to the current layout.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

HttpRelativeAdminRoot

Retrieves the relative URL of the Admin Server for a Content Server instance.

Defaults to `/cs-admin/` during installation.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the relative URL as a string.

Example

Returned for a Content Server:

```
HttpRelativeAdminRoot=/cs-admin/
```

HttpRelativeWebRoot

Defines the web server root directory as a relative URL.

- A relative root such as */domain/* is used rather than a full root such as *http://www.mycomputer.com/domain/*.
- Returns the relative web root directory as a string.
- There is no default value.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.10, "Directories and Paths"
- Section 4.2.25, "Web Servers"

Location

- System Properties, Internet tab, **Http Relative Web Root**
- Content Server: `IntradocDir/config/config.cfg`

Example

As a configuration setting, defines the relative web root:

```
HttpRelativeWebRoot=/domain/
```

As Idoc Script, returns the relative web root as a string:

```
<$HttpRelativeWebRoot$>
```

See Also

- "HttpAbsoluteWebRoot" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

HttpServerAddress

Defines the web server address as a partial URL.

A partial URL such as `mycomputer` is used rather than a full address such as `http://www.mycomputer.com/`.

Returns the web server address as a string.

There is no default value.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.10, "Directories and Paths"
- Section 4.2.25, "Web Servers"

Location

- System Properties, Internet tab, HTTP Server Address
- Content Server: *`IntradocDir`*`/config/config.cfg`

Example

As a configuration setting, defines the web server address:

```
HttpServerAddress=mycomputer
```

As Idoc Script, returns the web server address as a string:

```
<$HttpServerAddress$>
```

See Also

- "IdcCommandServerHost " in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- "IntradocServerHostName" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

HttpSystemHelpRoot

Similar to the `HttpHelpRoot` variable except this variable returns the path to the help files for the default system language.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the path to the help files.

Example

```
HttpSystemHelpRoot=/domain/help/
```

See Also

- HttpCommonRoot
- HttpHelpRoot
- HttpImagesRoot
- HttpWebRoot

HttpWebRoot

Retrieves the URL path of the `weblayout/` directory.

Multiple Content Server instances can share resources from one Content Server installation. This variable defines the URL path to the *DomainHome*`/ucm/cs/weblayout/` directory of the Content Server instance whose resource files are being shared.

The following configuration settings determine the value of this variable:

Variable	Description
UseSSL	When set to TRUE, the secure sockets layer (SSL) is used (*https* instead of *http*).
isAbsoluteWeb	Defines whether the complete URL is used instead of a relative path. This is an internal flag set by the Content Server instance and is not intended for user configuration.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.10, "Directories and Paths"

Output

Returns the relative URL as a string. If the URL is external, the complete URL is returned rather than the relative path.

Example

Returned for a Content Server:

`HttpWebRoot=/`*`domain`*`/`

See Also

- UseSSL
- HttpCommonRoot
- HttpHelpRoot
- HttpImagesRoot

HTTP_ACCEPT

Retrieves a list of content types that the browser can accept.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a list of content types as a comma-delimited string.

Example

As information output on a page or to a log:

```
HTTP_ACCEPT=text/xml,application/xml,application/xhtml+xml,text/html;
q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;
q=0.2,text/css,*/*;q=0.1
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_ACCEPT$>
<!--statement-->
```

HTTP_ACCEPT_ENCODING

Retrieves a list of compression encodings that the browser supports.

As of Content Server version 7.0, HTML files are compressed for delivery. GZIP compression is the default.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a list of compression encodings as a comma-delimited string.

Example

As information output on a page or to a log:

```
HTTP_ACCEPT_ENCODING=gzip, deflate, compress;q=0.9
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_ACCEPT_ENCODING$>
<!--statement-->
```

HTTP_ACCEPT_LANGUAGE

Retrieves a list of ISO languages that are set for the browser.

This variable might be useful for estimating which user locale to set for new or guest users.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a list of languages as a comma-delimited string.

Example

As information output on a page or to a log:

```
HTTP_ACCEPT_LANGUAGE=en-us,en;q=0.50
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_ACCEPT_LANGUAGE$>
<!--statement-->
```

HTTP_COOKIE

Retrieves the name/value pair of the cookie in the HTTP request header.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a string in the format `name1=string1; name2=string2`.

Example

As information output on a page or to a log:

```
HTTP_COOKIE=IntradocAuth=Basic; IntrdocLoginState=1
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_COOKIE$>
<!--statement-->
```

HTTP_HOST

Retrieves the name of the web server.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.13, "Idoc Script"

Output

Returns the web server name as a string.

Example

As information output on a page or to a log:

```
HTTP_HOST=centralserver
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_HOST$>
<!--statement-->
```

HTTP_INTERNETUSER

Retrieves the CGI parameter that the web server security filter passes to the system so that the system can set the user.

If this variable is not set, the value of the REMOTE_USER variable is used.

If HTTP_INTERNETUSER and REMOTE_USER variables are not set, the user is anonymous.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the user name as a string.

Example

As information output on a page or to a log:

```
HTTP_INTERNETUSER=admin
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_INTERNETUSER$>
<!--statement-->
```

HTTP_REFERER

Retrieves the complete URL of the referenced directory on the local server.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the complete URL as a string.

Example

As information output on a page or to a log:

```
HTTP_REFERER=http://centralserver/domain/
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_REFERER$>
<!--statement-->
```

HTTP_USER_AGENT

Retrieves the client browser type, version number, library, and platform for which the browser is configured.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a string in the format `software/version (library) (platform)`.

Example

As information output on a page or to a log:

```
HTTP_USER_AGENT=Mozilla/4.7 [en] (WinNT; U)
```

As part of an Idoc Script statement or evaluation:

```
<$if HTTP_USER_AGENT$>
<!--statement-->
```

IdcAuthExtraRequestParams

Specifies values for the web server plug-in to get from its local data and send in the header of its requests.

For example, if you have a plug-in to the web server filter getting some attributes for a user and placing them into a local value, you could specify the name of the attributes in `IdcAuthExtraRequestParams`, and the data would be passed to the Content Server instance.

This setting can also be used to access extra HTTP header values using Idoc Script. These variables will become part of the Data Binder environment variable set.

- The names in the `IdcAuthExtraRequestParams` list must match the header names without regard for case (case insensitive match).
- When used as Idoc Script variables, the names in the `IdcAuthExtraRequestParams` must match exactly (case sensitive match).

There is no default value.

For information about IdcCommand, see Chapter 27, "Using the IdcCommand Utility to Access Content Server."

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.25, "Web Servers"

Location

IntradocDir`/config/config.cfg`

Example

The following setting tells the IIS web server plug-in to send the local value of HTTP_REFERER in the header (IIS automatically adds the `HTTP_` prefix):

```
IdcAuthExtraRequestParams=referer
```

idocTestForInclude()

This function tests to find out if a dynamichtml resource exists.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.11, "Dynamicdata"

Parameters

This function has one parameter: `includeName` is the name of the dynamichtml resource.

Output

If the resource exists, then it returns TRUE.

Example

```
<$if idocTestForInclude("std_page_begin")$>
    <$include std_page_begin$>
<$endif$>
```

inc()

Adds the code from an include resource to the page.

This function does the same thing as the *include* keyword, except that it can take an Idoc Script variable as the parameter. For more information, see Section 3.4.1, "Keywords Versus Functions."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is a variable that defines the name of an include.

Output

Displays the code that is defined in the specified include resource.

Example

Say you wanted to execute some Idoc Script for some, but not all, of your custom metadata fields. You could dynamically create includes based on the field names (such as *specific_include_xComments*) by executing this Idoc Script:

```
<$loop DocMetaDefinition$>
    <$myInclude = "specific_include_" & dName$>
    <$exec inc(myInclude)$>
<$endloop$>
```

Note the use of the `exec` keyword, which suppresses the output of the include specified by the `inc` function. If you don't use *exec* before the `inc` function, the HTML inside the specified include will be displayed on the page.

Note that if the `specific_include_xComments` does not exist, this code will not throw an error because the output is not being displayed.

See Also

- include
- setResourceInclude()

incDynamicConversionByRule()

Returns the results of converting the `LatestReleased` revision of a document using the template and layout associated with a particular Dynamic Converter conversion rule.

Fragments created through forced conversions can be referenced directly using this function.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.6.2, "Dynamic Converter"

Parameters

Takes two parameters:

- The first parameter is the Content ID of a document.
- The second parameter is the name of a conversion rule.

Output

Retrieves the specified document converted according to the template and layout defined for the specified conversion rule.

Example

Retrieves the converted rendition of the content item with a Content ID of `PhoneList`, converted according to conversion rule `RuleA`:

```
<$incDynamicConversionByRule("PhoneList","RuleA")$>
```

See Also

- ForcedConversionRules

incGlobal()

Includes the entire contents of a text file in the display of the current page.

This function is used to generate the default portal page.

A global include file is a text file that contains HTML code, Idoc Script code, or both. It is loaded on server startup.

Global include files must have the `.inc` extension and must be located in the *DomainHome*`/ucm/cs/data/pages/` directory.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is the name of a global include file without the `.inc` file extension.

Output

Displays the code that is defined in the specified global include file.

Example

When you change the portal page using the Update Portal function in the Web Layout Editor, a global include file called `portal_message.inc` is created to contain your modified text. This file is then referenced in the `pne_home_page.htm` template file as follows:

```
<$incGlobal("portal_message")$>
```

include

Adds the code from an include resource (defined by `<@dynamichtml include_name@>`) to the page.

This keyword is the most commonly used command in Idoc Script, as it allows you to reuse small chunks of code on multiple template pages.

For more information, see Section 3.3.1, "Includes."

Type and Usage

- Section 3.4, "Special Keywords"
- Section 3.2.2, "Idoc Script Comments"

Parameters

The only parameter is the name of the include.

Output

Displays the code that is defined in the specified include resource.

Example

The following includes are used to create the beginning and end of most Content Server web pages. These includes are defined in the following file:

IdcHomeDir`/resources/core/standard_page.htm`

```
<$include std_html_head_declarations$>
<$include body_def$>
<$include std_page_begin$>
    Hello World!
<$include std_page_end$>
```

See Also

- Keywords Versus Functions
- setResourceInclude()
- inc()

incTemplate()

Adds the contents of a Content Server template to a page, after evaluating any Idoc Script.

You can use this function to include the content of an entire template. However, this usage is discouraged because resource includes are usually sufficiently flexible to support all requirements for the sharing of Idoc Script between pages.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is the name of a Content Server template file without the `.htm` file extension.

Output

Displays the code that is defined in the specified template.

Example

Retrieves the `new_look` template file.

```
<$incTemplate("new_look")$>
```

indexerSetCollectionValue()

This function modifies the logic of the search indexer. This function is designed to allow calculation of cumulative statistics about the index collection.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.3, "Content Items"

Parameters

Takes two parameters:

- The first parameter is the name of the flag used.
- The second parameter is the value attributed to the flag.

Any name and value can be used. These values become available the next time the resource include is executed for the index collection.

InstanceDescription

Defines a description for the instance.

The instance description is used in the Content Server interface.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"

Location

- System Properties, Server tab, Instance Description
- *IntradocDir*/config/config.cfg

Example

As a configuration entry:

```
InstanceDescription=Master_on_Server1
```

As Idoc Script, returns the server instance description as a string:

```
<$InstanceDescription$>
```

See Also

- "IDC_Name" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- "InstanceMenuLabel" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

isActiveTrace()

Checks if a section is being traced in the core.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.9, "Debugging"

Output

Returns active trace results.

Example

```
<$isActiveTrace()$>
```

See Also

- isVerboseTrace

isCheckin

Specifies if the current page is a checkin page.

When set to TRUE, the current page is identified as a checkin page.

When set to FALSE, the current page is not identified as a checkin page.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is a checkin page.
- Returns FALSE if the page is not a checkin page.

Example

Set at the top of a page:

```
isCheckin=1
```

Used as script:

```
<$if isCheckin or isUpdate or isQuery$>
    <$isFormSubmit = 1$>
    <$if not isQuery$>
        <$isEditMode = 1$>
        <$formName = "Checkin"$>
    <$endif$>
    <$else$>
        <$isFormSubmit = ""$>
        <$isEditMode = ""$>
<$endif$>
```

IsCheckinPreAuthed

Checks if a checkin application pre-authorized the current checkin by getting a security token.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the checkin is pre-authorized.
- Returns FALSE if the checkin is not pre-authorized.

Example

Checks for checkin pre-authorization:

```
<$IsCheckinPreAuthed$>
```

isComponentEnabled

Checks if the defined component in enabled.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.5, "Content Server"

Parameters

The only parameter is the component name.

Output

- Returns TRUE if the defined component is enabled.
- Returns FALSE if not enabled.

Example

```
<$isComponentEnabled("Folders")$>
```

IsContributor

Used to evaluate whether a user is a contributor. Generally used to determine whether to show special links on a page (for example, `std_page` uses it to decide whether to display the Content Manager link).

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.5, "Content Server"

Output

- Returns TRUE if the user is a contributor.
- Returns FALSE if the user is not a contributor.

Example

Used to evaluate whether a user is a contributor:

```
<$if IsContributor$>
```

IsCriteriaSubscription

Evaluates whether a subscription to the content item is criteria-based rather than based on the Content ID (`dDocName`).

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.5, "Content Server"

Output

- Returns TRUE if the subscription is criteria-based.
- Returns FALSE if the subscription is to a particular content item.

Example

Evaluates whether subscription is criteria based:

```
<$IsCriteriaSubscription$>
```

IsCurrentNav

Checks if the page currently being displayed is the same as the page being looped over while building the Next/Previous navigation on search results pages.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.20, "Searching"

Output

- Returns TRUE if the navigation loop is at the same page as the current page.
- Returns FALSE if the navigation loop is not at the same page as the current page.

Example

If the navigation loop is at the current page, the page number is displayed as plain text. If the navigation loop is not at the current page, the page number is displayed as a hypertext link:

```
<$loop NavigationPages$>
<$if IsCurrentNav$>
    <$HeaderPageNumber$>
<$else$>
    <a href="<$strRemoveWs(inc("searchapi_navigation_specific_page"))$>">
    <$HeaderPageNumber$></a>
<$endif$>
<$endloop$>
```

isDocPage

Specifies if the current page is a page that displays metadata (such as search, checkin, and content information pages).

When set to TRUE, the current page is identified as a document page.

When set to FALSE, the current page is not identified as a document page.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is a document page.
- Returns FALSE if the page is not a document page.

Example

Set at the top of a page:

```
isDocPage=1
```

Used as script:

```
<$if isDocPage$>
    <!--statement-->
<$endif$>
```

IsDynamic

Checks if the page is presented dynamically to the user.

Most pages viewed by the user are dynamic. However, some static pages are designed to be delivered to the user without changes. Examples are the guest portal page and the content of some auto generated e-mails.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is being presented dynamically to the user.
- Returns FALSE if the page is static or cannot be displayed.

Example

Evaluates whether the page is presented dynamically:

```
<$if IsDynamic$>
    <a href="<$redirect$>">
<$endif$>
```

IsDynamicConverterEnabled

Enables Dynamic Converter.

This value is set to FALSE during initial WebCenter Content installation.

This value is set to TRUE during Dynamic Converter installation.

Default is FALSE.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.6.2, "Dynamic Converter"

Location

- System Properties, Options tab, Enable Dynamic Converter
- Admin Server, General Configuration, Enable Dynamic Converter
- *IntradocDir*/config/config.cfg

Example

As a configuration setting, enables Dynamic Converter functionality:

```
IsDynamicConverterEnabled=true
```

As Idoc Script, used to evaluate whether dynamic converter functionality is enabled:

```
<$if IsDynamicConverterEnabled and isTrue(IsDynamicConverterEnabled)$>
    <tr>
      <td align="center" width="75">
        <a href="<$HttpCgiPath$>?IdcService=GET_TEMPLATE_CONVERSIONS">
          <img src="<$HttpImagesRoot$><$admin_conversion_image$>" border="0"
        width="<$adminImageWidth$>" height="<$adminImageHeight$>"></a>
      </td>
      <td width="10"> 
      </td>
      <td width="245"><span class=largeTableEntry>
        <a href="<$HttpCgiPath$>?IdcService=GET_TEMPLATE_CONVERSIONS">
        <$lc("wwTemplateConversions")$></span></a>
      </td>
    </tr>
<$endif$>
```

isEditMode

Specifies if metadata fields on the current page can be edited.

This variable is set on checkin and content information update pages.

When set to TRUE, metadata fields on the current page can be edited.

When set to FALSE, metadata fields on the current page cannot be edited.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if metadata fields on the current page can be edited.
- Returns FALSE if metadata fields on the current page cannot be edited.

Example

Set at the top of a page:

```
isEditMode=1
```

Used as script:

```
<$if isCheckin or isUpdate or isQuery$>
    <$isFormSubmit = 1$>
    <$if not isQuery$>
        <$isEditMode = 1$>
        <$formName = "Checkin"$>
    <$endif$>
    <$else$>
        <$isFormSubmit = ""$>
        <$isEditMode = ""$>
<$endif$>
```

IsEditRev

Checks whether the current revision is in an Editor step in a workflow.

When set to TRUE, reviewers can check out, edit, and check in the revision.

This variable is set by defining a workflow step as an Editor step in the Workflow Admin tool. This variable is evaluated on the checkin page.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.26, "Workflow"

Output

- Returns TRUE if the current revision is in an Editor step in a workflow.
- Returns FALSE if the current revision is not in an Editor step in a workflow.

Example

Provides workflow details:

```
<$if IsEditRev$>
    addCheckinValue("IdcService", "WORKFLOW_CHECKIN");
if (form.isFinished.checked)
    addCheckinValue("isFinished", form.isFinished.value);
<$else$>
    addCheckinValue("IdcService", "CHECKIN_SEL");
<$endif$>
```

isExcluded

Completely excludes the metadata field from the page.

Type and Usage

- Section 4.1.5.2.2, "Common Field Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the metadata field is excluded from the page.
- Returns FALSE if the metadata field is not excluded from the page.

Example

Excludes the current metadata field from the page:

```
<$isExcluded=1$>
```

Sets a custom variable depending on the active value of *isExcluded* for the current metadata field:

```
<$isCustomExcluded = getValue("#active", fieldName & ":isExcluded")$>
$isCustomRelocated = getValue("#active", fieldName & ":isRelocated")$>
<$if isCustomExcluded or (isCustomRelocated and not isRelocated) or isExcluded or
(isFieldHidden and not isFormSubmit)$>
    <$isFieldExcluded = 1$>
<$endif$>
```

See Also

- is Field Excluded
- isHidden
- isInfoOnly
- isRelocated
- optionListScript

IsExternalUser

Checks if the user is accessing the Content Server instance from an external system, such as a LDAP system.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.24, "Users"

Output

- Returns `TRUE` if the user is accessing the Content Server instance from an external system.
- Returns `FALSE` if the user is accessing the Content Server instance directly.

Example

Checks if user is at an external location:

```
<$IsExternalUser$>
```

IsFailedConversion

Checks if the Oracle WebCenter Content: Inbound Refinery system has failed to convert the content item.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.6.1, "Inbound Refinery"

Output

- Returns TRUE if the conversion process failed.
- Returns FALSE if no conversion failure was detected.

Example

Displays text if the conversion process was not complete:

```
<$if IsFailedConversion$>
    <p><font face="arial" size="2">
    The Refinery was unable to complete the conversion process.</p>
<$endif$>
```

IsFailedIndex

Checks if the Indexer has failed to index the content item.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.14, "Indexing"

Output

- Returns TRUE if the Indexer was unable to index the content item.
- Returns FALSE if the content item was indexed successfully.

Example

Displays text if the content item was not indexed:

```
<$if IsFailedIndex $>
    <p><font face="arial" size="2">
    Unable to index content item.</p>
<$endif$>
```

sawflies()

Checks if a string or expression evaluates to FALSE.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is the string or expression to be evaluated.

Output

If the parameter is a string:

- Returns TRUE if the string begins with F, f, N, n, or is 0.
- Returns FALSE if the string begins with any other character.

If the parameter is not a string:

- Returns TRUE if the value is 0.
- Returns FALSE if the value is not 0.

Example

Evaluates the string `false` and returns TRUE (1):

```
<$isFalse("false")$>
```

Evaluates that the integer five is greater than one and returns FALSE (0):

```
<$isFalse(5>1)$>
```

Evaluates the result of the equation as zero and returns TRUE (1):

```
<$isFalse(1-1)$>
```

Evaluates the string equality statement as true and returns FALSE (0):

```
<$isFalse(strEquals("abc","abc"))$>
```

See Also

- isTrue()

is Field Excluded

Specifies that the metadata field is excluded.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field is excluded.
- Returns FALSE if the field is not excluded.

Example

Determines if the metadata field is excluded:

```
<$if isCustomExcluded or (isCustomRelocated and not isRelocated) or isExcluded or
(isFieldHidden and not isFormSubmit)$>
    <$isFieldExcluded = 1$>
<$endif$>
```

See Also

- isExcluded
- isFieldHidden
- isFieldInfoOnly
- isFieldMemo

isFieldHidden

Specifies that the metadata field is hidden.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field is hidden.
- Returns FALSE if the field is not hidden.

Example

Determines if the metadata field is hidden:

```
<$if isHidden or isCustomHidden$>
    <$isFieldHidden = 1$>
<$else$>
    <$isFieldHidden = ""$>
<$endif$>
```

See Also

- isHidden
- is Field Excluded
- isFieldInfoOnly
- isFieldMemo

isFieldInfoOnly

Specifies that the metadata field is an information only field.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field is an information only field.
- Returns FALSE if the field is not an information only field.

Example

Determines if the metadata field is information only:

```
<$if isInfo or isCustomInfo or isFieldHidden or isInfoOnly$>
    <$isFieldInfoOnly = 1$>
<$else$>
    <$isFieldInfoOnly = ""$>
<$endif$>
```

See Also

- isInfoOnly
- is Field Excluded
- isFieldHidden
- isFieldMemo

isFieldMemo

Specifies that the metadata field is a memo field.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field is a memo field.
- Returns FALSE if the field is not a memo field.

Example

Determines if the field is a memo field:

```
<@dynamichtml compute_std_entry_type_info@>
<$if not hasOptionList and #active.fieldType like "Memo"$>
    <$isFieldMemo = 1$>
<$else$>
    <$isFieldMemo = ""$>
<$endif$>
<@end@>
```

See Also

- is Field Excluded
- isFieldHidden
- isInfoOnly

IsFilePresent

Checks if the page currently being displayed is for the revision being looped over while building the Revision History table on a content information page.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

- Returns TRUE if the loop is at the same revision as the current revision.
- Returns FALSE if the loop is not at the same revision as the current revision.

Example

If the loop is at the current revision, the revision number is displayed as plain text. If the loop is not at the current revision, the revision number is displayed as an active button.

```
<@dynamichtml doc_rev_info@>
<$if IsFilePresent$>
  <td width=10% align=center><span class=strongHighlight><$dRevLabel$></span></td>
  <td nowrap width=30%><span class=strongHighlight><$dInDate$></span></td>
  <td nowrap width=30%><span class=strongHighlight>
    <$if dOutDate$><$dOutDate$><$else$><$lc("wwNone")$><$endif$></span></td>
      <td width=20%><span class=strongHighlight>
      <$rptDisplayMapValue("StatusList", dStatus)$></span></td>
    <$else$>
  <td width=10%><form action="<$HttpCgiPath$>" method=GET>
  <input type=hidden name=IdcService value="DOC_INFO">
  <input type=hidden name=dID value="<$dID$>">
  <input type=hidden name=dDocName value="<$dDocName$>">
  <input type=submit value="       <$dRevLabel$>      ">
  </form></td>
...
<@end@>
```

isFormSubmit

Specifies if the current page is a submittable HTML form.

When set to TRUE, the current page is a submittable HTML form.

When set to FALSE, the current page is not a submittable HTML form.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is a submittable HTML form.
- Returns FALSE if the page is not a submittable HTML form.

Example

Set at the top of a page:

```
isFormSubmit=1
```

Adds a colon after the field captions if the page is not a submittable HTML form:

```
<@dynamichtml std_field_caption@>
    <span class=<$fieldCaptionStyle$>>
    <$fieldCaption$><$if not isFormSubmit$><$":"$><$endif$>
    </span>
<@end@>
```

IsFullTextIndexed

Checks if the Indexer has full-text indexed the content item.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.14, "Indexing"

Output

- Returns `TRUE` if the content item has been full-text indexed.
- Returns `FALSE` if the content item has not been full-text indexed.

Example

Provides a specified URL if the content item is full-text indexed:

```
<$if IsFullTextIndexed$>
    <a href="<$redirect$>">
<$endif$>
```

isHidden

Hides the metadata field from the user but includes the field as hidden data on the page.

Type and Usage

- Section 4.1.5.2.2, "Common Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the metadata field is hidden on the page.
- Returns FALSE if the metadata field is not hidden the page.

Example

Hides the current metadata field from the user:

```
<$isHidden=1$>
```

Sets a custom variable depending on the active value of *isHidden* for the current metadata field:

```
<$isCustomHidden = getValue("#active", fieldName & ":isHidden")$>
    <$if isHidden or isCustomHidden$>
<$isFieldHidden = 1$>
<$else$>
    <$isFieldHidden = ""$>
<$endif$>
```

See Also

- isExcluded
- isFieldHidden
- isInfoOnly
- isRelocated
- optionListScript

isInfo

Specifies if the current page is an information-only page.

When set to TRUE, the current page is identified as an information-only page.

When set to FALSE, the current page is not identified as an information-only page.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is an information-only page.
- Returns FALSE if the page is not an information-only page.

Example

Set at the top of a page:

```
isInfo=1
```

Used as script:

```
<$if isInfo$>
    <$captionFieldWidth="30%"$>
    <$captionEntryWidth="70%"$>
<$elseif isEditMode$>
    <$captionFieldWidth="20%"$>
    <$captionEntryWidth="80%"$>
<$endif$>
```

isInfoOnly

Displays the metadata field as information only, not as an input field.

Type and Usage

- Section 4.1.5.2.2, "Common Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the metadata field is displayed as information only.
- Returns FALSE if the metadata field can be edited.

Example

Displays the current metadata field as information only:

```
<$isInfoOnly=1$>
```

Sets a custom variable depending on the active value of *isInfoOnly* for the current metadata field:

```
<<$isCustomInfo = getValue("#active", fieldName & ":isInfoOnly")$>
<$if isInfo or isCustomInfo or isFieldHidden or isInfoOnly$>
    <$isFieldInfoOnly = 1$>
<$else$>
    <$isFieldInfoOnly = ""$>
<$endif$>
```

See Also

- isExcluded
- isFieldInfoOnly
- isHidden
- isRelocated
- optionListScript

IsJava

Displays the local data of a Content Server web page.

This variable can be set as a flag on a page or as a parameter to a service call.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.9, "Debugging"

Output

When set to `TRUE`, returns the local data in the DataBinder, in HDA format.

Example

When included in the code on a Content Server web page, displays the local data of a page:

```
<$IsJava=1$>
```

When included in a Content Server URL, displays the local data for the New Checkin page:

```
http://myinstance.com/idcplg/idc_cgi_isapi-instance.dll?IdcService=CHECKIN_NEW_
FORM&IsJava=1
```

isLayoutEnabled()

Used to determine if a particular layout is installed and enabled.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.5, "Content Server"

Parameters

Takes one parameter, the ID of the layout.

Example

In the std_resources.htm file:

```
<$if isLayoutEnabled("Trays")$><$do Publish=1$><$endif$>
```

isLinkActive

Used by Idoc Script to determine whether to include the profile in a display of profiles for check-in or search based on specified conditions, such as a user's name. If the specified condition or conditions evaluate to true, the value of `IsLinkActive` is set to true (`1`) and the profile is included in the list of available profiles.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.5, "Content Server"
- Section 3.3.3.1, "Personalization Functions"

Parameters

There are no parameters, just the setting of `1` or `0` (true or false) based on the evaluated condition or conditions.

Example

Here is the default script for a condition set with a single clause. If the user name matches "`guest`", then `isLinkActive` is set to `1` and the profile is included in the list of available profiles for that user.

```
<$if getUserValue("dName") like "guest"$>
<$isLinkActive=1$>
<$endif$>
```

IsLocalSearchCollectionID

Checks if the content item is in the local search collection. Searches for the content item's Content ID in the local search collection.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.5, "Content Server"

Output

- Returns TRUE if the content item is in the local collection.
- Returns FALSE if the content item is not found in the local collection.

Example

Evaluates whether a content item is from a local collection:

```
<@dynamichtml searchapi_define_result_doc_parameters@>
<$exec IsLocalSearchCollection="1"$>
<$if not IsLocalSearchCollectionID$>
    <!--Collection has external ID-->
<$exec IsLocalSearchCollection=""$>
```

IsLoggedIn

Checks if the current user is logged in.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user has logged in.
- Returns FALSE if the user has not logged in.

Example

Checks whether the user is logged in and has an e-mail address before performing a function.

```
<@dynamichtml subscription_action_script@>
function allowSubscription(form)
    {<$if IsLoggedIn$>
    <$if IsUserEmailPresent$>
    <$else$>
…}
<@end@>
```

IsMac

Checks if the client browser is running on a Macintosh operating system.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.2, "Clients"

Output

- Returns TRUE if the client browser is running on a Mac.
- Returns FALSE if the client browser is not running on a Mac.

Example

Redirects to a different URL if the browser is running on a Mac:

```
<$if IsMac$>
    <a href="<$redirect$>">
<$endif$>
```

IsMaxRows

Checks if there are more results on a Work In Progress or Report page than the `MaxQueryRows` setting allows. For more information, see "MaxQueryRows" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the number of results is greater than the number of rows permitted.
- Returns FALSE if the number of results is less than the number of rows permitted.

Example

Returns an error message if the maximum number of rows is exceeded:

```
<$elseif IsMaxRows$>
<table border=0 cellpadding=1 cellspacing=1 width="100%">
    <tr>
    <span class=smallHighlight><$lc("wwOutputLimitedByMaxRows")$>
    </span>
</tr>
</table>
```

isMultiOption

Specifies that a metadata field option list allows multiple values to be selected.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field is a multiselect option list.
- Returns FALSE if the field is not a multiselect option list.

Example

Specifies that the field is a multiselect option list:

```
<$isMultiOption=1$>
```

Determines the type of option list:

```
<$if #active.fieldOptionListType like "*multi*"$>
    <$exec isMultiOption=1$>
<$elseif #active.fieldOptionListType like "access*"$>
    <$exec isAccessList=1$>
    <$exec isStrictList=1$>
<$elseif not (#active.fieldOptionListType like "*combo*")$>
    <!--Strict choice-->
    <$exec isStrictList=1$>
<$endif$>
```

See Also

- isStrictList

IsMultiPage

Checks if multiple pages are needed for search results. This variable depends on the number of rows displayed per page, which is defined by the `ResultCount` input variable (default is 25).

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.20, "Searching"

Output

- Returns TRUE if the number of search results is greater than the number of rows permitted per page.
- Returns FALSE if the number of search results is less than the number of rows permitted per page.

Example

Evaluates number of rows and determines if multiple pages are needed:

```
<$IsMultiPage$>
```

isNew

Checks if the content item is new or a revision. This variable is set to TRUE by the CHECKIN_NEW_FORM and CHECKIN_SIMILAR_FORM standard services.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the content item is new.
- Returns FALSE if the content item is a revision to an existing content item.

Example

If the content item is new, a specified service is performed:

```
<$if isNew$>
    <input type=hidden name=IdcService value="CHECKIN_NEW">
<$endif$>
```

If the content item is a revision, the original content item author is used:

```
<$if not isNew$>value="<$dDocAuthor$>"<$endif$>
```

If the content item is new, the default accounts for the current user are used:

```
<$if isNew$>
    <$defaultAccounts$>
<$endif$>
```

IsNotLatestRev

Checks if the revision is the last revision to be checked in. This is not necessarily the last released revision.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.3, "Content Items"

Output

- Returns TRUE if the content item is other than the latest revision to be checked in.
- Returns FALSE if the content item is the latest revision to be checked in.

Example

Checks for the latest revision:

```
<$IsNotLatestRev$>
```

See Also

- ClientControlled
- HasLocalCopy

IsNotSyncRev

Checks whether the file on the client computer matches the most current revision by performing a revision ID (`dID`) comparison.

This variable is generally used to display an error message when the local copy of a content item has not been updated to the latest revision.

This variable is used to interface with client-side products, such as Desktop.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.3, "Content Items"

Output

- Returns TRUE if revisions do not match.
- Returns FALSE if revisions match.

Example

Checks for a match with the latest revision and displays an error message:

```
<$if IsNotSyncRev$>
The local copy of this content item has not been updated to the latest revision.
    Use Get Native File or Check Out to update your local copy of <$dDocName$>.
<$endif$>
```

IsOverrideFormat

Enables users to select a different conversion format upon checkin.

When set to TRUE, a Format option list is displayed for the Primary File and Alternate File on the checkin page.

Default is FALSE.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.12, "Field Display"

Location

- System Properties, **Options** tab, **Allow override format on check in**
- Admin Server, General Configuration, **Allow override format on check in**
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting:

```
IsOverrideFormat=true
```

As Idoc Script, returns the value of the override format function:

```
<$IsOverrideFormat$>
```

IsPageDebug

Enables a trace of all includes and calls to the Idoc Script `eval` function.

The contents of the `eval` function and any dynamically assigned includes are also shown as part of the trace. The trace is indented by one + character per nested level of `include` or `eval` call. The trace also shows any error messages (without the nested location information) and the output of any calls to the Idoc Script `trace` function.

Note: This setting is not supported in Internet Explorer version 6.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.9, "Debugging"

Output

Results of the trace can be viewed through the debug menu toolbar options, which are accessed from the debug trace displayed at the bottom of the web page.

Click the debug trace to display the debug menu toolbar, then click any of the following options.

Element	Description
hide all	Hides any open debug popups.
idocscript trace	Displays a tree structure view of all includes being called.
request binder	Displays information on Local Data and ResultSets. The request binder is the DataBinder sent to the server, that is, the service name and any service-specific parameters. Click a heading to expand the view of detailed information.
response binder	Displays information on Local Data and ResultSets. The response binder is the DataBinder immediately after the service is processed but before the response page is generated. Click a heading to expand the view of detailed information.
final page binder	Displays information on Local Data and ResultSets. The final page binder reflects what was changed in the DataBinder while the page was being generated. Click a heading to expand the view of detailed information.
javascript log	Displays Logger Console window with default setting of Verbose. Select checkboxes to display the following options: info, warn, error, time, window, trace, global, schema, javascript, LogReader. Click **Pause** or **Clear** to control the speed and amount of information displayed on the Logger Console.

If you are interested in API-level interaction with Content Server, consult the request and response binders. For example, when you do a SOAP request or use other service-based APIs, the information in the final page binder is meaningless. When you append `IsJava=1` to the request, the response binder in HDA format is returned.

If you are interested in customization, page generation, and so on, you may want to consult the final page binder and, in some cases, determine how it differs from the response binder.

Example

Used as a configuration setting in the Content Server `config.cfg` file, so it applies to the entire server:

```
IsPageDebug=1
```

In a web browser, added to the end of the page's URL in the Address field:

```
&IsPageDebug=1
```

Used on a template page or in an Idoc Script include:

```
<$IsPageDebug=1$>
```

See Also

- eval()
- setResourceInclude()
- trace()

IsPromptingForLogin

Checks if the Content Server instance is set to prompt for login or if login is being handled programmatically.

This variable is set to `TRUE` in situations such as cookie login pages, where the last request failed because the user is not logged in yet.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.5, "Content Server"
- Section 4.2.21.2, "External Security"

Output

- Returns `TRUE` if the Content Server instance is set to prompt for login.
- Returns `FALSE` if login is being handled programmatically.

Example

Evaluates if server is set to prompt for login:

```
<$IsPromptingForLogin$>
```

isQuery

Specifies if the current page is a search page.

When set to TRUE, the current page is identified as a search page.

When set to FALSE, the current page is not identified as a search page.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is a search page.
- Returns FALSE if the page is not a search page.

Example

Set at the top of a page:

```
isQuery=1
```

Used as script:

```
<$if isCheckin or isUpdate or isQuery$>
    <$isFormSubmit = 1$>
    <$if not isQuery$>
<$isEditMode = 1$>
<$formName = "Checkin"$>
    <$endif$>
    <$else$>
<$isFormSubmit = ""$>
<$isEditMode = ""$>
<$endif$>
```

isRelocated

Excludes the metadata field unless the local variable is set to TRUE while the include is evaluated.

This variable is typically used to set a custom location for a metadata field. It allows a field to be defined on a page more than once, with the location of the field that is actually generated depending on the value of this variable.

Type and Usage

- Section 4.1.5.2.2, "Common Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the metadata field is included at that location on the page.
- Returns FALSE if the metadata field is excluded from that location on the page.

Example

Includes the current metadata field on the page:

```
<$isRelocated=1$>
```

Sets a custom variable depending on the active value of *isRelocated* for the current metadata field:

```
<$isCustomExcluded = getValue("#active", fieldName & ":isExcluded")$>
<$isCustomRelocated = getValue("#active", fieldName & ":isRelocated")$>
<$if isCustomExcluded or (isCustomRelocated and not isRelocated) or isExcluded or
(isFieldHidden and not isFormSubmit)$>
    <$isFieldExcluded = 1$>
<$endif$>
```

See Also

- isExcluded
- isHidden
- isInfoOnly
- optionListScript

IsRequestError

Checks if there is a request error condition present in the Content Server instance by evaluating the StatusCode variable.

If `StatusCode` is set to a negative numeric value (-1), there is a request error condition present in the Content Server instance.

The typical behavior when a request error condition is present is to abort the display of the current page and substitute an error page.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.9, "Debugging"
- Section 4.2.5, "Content Server"

Output

- Returns `TRUE` if there is a request error condition present in the Content Server instance (`StatusCode = -1`).
- Returns `FALSE` if there is no request error condition present on the Content Server instance (`StatusCode` is other than a negative numeric value).

Example

Evaluates the request error condition:

```
<$IsRequestError$>
```

See Also

- abortToErrorPage()
- executeService()
- StatusCode

isRequired

Specifies if a value is required for a metadata field.

When set to TRUE, the metadata field is required.

When set to FALSE, the metadata field is optional.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the metadata field is required.
- Returns FALSE if the metadata field is optional.

Example

Defines the *dDocTitle* metadata field as a required field:

```
<$fieldName = "dDocTitle", fieldCaption = lc("wwTitle"), isRequired = 1,
    fieldType = "BigText", requiredMsg = lc("wwSpecifyTitle")$>
```

See Also

- requiredMsg

IsSavedQuery

Checks if a query has been saved to the current user's portal navigation bar.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.20, "Searching"

Output

- Returns TRUE if the current query has been saved.
- Returns FALSE if the current query has not been saved or no query is found.

Example

Evaluates query status:

```
<$IsSavedQuery$>
```

IsSoap

Displays the local SOAP data of a Content Server web page.

This variable can be set as a flag on a page or as a parameter to a service call.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.9, "Debugging"

Output

When set to `TRUE`, returns the underlying SOAP code, in XML format.

Example

When included in the code on a Content Server web page:

```
<$IsSoap=1$>
```

When included in a Content Server URL:

```
http://myinstance.com/idcplg/idc_cgi_isapi-instance.dll?IdcService=CHECKIN_NEW_
FORM&IsSoap=1
```

isStrictList

Specifies that a metadata field option list does not allow multiple values to be selected.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the field is not a multiselect option list.
- Returns FALSE if the field is a multiselect option list.

Example

Specifies that the field is not a multiselect option list:

```
<$isStrictList=1$>
```

Determines the type of option list:

```
<$if #active.fieldOptionListType like "*multi*"$>
    <$exec isMultiOption=1$>
<$elseif #active.fieldOptionListType like "access*"$>
    <$exec isAccessList=1$>
    <$exec isStrictList=1$>
<$elseif not (#active.fieldOptionListType like "*combo*")$>
    <!--Strict choice-->
    <$exec isStrictList=1$>
<$endif$>
```

See Also

- isMultiOption

IsSubAdmin

Checks if the current user has subadministrator rights to at least one administrative application.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user has subadministrator rights.
- Returns FALSE if the user does not have subadministrator rights.

Example

Checks whether the user is logged in and has subadministrator rights before performing a function.

```
<@dynamichtml subscription_action_script@>
    function allowSubscription(form)
    {
    <$if IsLoggedIn$>
    <$if IsSubAdmin$>
    <$else$>
    …
    }
<@end@>
```

IsSun

Checks if the client browser is running on a Sun system.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.2, "Clients"

Output

- Returns TRUE if the client browser is running on Sun.
- Returns FALSE if the client browser is not running on Sun.

Example

Redirects to a different URL if the browser is running on Sun:

```
<$if IsSun$>
    <a href="<$redirect$>">
<$endif$>
```

IsSysManager

Checks if the current user has the `sysmanager` role, meaning the user has access to the Admin Server. This variable is usually used to conditionally display specific navigation links.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns `TRUE` if the user has the `sysmanager` role.
- Returns `FALSE` if the user does not have the role

isTrue()

Checks if a string or expression evaluates to TRUE.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is the string or expression to be evaluated.

Output

If the parameter is a string:

- Returns TRUE if the string begins with T, t, Y, y, or is 1.
- Returns FALSE if the string begins with any other character.

If the parameter is not a string:

- Returns TRUE if the value is not 0.
- Returns FALSE if the value is 0.

Example

Evaluates the string `yes` and returns TRUE (1):

```
<$isTrue("yes")$>
```

Evaluates that the integer five is greater than one and returns TRUE (1):

```
<$isTrue(5>1)$>
```

Evaluates the result of the equation as zero and returns FALSE (0):

```
<$isTrue(1-1)$>
```

Evaluates the string equality statement as true and returns TRUE (1):

```
<$isTrue(strEquals("abc","abc"))$>
```

See Also

- sawflies()

isUpdate

Specifies if the current page is a content information update page.

When set to TRUE, the current page is identified as a content information update page.

When set to FALSE, the current page is not identified as a content information update page.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the page is a content information update page.
- Returns FALSE if the page is not a content information update page.

Example

Set at the top of a page:

```
isUpdate=1
```

Used as script:

```
<$if isCheckin or isUpdate or isQuery$>
    <$isFormSubmit = 1$>
    <$if not isQuery$>
        <$isEditMode = 1$>
        <$formName = "Checkin"$>
    <$endif$>
    <$else$>
        <$isFormSubmit = ""$>
        <$isEditMode = ""$>
<$endif$>
```

isUploadFieldScript

Specifies that an include is being used inside JavaScript. It is used to determine how metadata fields are uploaded.

When set to TRUE, the include is being used inside JavaScript.

When set to FALSE, the include is being used inside JavaScript.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the include is being used inside JavaScript.
- Returns FALSE if the include is not being used inside JavaScript.

Example

Set in the include that defines the JavaScript wrapper:

```
<@dynamichtml std_upload_java_script@>
    <script language="JavaScript">
    // Java script for uploading.
    <$isUploadFieldScript = 1$>
    <$include std_upload_info_script$>
    <$isUploadFieldScript = ""$>
    </script>
<@end@>
```

Used as script:

```
<$if isUploadFieldScript$>
    <$defaultFieldInclude = "std_file_entry"$>
<$else$>
    <$defaultFieldInclude = "std_nameentry_row"$>
    <$fieldCaptionInclude = "std_field_caption"$>
    <$fieldEntryInclude = "std_file_entry"$>
<$endif$>
```

IsUploadSockets

Used by the Upload applet to determine whether the upload socket should be used. This is an internal flag and is not intended for user configuration.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.2, "Clients"
- Section 4.2.21.2, "External Security"

Output

- Returns TRUE if the upload socket is defined for use with the Upload applet.
- Returns FALSE if the upload socket should not be used.

Example

N/A

IsUserEmailPresent

Checks if an e-mail address is defined for the current user.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.24, "Users"

Output

- Returns TRUE if an e-mail address is defined for the user.
- Returns FALSE if an e-mail address is not defined for the user.

Example

Checks whether the user is logged in and has an e-mail address before performing a function.

```
<@dynamichtml subscription_action_script@>
    function allowSubscription(form)
    {
    <$if IsLoggedIn$>
    <$if IsUserEmailPresent$>
    <$else$>
    …
    }
<@end@>
```

isUserOverrideSet()

Enables users to check in content for other users.

This affects the Author option list on checkin pages. By default, only administrators are allowed to specify another user as the Author during checkin.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Parameters

The only parameter is the value TRUE or FALSE.

Output

Evaluates to TRUE if the user override is enabled.

Example

Enables users to check in content items with another user specified as the Author:

```
<$isUserOverrideSet(true)$>
```

See Also

- ExclusiveCheckout

isValidateFile()

Used as a parameter to the `Upload` applet. This variable verifies that the file to be uploaded exists. In order to be used, a component must be created that overwrites the applet definition as defined in the `std_multiupload_applet_definition` include in the std_page.htm resource file.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"

Example

Used as a configuration entry:

```
isValidateFile=true
```

isVerboseTrace

Checks if a section is being traced in the core. Verbose trace generates a full report.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.9, "Debugging"

Output

- Returns TRUE if tracing is set to Verbose.
- Returns FALSE if tracing is set to any other debug level.

Example

```
<$isVerboseTrace()$>
```

See Also

- isActiveTrace()

IsWindows

Checks if the client browser is running on a Windows operating system.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.2, "Clients"

Output

- Returns TRUE if the client browser is running on Windows.
- Returns FALSE if the client browser is not running on Windows.

Example

Redirects to a different URL if the browser is running on Windows:

```
<$if IsWindows$>
    <a href="<$redirect$>">
<$endif$>
```

IsWorkflow

Checks if the content item on a checkin page is in a workflow.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.26, "Workflow"

Output

- Returns TRUE if the content item is in a workflow.
- Returns FALSE if the content item is not in the workflow.

IsXml

Displays a raw dump of the HTML form when set to TRUE in a URL to a HCSP or HCSF file. This is useful for extracting form data or submitting forms from a remote application.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.9, "Debugging"

Output

- Returns the XML data contained in the form.

See Also

- IsJava
- IsSoap

isZoneSearchField

This function is used to specify a field for full text searching.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.20, "Searching"

Parameters

The name of the field to be checked.

js()

Formats a string for use in a "..." JavaScript literal string declaration.

This function performs string manipulation such as changing double quotes to single quotes.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string formatted for use in a JavaScript string declaration.

Example

Formats the string *variablestring* for use in a JavaScript string declaration:

```
<$js("variablestring")$>
```

jsFilename()

Used by schema. Encodes a string that may contain non-ASCII characters into the valid filename strings required for the operating system and Java Script (performs an encoding function).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

The only parameter is the string to be encoded.

Output

Returns an encoded string.

Example

```
<$jsFilename(fileName)$>
```

Json

Displays the local data of a Content Server web page.

This variable can be set as a flag on a page or as a parameter to a service call.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.9, "Debugging"

Output

When set to `TRUE`, returns the local data in the DataBinder in a JavaScript object, in compliance with the JSON (Javascript over network) specification.

Example

When included in the code on a Content Server web page, displays the local data of a page:

```
<$Json=1$>
```

When included in a Content Server URL, displays the local data for the New Checkin page:

```
http://myinstance.com/idcplg/idc_cgi_isapi-instance.dll?IdcService=CHECKIN_NEW_FORM&Json=1
```

lastEntryTs

Retrieves the timestamp from the last time the workflow step was entered.

This variable can be used to create conditional statements, but it should not be hard-coded or altered.

The last entry time is localized in the companion file and maintained in the key:

```
<step_name>@<workflow_name>.lastEntryTs
```

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.8, "Date and Time"

Output

Returns the timestamp in the format defined by dateCurrent.

Example

The following code defines a jump called `LastEntry`, which exits to the parent workflow and notifies the reviewers if the last time the step was entered was more than one week ago:

```
<$if parseDate(wfCurrentGet("lastEntryTs")) < parseDate(dateCurrent(-7))$>
    <$wfSet("WfJumpName","LastEntry")$>
    <$wfSet("WfJumpTargetStep",wfExit(0,0))$>
    <$wfSet("WfJumpEntryNotifyOff","0")$>
<$endif$>
```

lc()

Retrieves the value of a localization string based on the current user's locale.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.15, "Localization"

Parameters

Takes one required parameter and any number of optional parameters:

- The first parameter is the localization key (such as `apLabelHelp` or `wwMyString`).
- The optional parameters are expressions that are resolved according to arguments inside curly braces in the localized string (for example, `{1}`).

Output

- Returns the value of the localization string for the current user's locale.
- Returns the string ID (such as *wwMyString*) if the value is not found.

Example

Retrieves the options for metadata search operators on a search page:

```
<select name="op" >
    <option value="Contains"><$lc("wwContains")$>
    <option value="Matches"><$lc("wwMatches")$>
    <option value="Starts"><$lc("wwStarts")$>
    <option value="Ends"><$lc("wwEnds")$>
    <option selected value="Substring"><$lc("wwSubstring")$>
</select>
```

Sets the subject line for a *workflow started* notification e-mail. If the name of the workflow (*dWfName*) is *Marketing*, the resulting value in English is *Workflow 'Marketing' has been started*.

```
<@dynamichtml wf_started_subject@>
    <$lc("wwWfStarted", dWfName)$>
<@end@>
```

lcCaption()

Function that wraps a string into a caption. This will usually place a colon to the right of the string. For right-to-left reading languages, such as Hebrew and Arabic, the colon is placed on the left of the string.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.15, "Localization"

Parameters

Takes one required parameter:

- The string to be wrapped.
- optional parameters are expressions that are resolved according to arguments inside curly braces in the localized string (for example, `{1}`).

LmDefaultLayout()

Defines the default layout to use (for example, Top Menus or Trays).

The default installation value is `Trays`.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"

Location

IntradocDir`/config/config.cfg`

Example

```
LmDefaultLayout=Top Menus
```

LmDefaultSkin()

Defines the default skin to use (for example, Oracle, Oracle2).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.5, "Content Server"

Location

IntradocDir/config/config.cfg

Example

```
LmDefaultSkin=Oracle
```

lmGetLayout()

Retrieves the layout selected by the user (for example, Top Menus or Trays).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.5, "Content Server"

Parameters

None.

Example

```
<$lmGetLayout()$>
```

See Also

- lmGetSkin()

lmGetSkin()

Retrieves the skin selected by the user.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.5, "Content Server"

Parameters

None.

Example

```
<$lmGetSkin()$>
```

See Also

- lmGetSkin()

loadCollectionInfo()

Loads metadata for a search collection.

Used by the search service to load metadata about a search collection.

This function is not intended for user configuration.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.20, "Searching"

Parameters

Takes one parameter, the name of the search collection to be loaded.

Output

None.

Example

Loads search collection information:

```
<$loadCollectionInfo(collection_name)$>
```

loadDocMetaDefinition()

Loads the `DocMetaDefinition` database table into the active data as a ResultSet.

After the `DocMetaDefinition` database table is loaded, it can be looped on.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

None.

Output

None.

Example

Loads the `DocMetaDefinition` table into the active data as a ResultSet.

```
<$loadDocMetaDefinition()$>
```

loadDocumentProfile()

Loads a content profile as specified by the trigger value for page presentation. This is called during the presentation of Search, Check In, Info, and Update pages. This function is called on request and the page on which it is called determines the action (for example, search, or info).

This information is dependent on context. The `isCheckin`, `isUpdate`, `isQuery`, or `isInfo` variables are set in local data and direct the `loadDocumentProfile` function to the page to be loaded and presented.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

None.

Output

None.

Example

```
<$loadDocumentProfile()$>
```

See Also

- utLoadDocumentProfiles()

loadSchemaData()

Loads the data from a schema ResultSet into the local data of the current data binder.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.19, "Schemas"

Parameters

This function can take zero, one, or two parameters.

- If passed zero arguments, it fills the data binder with the data in the active ResultSet's row. Returns no value.
- If passed one argument, the argument is the name of the ResultSet to get the values from the current row. Returns no value.
- If passed two arguments, the first argument is the ResultSet name and the second argument is the key identifying the data object to load. Returns 0 if the data does not exist or 1 if it does exist. The use is True() or False() to conditionally execute scripts based on the return value.

Example

```
<$loadSchemaData()$>
```

See Also

- clearSchemaData()

loadSearchOperatorTables()

Loads mappings between the full set of operator names and the search syntax. Used on the Search page.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

None.

Output

Returns expressions associated with operator names (search operator map).

Example

```
<$loadSearchOperatorTables()$>
```

loadUserMetaDefinition()

This function loads the custom user metadata definition. This exists in the `UserMetaDefinition` table which is placed on the page as a ResultSet. This is used on LDAP administration pages to help administers map user metadata to the Content Server instance.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

None.

localPageType

This function returns the page type for a page in the library folders. This may be a directory or a query page. This function is usually used by the Layout Manager API. When `Trays` is the selected layout, it is used to construct a tree view of the library's pages.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Parameters

Takes one parameter, the link data used to construct the tree view.

MajorRevSeq

Defines the major sequence for revision numbers.

Returns the major revision label sequence (returns the value of the configuration setting).

Returns a string.

Default is *1-99*.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"

Location

- System Properties, Options tab, Major Revision Label Sequence
- Admin Server, General Configuration, Major Revision Label Sequence
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting:

```
MajorRevSeq=A-Z
```

As script, returns the value of the configuration setting:

```
<$MajorRevSeq$>
```

See Also

- MinorRevSeq

MaxCollectionSize

Defines the number of files to be passed to the Indexer in one batch.

Valid range is 1 to 65535.

A value of 2000 is recommended for large index collections. Lower values will result in inefficient indexing performance.

Returns the number of files per batch.

Default is 25.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.14, "Indexing"

Location

- Repository Manager, Indexer tab, Configure, Content Items Per Indexer Batch
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting, defines the batch size:

```
MaxCollectionSize=25
```

As script, returns the value from configuration files:

```
<$MaxCollectionSize$>
```

maxLength

Sets the maximum number of characters allowed in a metadata field.

Type and Usage

- Section 4.1.5.2.2, "Common Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the maximum length of the field.

Example

Sets the maximum length of the field to 100 characters:

```
<$maxLength=100$>
```

Specifies a custom field length based on the length of the current field:

```
<$customFieldLength = getValue("#active", fieldName & ":maxLength")$>
```

MinorRevSeq

Defines the minor sequence for revision numbers.

Returns the minor revision label sequence (returns the value of the configuration setting).

Returns a string.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"

Location

- System Properties, Options tab, Minor Revision Label Sequence
- Admin Server, General Configuration, Minor Revision Label Sequence
- *IntradocDir*/config/config.cfg

Example

As a configuration setting:

```
MinorRevSeq=a-c
```

As script, returns the value of the configuration setting:

```
<$MinorRevSeq$>
```

See Also

- MajorRevSeq

MSIE

Checks whether the client browser is Microsoft Internet Explorer.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.2, "Clients"

Output

- Returns TRUE if the client browser is Internet Explorer.
- Returns FALSE if the client browser is not Internet Explorer.

Example

Redirects to a different URL if the browser is anything other than Internet Explorer:

```
<$if not MSIE$>
    <a href="<$redirect$>">
<$endif$>
```

MultiUpload

Enables the multiple file Upload Applet.

When set to TRUE, the Upload Applet is enabled so that multiple files can be zipped and checked in as a single content item.

When set to FALSE, the Upload Applet is disabled.

Default is FALSE.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"
- Section 4.2.2, "Clients"

Location

- System Properties, **Options** tab, **Enable upload applet**
- Admin Server, General Configuration, **Enable upload applet**
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting, enables the Upload Applet:

```
MultiUpload=true
```

As script, evaluates the condition of the Upload Applet:

```
<$MultiUpload$>
```

See Also

- DownloadApplet
- UploadApplet

NoMatches

Checks whether matches were found from a search query.

Generally used to display a message on the search results page.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.20, "Searching"

Output

- Returns TRUE if no matches were found.
- Returns FALSE if any matches were found.

Example

Displays text if no matches were found from a query:

```
<$if NoMatches$>
    <p><font face="arial" size="2">
    Found no matches out of <$TotalDocsProcessed$> documents searched matching the
query.</p>
<$endif$>
```

noMCPrefill

Specifies that the MultiCheckin component should not prefill metadata fields.

This variable can be used on special pages such as custom HCST and contribution folders component pages.

When set to TRUE, the MultiCheckin component will not prefill metadata fields.

When set to FALSE, the MultiCheckin component will prefill metadata fields.

Type and Usage

- Section 4.1.5.1, "Page Display Variables"
- Section 4.2.16, "Page Display"

Output

- Returns `TRUE` if the `MultiCheckin` component will not prefill metadata fields.
- Returns `FALSE` if the `MultiCheckin` component will prefill metadata fields.

Example

Set near the top of the page:

```
noMCPrefill=1
```

Used as script:

```
<$noMCPrefill$>
```

NotificationQuery

This setting defines criteria for the automatic query that searches for expired content.

If `NotificationQuery` is not set, the default value is all content that expires in the next seven days.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"

Parameters

There is one parameter, the query to be used. The query can be one of the following. See the "Example" section for sample queries.

- An Idoc Script query, built from Idoc Script.
- A URL encoded query. This uses the URL displayed in the web browser address when a search is performed.
- A plain text query which defines the search variables.

Location

IntradocDir`/config/config.cfg`

Example

Idoc Script Example

When used in conjunction with database indexing, the following query provides email notification for all documents that expire in seven days:

```
NotificationQuery=dOutDate < '<$dateCurrent(7)$>'>
```

URL Encoded Example

The following query returns all content expiring after August 1, 2007. The URL from the browser address line is copied, beginning with the `QueryText` portion:

```
NotificationQuery=QueryText=dOutDate+%3C+%608%2F1%2F06%60&SearchProviders= [...}
```

Plain Text Query

The following query returns all content expiring after August 1, 2007:

```
NotificationQuery=8/1/07
```

See Also

- "EnableExpirationNotifier" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- "NotificationIntervalInDays" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- "NotificationMaximum" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

- "NotifyExtras" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- "NotifyTime" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

OneMatch

Checks if only one match was found from a search query.

Generally used to display a message on the search results page.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.20, "Searching"

Output

- Returns `TRUE` if only one match was found.
- Returns `FALSE` if no matches or more than one match were found.

Example

Displays text if only one match was found from a query:

```
<$if OneMatch$>
    <p><font face="arial" size="2">
    Found <$TotalRows$> document matching the query.</p>
<$endif$>
```

optionListKey

Specifies the name of a `ResultSet` column that contains option list values.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the values of the option list `ResultSet` column.

Example

Specifies `dFormat` as the column in the `DocFormats` ResultSet to get option list values from for the standard `Format` field on a checkin page:

```
<@dynamichtml std_override_format_field@>
<$if not isInfo and IsOverrideFormat$>
<$fieldIsOptionList = 1, optionListResultSet = "DocFormats",
    optionListKey = "dFormat",
    optionListValueInclude = "std_override_format_option_value",
    addEmptyOption = 1, emptyOptionLabel = lc("wwEmptyFormatOption")$>
<$include std_display_field$>
<$endif$>
<@end@>7
```

See Also

- optionListResultSet

optionListName

Specifies the name of an option list.

For standard metadata fields, this is the name of the internal option list. For more information, see Section 3.6.3.1, "Internal Option Lists."

For custom metadata fields, this is the name of the field with a suffix of `.options`.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the option list name.

Example

Sets the option list name to `docAuthors` if the field is not restricted to a single user:

```
<$if SingleUser$>
    <$isInfoOnly = 1$>
  <$else$>
    <$fieldIsOptionList = 1, optionListName = "docAuthors"$>
    <$if HasExternalUsers$><$fieldOptionListType= "combo"$>
    <$endif$>
<$endif$>
```

Defines the default option list script:

```
<$if optionsAllowPreselect and fieldValue$>
    <$defaultOptionListScript = "<$optList " & optionListName & ":fieldValue$>"$>
<$else$>
    <$defaultOptionListScript = "<$optList " & optionListName & ":noselected$>"$>
<$endif$>
```

See Also

- defaultOptionListScript
- fieldValue
- optionListScript
- optList()
- fieldIsOptionList
- fieldOptionListType

optionListResultSet

Specifies the name of a ResultSet that contains option list values.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the option list ResultSet.

Example

Specifies *DocFormats* as the option list ResultSet for the standard *Format* field on a search page:

```
<@dynamichtml std_format_fields@>
<$if ClientControlled or dFormat or dExtension$>
<$fieldName = "dFormat", fieldCaption = lc("wwNativeFormat"),
    optionListResultSet = "DocFormats"$> <$include std_display_field$>
<$fieldName = "dExtension", fieldCaption = lc("wwNativeExtension"),
    fieldWidth = 10$>
<$include std_display_field$>
<$endif$>
<@end@>
```

Creates an option list by looping over a ResultSet:

```
<@dynamichtml compute_resultset_option_list_script@>
    <$if not optionListKey$>
<$optionListKey = fieldName$>
    <$endif$>
    <$defaultOptionListScript = "<$loop " & optionListResultSet & "$>" &
    "<$inc('std_resultset_option_list_item')$>" & "<$endloop$>"$>
<@end@>
```

See Also

- optionListKey
- defaultOptionListScript
- optionListValueInclude

optionListScript

When this variable is set to a non-empty value, the eval function is used when displaying the option list for the field. This variable allows the standard implementation of option lists (defined by the `defaultOptionListScript` variable) to be overridden.

Type and Usage

- Section 4.1.5.2.2, "Common Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns `TRUE` if the value is nonempty.
- Returns `FALSE` if the value is an empty string.

Example

Defines a custom script for creation of an option list:

```
<$customOptionListScript = getValue("#active", fieldName & ":optionListScript")$>
```

See Also

- defaultOptionListScript

optionListValueInclude

Specifies an include that defines the values for an option list.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

Returns the include code.

Example

Defines `std_override_format_option_value` as the option list value include:

```
<@dynamichtml std_override_format_field@>
<$if not isInfo and IsOverrideFormat$>
<$fieldIsOptionList = 1, optionListResultSet = "DocFormats",
    optionListKey = "dFormat",
    optionListValueInclude = "std_override_format_option_value",
    addEmptyOption = 1, emptyOptionLabel = lc("wwEmptyFormatOption")$>
<$include std_display_field$>
<$endif$>
<@end@>
<@dynamichtml std_override_format_option_value@>
<$dDescription$>
<@end@>
```

Specifies the include to use to display options in an option list from a ResultSet:

```
<@dynamichtml std_resultset_option_list_item@>
<$curValue = getValue("#active", optionListKey)$>
<option value="<$curValue$>" <$if optionsAllowPreselect and strEquals(curValue,
fieldValue)$>selected<$endif$>>
<$if optionListValueInclude$>
<$inc(optionListValueInclude)$>
<$else$>
<$curValue$>
<$endif$>
<@end@>
```

See Also

- optionListResultSet

optionsAllowPreselect

Specifies that the metadata field option list can be prefilled with its last value.

Type and Usage

- Section 4.1.5.2.3, "Other Field Display Variables"
- Section 4.2.12, "Field Display"

Output

- Returns TRUE if the option list can be prefilled.
- Returns FALSE if the option list cannot be prefilled.

Example

Specifies that the option list can be prefilled:

```
<$optionsAllowPreselect=1$>
```

Determines if the option list will be prefilled:

```
<$if optionsAllowPreselect and fieldValue$>
<$defaultOptionListScript = "<$optList " & optionListName & ":fieldValue$>"$>
<$else$>
<$defaultOptionListScript = "<$optList " & optionListName & ":noselected$>"$>
<$endif$>
```

optList()

Generates an option list.

This function is used extensively to create option lists on Content Server pages.

This function only produces output when used with a service that calls `loadMetaOptionsList`

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.12, "Field Display"

Parameters

Takes one parameter and one optional argument to the parameter:

The only parameter is a field name, option list key, or variable.

- **Field name** syntax is `<$optList fieldName$>`. For custom metadata fields, the field name will resolve to `<$optList xFieldName.options$>`, or you can specify the `.options` suffix directly.
- **Option list key** syntax is `<$optList ListName$>`.
  - For standard metadata fields, the *ListName* is the name of the internal option list (such as *docAuthors*). For more information, see Section 3.6.3.1, "Internal Option Lists."
  - For custom metadata fields, the *ListName* is the name of the option list key, such as *FieldNameList*.
- **Variable** syntax is `<$optList variable$>`. The variable must resolve to a field name or an option list key.
- One of the following optional arguments can be added to the parameter:
  - **:noselected**-No values are selected when the option list is displayed.
  - **:fieldValue**-The value specified by the *fieldValue* variable is selected as the default value in the option list.

Output

Returns a list of values.

Example

This script generates a list of possible authors from the internal *docAuthors* list:

```
<$optList docAuthors$>
```

This script generates a list of the options specified in the *xRegion* custom metadata field:

```
<$optList xRegion.options$>
```

This script generates an option list from the variable *optionListName* and specifies the default value:

```
<$if optionsAllowPreselect and fieldValue$>
```

```
    <$defaultOptionListScript = "<$optList " & optionListName & ":fieldValue$>"$>
<$else$>
    <$defaultOptionListScript = "<$optList " & optionListName & ":noselected$>"$>
<$endif$>
```

See Also

- Section 3.6.3, "Option Lists"
- defaultOptionListScript
- optionListName
- optionListScript
- optionsAllowPreselect
- rsMakeFromList()

PageParent

Checks whether a directory page in the Library has a parent page.

Type and Usage

- Section 4.1.9, "Value Variables"
- Section 4.2.16, "Page Display"

Output

- Returns TRUE if the directory page is a child (subfolder) of another directory page.
- Returns FALSE if the directory page is not a child (subfolder).

Example

Checks if the directory page is a subfolder:

```
<$PageParent$>
```

parseDataEntryDate()

Parses a date but uses the failover logic for using the alternate parsing formats.

Dates convert to milliseconds when used with standard comparison operators. For example, the expression `(60*60*1000)` equals one hour.

A common usage of this function is to adjust the current time using a multiplication expression that represents a number of seconds, minutes, hours, or days.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is an expression to be parsed.

Output

Returns a Java date object, which is converted to a string for display.

Example

```
<$parseDataEntryDate$>
```

parseDate

Parses a date/time to allow evaluation and arithmetic.

Dates convert to milliseconds when used with standard comparison operators. For example, the expression `(60*60*1000)` equals one hour.

A common usage of this function is to adjust the current time using a multiplication expression that represents a number of seconds, minutes, hours, or days.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

The only parameter is an expression to be parsed.

Output

Returns a Java date object, which is converted to a string for display.

Example

Each of the following expressions returns the date and time one day in the past:

```
<$parseDate(dateCurrent(-1))$>
<$parseDate(dateCurrent()-(24*60*60*1000))$>
<$dateCurrent(-1)$>
<$dateCurrent()-(24*60*60*1000)$>
```

Returns the time one hour in the future. The first line adds one hour using a time multiplication expression, assigns that time and date to a custom variable, and suppresses the output. The second line references the custom variable and defines that only the time is displayed:

```
<$exec my_customParseTime parseDate(dateCurrent()+(1000*60*60))$>
<$formatTimeOnly(my_customParseTime)$>
```

Returns the date one year in the future. The first line adds one year using a time multiplication expression, assigns that time and date to a custom variable, and suppresses the output. The second line references the custom variable and defines that only the date in long format is displayed:

```
<$exec my_customParseTime parseDate(dateCurrent()+(1000*60*60*24*365))$>
<$formatTimeOnly(my_customParseTime)$>
```

This script evaluates whether the date seven days in the future is greater than the expiration date and returns a message to the user if true:

```
<$if dOutDate$>
    <$if dateCurrent(7) > parseDate(dOutDate)$>
Content item expires in one week.
     <$endif$>
<$endif$>
```

This script uses `parseDate` within a conditional statement for customized workflow jumps. The script specifies that if the last time we entered this step was four days ago, go to the first step in workflow `wf_late` and set the return step to be the next step:

```
<$if parseDate(wfCurrentGet("lastEntryTs")) < dateCurrent(-4)$>
    <$wfSet("wfJumpName", "lateJump")$>
    <$wfSet("wfJumpTargetStep", "step_1@wf_late")$>
    <$wfSet("wfJumpReturnStep", wfCurrentStep(1))$>
    <$wfSet("wfJumpEntryNotifyOff", "0")$>
<$endif$>
```

See Also

- dateCurrent()

parseDateWithPattern()

Parses a date/time to a specified date/time pattern.

Dates convert to milliseconds when used with standard comparison operators. For example, the expression `(60*60*1000)` equals one hour.

A common usage of this function is to adjust the current time using a multiplication expression that represents a number of seconds, minutes, hours, or days.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.8, "Date and Time"

Parameters

Takes two parameters:

- The first parameter is a date string used by the Content Server instance, or a date object created with the parseDate or dateCurrent function.
- The second parameter is the date/time pattern, which is a standard Java `SimpleDateFormat` pattern string, such as `MM/dd/yyyy`.

Output

Returns a Java date object, which is converted to a string for display.

Example

Displays the current date and time in the format specified by the pattern (for example, `Wed, 4 Jul 2001 12:08:56 -0700`):

```
<$parseDateWithPattern(<$dateCurrent()$>,"EEE, d MMM yyyy HH:mm:ss Z")$>
```

See Also

- parseDate
- dateCurrent()
- formatDateWithPattern()

PATH_INFO

This setting is obsolete. The web server filter no longer sends this information.

Retrieves additional information about the file system path to the CGI computer.

When the virtual path is returned by the PATH_TRANSLATED variable, any additional information at the end of this path is also returned as PATH_INFO.

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the additional virtual path information as a string.

Example

Retrieves additional CGI path information:

```
<$PATH_INFO$>
```

See Also

- PATH_TRANSLATED

PATH_TRANSLATED

Retrieves the file system path to the CGI computer, for example:

`c:/domain/weblayout/idcplg/idc_cgi_isapi-idcm1.dll.`

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the virtual path as a string.

Example

Retrieves the CGI path:

```
<$PATH_TRANSLATED$>
```

pneNavigation()

Enables the left sidebar navigation.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.16, "Page Display"

Parameters

Set as a name/value pair:

```
pneNavigation=1
```

Default value is 1 (enabled).

To disable this function, it must be set to a null string.

Output

Evaluates to `TRUE` or `FALSE`.

Example

Enables the sidebar navigation:

```
<$pneNavigation=1$>
```

To force the sidebar navigation off, set it to a null string:

```
<$pneNavigation=""$>
```

Setting the definition to other than 1 or a null string value is invalid and will not disable the sidebar navigation:

```
<$pneNavigation=0$>
```

QUERY_STRING

Retrieves the string that follows the ? delimiter in the URL for a query.

This variable is specific to the current CGI request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.20, "Searching"
- Section 4.2.25, "Web Servers"

Output

Returns the query information as a string.

Example

As information output on a page or to a log:

```
QUERY_STRING=IdcService=GET_DOC_PAGE&Action=GetTemplatePage&Page=STD_QUERY_PAGE
```

As part of an Idoc Script statement or evaluation:

```
<$if QUERY_STRING$>
<!--statement-->
```

regexMatches()

Searches a string for a specific pattern using a regular expression to do matching. Regular expression constructs can contain characters, character classes, and other classes and quantifiers. For details about the Java API for Class Pattern, see `http://www.oracle.com/technetwork/java/index.html`.

This feature is only available with JVM 1.4 or later versions; this is the default version for WebCenter Content version 7.0 and later.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.5, "Content Server"

Parameters

Takes two parameters:

- The first parameter is the string to match.
- The second parameter is the expression.

Example

The following example returns `FALSE`, because the string does not match the expression:

```
<$regexMatches("abcdef","abc")$>
```

The following example returns `TRUE` because the wild cards are present. If standard wild cards such as the asterisk (*) were used instead of the dot-asterisk (.*) convention, the match would fail.

```
<$regexMatches("abcdef", ".*abc.*")$>
```

See Also

- Section 3.5, "Operators"
- regexReplaceAll()
- regexReplaceFirst()
- strEquals()
- strIndexOf()

regexReplaceAll()

Searches a string for a specific pattern using a regular expression to do matching and replacing. Regular expression constructs can contain characters, character classes, and other classes and quantifiers. For details about the Java API for Class Pattern, see `s://www.oracle.com/technetwork/java/index.html`.

This feature is only available with JVM 1.4 or later versions; this is the default version for WebCenter Content version 7.0 and later.

It replaces all instances of the regular expression with the replacement string.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.3, "Content Items"

Parameters

Takes three parameters:

- The first parameter is the original string.
- The second parameter is the string to match.
- The third parameter is the replacement string.

Example

The following example returns `xyzdef xyzdef`:

```
<$regexReplaceAll("abcdef abcdef","abc","xyz")$>
```

See Also

- Section 3.5, "Operators"
- regexMatches()
- regexReplaceFirst()
- strEquals()
- strIndexOf()
- strSubstring()

regexReplaceFirst()

Searches a string for a specific pattern using a regular expression to do matching and replaces the first instance with a replacement string. Regular expression constructs can contain characters, character classes, and other classes and quantifiers. For details about the Java API for Class Pattern, see `http://www.oracle.com/technetwork/java/index.html`.

This feature is only available with JVM 1.4 or later versions; this is the default version for WebCenter Content version 7.0 and later.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.3, "Content Items"

Parameters

Takes three parameters:

- The first parameter is the original string.
- The second parameter is the string to match.
- The third parameter is the replacement string.

Example

The following example returns `xyzdef abcdef`:

```
<$regexReplaceFirst("abcdef abcdef","abc","xyz")$>
```

See Also

- Section 3.5, "Operators"
- regexMatches()
- regexReplaceAll()
- strEquals()
- strIndexOf()
- strSubstring()

REMOTE_ADDR

Returns the IP address of the remote host making the request.

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the IP address as a string.

Example

As information output on a page or to a log:

```
REMOTE_ADDR=207.0.0.1
```

As part of an Idoc Script statement or evaluation:

```
<$if REMOTE_ADDR$>
<!--statement-->
```

See Also

- REMOTE_HOST

REMOTE_HOST

Returns the name of the remote host making the request.

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

- Returns the host name as a string.
- If the hostname is unknown to the server, returns the value of REMOTE_ADDR as a string.

Example

As information output on a page or to a log:

```
REMOTE_HOST=207.0.0.1
```

As part of an Idoc Script statement or evaluation:

```
<$if REMOTE_HOST$>
<!--statement-->
```

See Also

- REMOTE_ADDR

REQUEST_METHOD

Retrieves the method that was used to make the request.

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the request method as a string.

Example

As information output on a page or to a log:

```
REQUEST_METHOD=GET
```

As part of an Idoc Script statement or evaluation:

```
<$if REQUEST_METHOD$>
<!--statement-->
```

requiredMsg

Specifies the error message to be displayed if a required metadata field does not have a value upon checkin.

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the error message as a string.

Example

Defines the required field error message for the *dDocTitle* metadata field as the `wwSpecifyTitle` localized string:

```
<$fieldName = "dDocTitle", fieldCaption = lc("wwTitle"), isRequired = 1, fieldType
= "BigText", requiredMsg = lc("wwSpecifyTitle")$>
```

See Also

- isRequired

ResultsTitle

Defines a title for the search results page.

This variable is used by the Web Layout Editor to name the search results page and display a heading at the top of that page.

Usage

- Section 4.2.16, "Page Display"
- Section 4.2.20, "Searching"

Output

Returns the page title as a string.

Example

As an HDA entry, names the search results page:

```
@Properties LocalData
ResultsTitle=Content Items
@end
```

As a script, returns the defined name:

```
<$if ResultsTitle$>ResultsTitle=<$url(ResultsTitle)$>
```

rptDisplayMapValue()

This function is meant for internal use only.

This function returns a localized string representation of an internal key code. These key codes are used to store status and state flags for content items. This function is used on the Content Information page and workflow pages to display descriptions of the internal state.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.15, "Localization"

Parameters

Takes two parameters:

- The name of the table where the keys are stored.
- The key to be localized.

rs()

This function returns a ResultSet given the name of the ResultSet. Because the actual object it returns is a ResultSet, certain actions can be performed on the return value that cannot be performed on the return values of other functions. In particular, the loop syntax can be applied to the result.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

This function has one parameter, `resultSetName`, which is the name of the ResultSet to return.

Output

A ResultSet object if found, otherwise null.

Example

Suppose `MyResultSet` is the name of a ResultSet, then you could loop on the ResultSet using the following syntax:

```
<$loop rs("MyResultSet")$>
... Script inside loop ...
<$endloop$>
```

This function also can be used when temporarily assigning variables when calling Idoc Script functions. In particular, the following example will temporarily assign the variable `rsParam` to point to the same ResultSet as pointed to by `MyResultSet` for the duration of the call to include the resource include *`my_include`*. If the result of the `rs` function is assigned to a variable, then that variable will have a shared pointer to the ResultSet creating the same effect as if `rsCreateReference` were called.

```
<$inc("my_include", rsParam=rs("MyResultSet"))$>
```

rsAddFields()

Adds new fields to a ResultSet. This function will only add the field if it is not already present. Note that setValue(...) can also add new fields but it only works on ResultSets that are nonempty and are on a currently valid row.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of the ResultSet to get new fields (columns) added.
- The second parameter is a comma separated list of fields to add.(the column names to be added).

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsAddFields(rsName, fieldsList)$>
```

rsAddFieldsWithDefaults()

This function adds new fields to an existing ResultSet. Unlike `rsAddFields`, this function provides the ability to specify default values for any fields that are added. Note that `rsAddFieldsWithDefaults` never over-writes any data in fields that already exist; it only adds new fields to a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The following table lists parameters for the function.

Parameters	Description
`resultSetName`	The name of the ResultSet.
`newFields`	A comma-separated list of fields to append to the ResultSet. Any specified fields that already exist in the ResultSet are ignored.
`defaultValues`	An optional comma-separated list of default values that are set for the new fields in each row. Each value in this list corresponds to the new field in the same spot of the list newFields. If you need to set a default value with a comma in it, you can use '^' as a comma. If the defaultValues list is longer or shorter than newFields list, then the function ignores the extra default values or uses the empty string for unspecified default values, respectively.

Output

Nothing.

Example

Adding fields

```
<$rsCreateResultSet("MyList", "field1,field2,field3")$>
<$rsAppendRowValues("MyList", "A,B,C,D,E,F")$>
<$rsAddFieldsWithDefaults("MyList", "field4,field5,field6")$>
```

In this basic example, we create a ResultSet named `MyList`, fill it with some values, then add three more fields to `MyList`, without setting default values. The resulting table will look like the following:

field1	field2	field3	field4	field5	field6
A	B	C			
D	E	F			

Adding fields with default

```
<$rsCreateResultSet("MyList", "field1,field2,field3")$>
<$rsAppendRowValues("MyList", "A,B,C,D,E,F")$>
<$rsAddFieldsWithDefaults("MyList", "field4,field5,field6", "X,Y,Z")$>
```

Here we define default values for the new fields. The resulting table will look like the following:

field1	field2	field3	field4	field5	field6
A	B	C	X	Y	Z
D	E	F	X	Y	Z

When a field already exists

```
<$rsCreateResultSet("MyList", "field1,field2,field3")$>
<$rsAppendRowValues("MyList", "A,B,C,D,E,F")$>
<$rsAddFieldsWithDefaults("MyList", "field1,field4,field5", "X,Y,Z")$>
```

In this case, we try and add a field that already exists in `MyList`. This action is completely ignored, and old field values are preserved. The resulting table will look like the following:

field1	field2	field3	field4	field5
A	B	C	X	Y
D	E	F	X	Y

Fewer default values specified

```
<$rsCreateResultSet("MyList", "field1,field2,field3")$>
<$rsAppendRowValues("MyList", "A,B,C,D,E,F")$>
<$rsAddFieldsWithDefaults("MyList", "field4,field5,field6", "X,Y")$>
```

Notice in this example how there are not enough default values in the default value list for all fields. In this case, it just fills in blanks for the unspecified column.

field1	field2	field3	field4	field5	field6
A	B	C	X	Y	
D	E	F	X	Y	

See Also

- rsAddFields()

rsAddRowCountColumn()

This function adds a new field whose value for each row is the row count for that row. The count starts at 0. This function is useful to use just before using `rsSort(...)` on XML data island ResultSets (inside `.hcsp` files) so that the original row location can be preserved.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of ResultSet to be modified so that it has an new row count column.
- The second parameter is the name of the field that will hold the row count.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsAddRowCountColumn(rsName, countFieldName)$>
```

rsAppend()

This function is similar in nature to `rsMerge`, except there are no merge test criteria. All rows from the ResultSet `rsNameSource` are appended to the ResultSet `rsNameTarget`. In addition, all fields in `rsNameSource` not in `rsNameTarget` will be added to `rsNameTarget`. Useful for doing a simple combining of ResultSets.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the ResultSet that will be modified by the merge.
- The second parameter is the ResultSet containing the rows that will be appended to the target.

Output

- Returns `TRUE` if the function is successful.
- Returns `FALSE` if the function fails.

Example

```
<$rsAppend("targetRsetName", "sourceRsetName")$>
```

See Also

- rsMerge()

rsAppendNewRow()

Appends a new row to the end of the ResultSet. The current row is not affected.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The only parameter is the name of the ResultSet to receive the new row.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

```
<$rsAppendNewRow("SearchResults")$>
```

rsAppendRowValues()

A utility function for adding new rows to a ResultSet. The values list is a comma-delimited list of values (using the escape rule of ',' -> '^', '^' -> '#^', '#' -> '##' to encode each value) split up into rows. Assume *n*fields is the number of fields in the ResultSet. The values list is split up into rows by counting *n*fields values to create a new row and then taking the next *n*fields values to create each following row and so on. If the number of values is not an exact multiple of *n*fields then the last row has its fields values padded out with empty strings. This function is useful for compactly hard coding ResultSets using Idoc Script.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameters the name of the ResultSet that is to get new rows.
- The second parameter is the new values to add.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rowValues = "a1, b1, a2, b2"$>
<$rsCreateResultSet("ResultSet1", "ColumnA,ColumnB")$>
<$rsAppendRowValues("ResultSet1", rowValues$>
<table border=2>
    <tr><td>ColumnA</td><td>ColumnB</td></tr>
    <$loop ResultSet1$>
    <tr><td><$ColumnA$></td><td><$ColumnB$></td>
</tr>
<$endloop$>
</table>
```

The resulting HTML would look like the following.

Column A	Column B
Column A	Column B
A1	B1
A2	B2

rsCopyFiltered()

This function copies only selected rows from one ResultSet to create another ResultSet. Any prior ResultSet with name `rsNameTarget` will be replaced. The rows are selected by testing each row's value of `filterField` using a test against the pattern in `filterPattern`. The pattern match is the same as used in the Idoc Script `like` operator.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes four parameters:

- The first parameter is the ResultSet that is providing the rows to copy.
- The second parameter is the new ResultSet created by the copy.
- The third parameter is the name of the field being tested.
- The fourth parameter is the pattern match to apply to see if the row should be copied.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsCopyFiltered(rsNameSource, rsNameTarget, filterField, filterPattern)$>
```

rsCreateReference()

This function sets an Idoc Script variable with a shared reference to a pre-existing ResultSet in the request data.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

This function has the following parameters.

Parameters	Description
`sourceResultSet`	The name of a pre-existing ResultSet.
`targetResultSet`	The name of the variable that will share a reference to precisely the same ResultSet.

Output

If successful, returns TRUE.

Example

The current value of `<$MyResultSet.field2$>` and `<$MyResultSetAlias.field2$>` will both be the letter `E`.

```
<$rsCreateResultSet("MyResultSet", "field1,field2,field3")$>
<$rsAppendRowValues("MyResultSet", "A,B,C,D,E,F")$>
<$rsCreateReference("MyResultSet", "MyResultSetAlias")$>
<$rsNext("MyResultSetAlias")$>
```

rsCreateResultSet()

Creates a ResultSet initialized with the list of fields in `fieldList`. All the fields will be set as `string` type.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of the ResultSet to create.
- The second parameter is a comma separated list of fields.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsCreateResultSet(rsName, fieldList)$>
```

rsDeleteRow()

Deletes the current row in the ResultSet. The current row is advanced to the next row (or points to after the end of the ResultSet if the end row in the ResultSet is deleted).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The only parameter is the name of the ResultSet whose current row is deleted.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

```
<$rsDeleteRow("Folders")$>
```

rsDocInfoRowAllowDisclosure()

Checks if the current user is allowed to view the URL of the content item referenced by the current row of a ResultSet.

This function is useful for selectively showing the URLs of a ResultSet generated by a content item query.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"
- Section 4.2.21.1, "Internal Security"

Parameters

The only parameter is the name of a ResultSet whose current row contains information about a content item.

Output

- Returns TRUE if the user is allowed to view the URL of the content item.
- Returns FALSE if the user is not allowed view the URL or the ResultSet is empty.

Example

Checks if the user can view the URL of the content item in the current row of `resultSet1`.

```
<$rsDocInfoRowAllowDisclosure("resultSet1")$>
```

rsExists()

Checks to see if a ResultSet exists.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes one parameter, the name of the ResultSet.

Output

- Returns TRUE if the ResultSet is found.
- Returns FALSE if the ResultSet does not exist or if it is empty.

Example

```
<$if rsExists("resultSet1")$>
    code to execute if TRUE
<$endif$>
```

rsFieldByIndex()

Retrieves the name of the field at a specified column index, starting from zero.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The name of the ResultSet.
- The index number.

Example

```
<$rsFieldName = rsFieldByIndex("resultSet1", "2")$>
```

rsFieldExists()

Checks to see if the named ResultSet contains the specific field.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The name of the ResultSet to be searched.
- The name of the field to be found.

Output

- Returns TRUE if the field is found.
- Returns FALSE if the field does not exist.

Example

```
<$if rsFieldExists("resultSet1", "FieldName1")$>
    code to execute if TRUE
<$endif$>
```

rsFindRowPrimary()

Searches the first column of a ResultSet for a matching value.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of a ResultSet whose first column is to be searched.
- The second parameter is the value to be searched for.

Output

- Returns TRUE if the specified value is found in the first column of the specified ResultSet.
- Returns FALSE if the specified value is not found.

Example

Searches the first column of *resultSet1* until a value matching *value1* is found.

```
<$rsFindRowPrimary("resultSet1","value1")$>
```

rsFirst()

Moves to the first row in a ResultSet.

Type and Usage

- SSection 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The only parameter is the name of a ResultSet.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

Advances the `SearchResults` ResultSet to the first row:

```
<$exec rsFirst("SearchResults")$>
```

rsInsertNewRow()

Inserts a row just before the current row of the ResultSet being manipulated. The new row then becomes the current row.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The only parameter is the name of the ResultSet in which to have a new row inserted.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsInsertNewRow("SearchResults")$>
```

rsIsRowPresent()

Checks the status of the current row in a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes one parameter, the name of the ResultSet.

Output

- Returns TRUE if the ResultSet is currently on a valid row.
- Returns FALSE if not on a valid row.

Example

```
<$if rsIsRowPresent("resultSet1")$>
    code to execute if TRUE
<$endif$>
```

rsLoopInclude()

This function loops on a ResultSet, executing a resource include once for each row. During the execution of this include, the ResultSet is temporarily made active (as is done with the standard <$loop ...$> construction).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

This function has the following parameters:

Parameters	Description
ResultSetName	The name of the ResultSet to loop over.
includeName	The name of the resource include to execute.

Output

The combined output of the various iterations of the resource include.

Example

```
<$rsLoopInclude(resultSetName, includeName)$>
```

rsLoopSingleRowInclude()

This function temporarily promotes a ResultSet to be active while executing a single resource include. It functions much like rsLoopInclude, however the include is only executed once for the current row of the ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

This function has the following parameters.

Parameters	Description
`ResultSetName`	The name of the ResultSet to make active.
`includeName`	The name of the resource include to execute.

Output

The output of the resource include.

Example

```
<$rsLoopSingleRowInclude(resultSetName, includeName)$>
```

See Also

- rsLoopInclude()

rsMakeFromList()

Creates a single-column ResultSet from an option list.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two required parameters and one optional parameter:

- The first parameter is the name of the ResultSet to be created.
- The second parameter is the name of an existing option list (for example, `docAccounts` or `xMyCustomField.options`).
- The third optional parameter is the name of the ResultSet column. If no value is specified, the column name defaults to `row`.

Output

Creates a ResultSet with one column, populated by the values in the specified option list.

Example

Creates two ResultSets called `ListA` and `ListB` from the `securityGroups` and `docAuthors` option lists:

```
<$lista = "securityGroups"$>
<$rsMakeFromList("ListA", lista)$>
<$rsMakeFromList("ListB", "docAuthors", "name")$>
<table border=2>
    <tr><td>Security Groups</td><td>Authors</td></tr>
    <tr>
        <td><$loop ListA$><$row$><br>
        <$endloop$></td>
        <td><$loop ListB$><$name$><br>
        <$endloop$></td>
    </tr>
</table>
```

The ResultSets would look like the following.

```
@ResultSet ListA
1
row
Public
Secure
@end
@ResultSet ListB
1
name
hchang
pkelly
sysadmin
user1
```

```
@end
```

The resulting HTML would look like the following.

ListA	ListB
Security Groups	Authors
	hchang
Public	okelly
Secure	sysadmin
	user1

See Also

- rsMakeFromString()
- Option Lists
- optList()

rsMakeFromString()

Creates a single-column ResultSet from a string.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two required parameters and one optional parameter:

- The first parameter is the name of the ResultSet to be created.
- The second parameter is a comma-delimited list of strings to be parsed (such as `a,b,c,d`), or a variable that has a comma-delimited string as its value.
- The third optional parameter is the name of the ResultSet column. If no value is specified, the column name defaults to `row`.

Output

Creates a ResultSet with one column, populated by the specified string values.

Example

Creates two ResultSets, called `StringA` and `StringB`:

```
<$stringa = "a,b,c,d"$>
<$rsMakeFromString("StringA", stringa)$>
<$rsMakeFromString("StringB", "A,B,C,D", "name")$>
<table border=2>
    <tr><td>StringA</td><td>StringB</td></tr>
    <tr>
        <td><$loop StringA$><$row$><br>
        <$endloop$></td>
        <td><$loop StringB$><$name$><br>
        <$endloop$></td>
    </tr>
</table>
```

The ResultSets would look like the following.

```
@ResultSet StringA
1
row
a
b
c
d
@end
@ResultSet StringB
1
name
A
B
C
D
```

```
@end
```

The resulting HTML would look like the following.

Stringa	Stringb
A	A
B	B
C	C
D	D

See Also

- rsMakeFromList()

rsMerge()

Merges the rows of one ResultSet into another. If the value from the `rsNameSource` ResultSet in the `rsCommonField` field matches the value for the same field in a row for `rsNameTarget`, then that row will be replaced. Otherwise, the row from rsNameSource will be appended. If there are multiple rows in `rsNameTarget` that are matched by the same value, then only the first row is replaced and it is replaced by the last row in `rsNameSource` that has a matching row. Any fields in the ResultSet `rsNameSource` that are not in `rsNameTarget` are added as new fields to `rsNameTarget`. This function is best performed on ResultSets that have only unique values in the `rsCommonField` field so that issues with multiple matches are avoided.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes three parameters:

- The first parameter (rsNameTarget) is the ResultSet that will be modified by the merge.
- The second parameter (rsNameSource) is the ResultSet containing the rows that will be merged into the target.
- The third parameter (rsCommonField) is the field that occurs in both results sets that will be used as the basis of the merge.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsMerge(rsNameTarget, rsNameSource, commonField$>
```

See Also

- rsAppend()

rsMergeDelete()

Analogous to `rsMergeReplaceOnly`, except matching rows in the ResultSet `rsNameTarget` are removed instead of replaced. It is useful for removing rows from one ResultSet that already occur in another.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes three parameters:

- The first parameter (rsNameTarget) is the ResultSet that will be modified by the merge.
- The second parameter (rsNameSource) is the ResultSet containing the rows that will be merged into the target.
- The third parameter (rsCommonField) is the field that occurs in both results sets that will be used as the test for which rows to remove from `rsNameTarget`.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsMergeDelete(rsNameTarget, rsNameSource, rsCommonField)$>
```

rsMergeReplaceOnly()

Similar to `rsMerge`, but rows are only replaced; none are appended to the ResultSet `rsNameTarget`. In addition, new fields from `rsNameSource` are not added to `rsNameTarget`.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes three parameters:

- The first parameter (rsNameTarget) is the ResultSet that will be modified by the merge.
- The second parameter (rsNameSource) is the ResultSet containing the rows that will be merged into the target.
- The third parameter (rsCommonField) is the field that occurs in both results sets that will be used as the basis of the merge

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$rsMergeReplaceOnly(rsNameTarget, rsNameSource, rsCommonField) $>
```

rsNext()

Moves to the next row in a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The only parameter is the name of a ResultSet.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

Advances the *SearchResults* ResultSet to the next row:

```
<$exec rsNext("SearchResults")$>
```

rsNumFields()

Provides a count of the number of fields in a ResultSet. This is useful when combined with `rsFieldByIndex`.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes one parameter, the name of the ResultSet.

Output

Returns the number of fields.

Example

```
<$numFields = rsNumFields("resultSet1")$>
```

See Also

- rsFieldByIndex()

rsNumRows()

Provides a count of the number of rows in a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes one parameter, the name of the ResultSet.

Output

Returns the number of rows.

Example

```
<$numRows = rsNumRows("resultSet1")$>
```

rsRemove()

Removes a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes one parameter, the name of the ResultSet.

Example

```
<$rsRemove("resultSet1")$>
```

rsRename()

Renames a ResultSet.

If you use a variable to define the new ResultSet name, you will need to loop over the new ResultSet variable manually using `rsFirst`, `rsNext`, and `#isRowPresent`. For example:

```
<$rsNewName="MySearchResults"$>
<$rsRename("SearchResults", rsNewName)$>
    <$rsFirst(rsNewName)$>
    <$loopwhile getValue(rsNewName, "#isRowPresent")$>
        <!--output code-->
        <$rsNext(rsNewName)$>
    <$endloop$>
```

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of an existing ResultSet.
- The second parameter is the new ResultSet name. The call will fail if the ResultSet already exists.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

Renames the `SearchResults` ResultSet to `MySearchResults`:

```
<$rsRename("SearchResults", "MySearchResults")$>
```

rsRenameField()

Renames a field in the ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes three parameters:

- The first parameter is the name of the ResultSet whose field is being renamed.
- The second parameter is the current name of the field.
- The third parameter is the new name of the field.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

Renames the `invoice` field to `outlays` in the `warehouse` ResultSet.

```
<$exec rsRenameField("warehouse", "invoice", "outlays")$>
```

rsSetRow()

Moves to a specified row in a ResultSet.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of the ResultSet.
- The second parameter is the number of the row.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

Advances the `SearchResults` ResultSet to the 10th row:

```
<$exec rsSetRow("SearchResults",10)$>
```

rsSort()

Sorts a ResultSet by a particular column.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes four parameters. For example:

```
rsSort(rsName,sortCol,sortType,sortOrder)
```

Parameter	Description
rsName	The name of the ResultSet.
sortCol	The name of the column to sort by.
sortType	The type of sort (defaults to *int*): **string**: sort alphabetically **int**: sort numerically **date**: sort by date
sortOrder	The sort order (defaults to *asc*): **asc**: ascending order **desc**: descending order

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

Sorts the `SearchResults` ResultSet by Content ID in descending order:

```
<$rsSort("SearchResults","dDocName","string","desc")$>
```

rsSortTree()

Sorts a ResultSet that is a representation of an XML data structure (with nodes, parent nodes, and depth attributes).

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes seven parameters. For example:

```
rsSortTree(rsName,itemIdCol,parentIdCol,nestLevelCol,sortCol,sortType,sortOrder)
```

Parameter	Description
rsName	The name of the ResultSet.
itemIdCol	The name of the column that contains the unique ID for each node.
parentIdCol	The name of the column that contains the ID for the parent node, if one exists.
nestLevelCol	The name of the column that contains the nest level (depth) for each node.
sortCol	The name of the column to sort by.
sortType	The type of sort (defaults to *int*): **string**: sort alphabetically **int:** sort numerically **date**: sort by date
sortOrder	The sort order (defaults to *asc*): **asc**: ascending order **desc**: descending order

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails or the ResultSet is empty.

Example

Sorts the `discussionPosts` ResultSet by the `itemNumber` column in ascending order:

```
<$rsSortTree("discussionPosts", "discussionPosts!itemNumber",
    "discussionPosts!parentNumber", "dpItemNestLevel",
    "discussionPosts!itemNumber", "int", "asc")$>
```

SafeDir

Retrieves the location where extra Content Server files are moved by the Content Server Analyzer utility.

Default is a subdirectory in the log directory specified by the `IdcAnalyzeLogDir` setting.

This variable is not settable.

Type and Usage

- Section 4.1.6.4, "Other Read-Only Variable"
- Section 4.2.9, "Debugging"
- Section 4.2.10, "Directories and Paths"

Example

As script, returns the value of the configuration setting:

```
<$SafeDir$>
```

See Also

- "IdcAnalyzeLogDir" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

SCRIPT_NAME

Retrieves the relative path to the CGI linking file.

The CGI linking file is the executable file for the web server security filter, which is a different file for each operating system and web server. For example, the file name is `nph-idc_cgi.exe` for Solaris.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the relative path as a string.

Example

As information output on a page or to a log:

```
SCRIPT_NAME=/idcplg/idc_cgi_isapi-instance>.dll
```

As part of an Idoc Script statement or evaluation:

```
<$if SCRIPT_NAME$>
<!--statement-->
```

SelfRegisteredAccounts

Defines the default accounts and permissions to be given to self-registered users.

This is a comma-delimited list.

Permissions for each account must be specified in parentheses after the account name.

The `#none` entry grants privileges to content items that have no account assigned. The `#all` entry grants privileges to all accounts.

There is no default value.

Returns the list of accounts as a string.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.21.1, "Internal Security"

Location

IntradocDir`/config/config.cfg`

Example

As a configuration setting:

```
SelfRegisteredAccounts=#none(RWDA),USERS/<$NewUser$>,BOS(R)
```

As script, returns the defined account information as a string:

```
<$SelfRegisteredAccounts$>
```

See Also

- SelfRegisteredAccounts

SelfRegisteredRoles

Defines the default roles to be given to self-registered users.

This is a comma-delimited list.

There is no default value.

Returns the roles as a string.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.21.1, "Internal Security"

Location

IntradocDir/config/config.cfg

Example

As a configuration setting:

```
SelfRegisteredRoles=guest,salesRole
```

As script, returns the defined roles as a string:

```
<$SelfRegisteredRoles$>
```

See Also

- SelfRegisteredRoles

SERVER_NAME

Retrieves the hostname, DNS alias, or IP address of the web server as it would appear in a self-referencing URL.

This variable is not request-specific; it is set for all requests.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the server information as a string.

Example

As information output on a page or to a log:

```
SERVER_NAME=centralserver
```

As part of an Idoc Script statement or evaluation:

```
<$if SERVER_NAME$>
<!--statement-->
```

SERVER_PORT

Retrieves the web server port number to which the request was sent.

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns the port number as a string.

Example

As information output on a page or to a log:

```
SERVER_PORT=80
```

As part of an Idoc Script statement or evaluation:

```
<$if SERVER_PORT$>
<!--statement-->
```

SERVER_PROTOCOL

Returns the protocol and revision of the incoming request.

This variable is specific to the current gateway program request.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a string in the format `protocol/revision`.

Example

As information output to a log:

```
SERVER_PROTOCOL=HTTP/1.0
```

As part of an Idoc Script statement or evaluation:

```
<$if SERVER_PROTOCOL$>
<!--statement-->
```

SERVER_SOFTWARE

This setting is obsolete for the 7.0 version and later.

The web server filter no longer sends this information.

Returns the name and version of the web server software that is answering the request.

This is also the server running the gateway.

This variable is not request-specific; it is set for all requests.

Type and Usage

- Section 4.1.3, "Environment Variables"
- Section 4.2.25, "Web Servers"

Output

Returns a string in the format `name/version`.

Example

As information output on a page or to a log:

```
SERVER_SOFTWARE=Microsoft-IIS/4.0
```

As part of an Idoc Script statement or evaluation:

```
<$if SERVER_SOFTWARE$>
<!--statement-->
```

setContentType()

Sets the content type for the returned page so the browser renders it properly. For example, a content type (file-format type or MIME type) of text/plain, application/excel, or text/xml.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

The only parameter is the content type.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$setContentType("text/plain")$>
```

setCookie

Sets a cookie value in a browser.

Used to retain session information or information about anonymous users.

An expiration date can also be passed to make the cookie expire at a specified time.

Important: Do not store secure data in a cookie. You should limit the use of cookies due to the risk of malicious users setting cookies on your site.

Type and Usage

- Section 4.1.7, "Settable Variables"
- Section 4.2.25, "Web Servers"

Parameters

This function can take three parameters:

- The first parameter is the name of the cookie.
- The second parameter is the value to be set.
- The third optional parameter is an expiration indicator specifying when the cookie will expire.

Example

The following example sets the cookie `myCounter` in the user's browser and increments the value when the user visits the page. The cookie will expire in one day.

```
<$myCounter=getCookie("myCounter")$>
<$if not myCounter$>
    <$myCounter=1$>
<$endif$>
Num times loaded: <$myCounter$>
<$setCookie("myCounter", myCounter+1, dateCurrent(1))$>
```

See Also

- getCookie

setExpires()

This function can be used to set an absolute time for an expiration of a page. This is best used for pages with a long life that may be requested frequently. Due to problems in time synchronization between the client and the server, it is not totally reliable, especially for short-lived caches. The data must follow one of the acceptable HTTP date formats.

Refer to the W3 Protocols for more information:

http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is an absolute date, formatted in the manner HTTP requires, after which this page should be refreshed

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

This will cause the page to expire at the given absolute time. Note that all time stamps must be in the GMT time zone.

```
<$setExpires("Sat, 02 Aug 2003 24:00:00 GMT")$>
```

This is a way to use other Idoc Script functions to set the expiration date to 14 days from the current date.

```
<$date=formatDateWithPattern(dateCurrent(14), "EEE, dd MMM yyyy")$>
<$setExpires(date & " 24:00:00 GMT"))$> >
```

See Also

- setHttpHeader()

setHttpHeader()

This function can be used to set any HTTP header. Values include, but are not limited to, `Cache-Control`, `Content-Type`, `Last-Modified`, or any of the other acceptable HTTP headers.

Refer to the W3 Protocols for more information:

`http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3`

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

Takes two parameters. For example:

```
setHttpHeader(headerName, headerValue)
```

Parameter	Description
headerName	The name of a valid HTTP header.
headerValue	The value for the header.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

This sets the content-type to XML, to enable your browser to render it with the passed XSLT.

```
<$setHttpHeader("Content-Type", "text/xml")$>
```

See Also

- setMaxAge()

setMaxAge()

This function can be used to set an absolute time for an expiration of a page. This is best used for pages with a long life that may be requested frequently. Due to problems in time synchronization between the client and the server, it is not totally reliable, especially for short-lived caches. The data must follow one of the acceptable HTTP date formats.

Refer to the W3 Protocols for more information:

http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

The only parameter is the lifetime of this page in seconds.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

This will expire the content of the page immediately:

```
<$setMaxAge(0)$>
```

This will set the cache lifetime for this page to 10 minutes, and tell the browser that it absolutely must try to refresh the page after that time is up:

```
<$setMaxAge(600)$>
<$setHttpHeader("Cache-Control", "must-revalidate")$>
```

See Also

- setHttpHeader()

setResourceInclude()

Defines an include.

This function allows dynamically constructed script to be assigned to an include (much as the eval function enables such script to be parsed and evaluated).

If the specified include already exists, the *super* keyword can be used in the new script to refer to the existing include.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.13, "Idoc Script"

Parameters

Takes two parameters:

- The first parameter is the name of the include.
- The second parameter is a string that defines the contents of the include. This string can contain Idoc Script.

Output

Returns 0 if the include could not be parsed, 1 if it can be parsed.

Use `exec` to suppress this behavior.

Example

Uses the string "My name is *resource include*" to dynamically construct script:

```
<$setResourceInclude("my_name","My name is <$my_name$>")$>
```

Uses the `super` keyword to modify the `std_display_field` include:

```
<$myInclude="<span class='field'><$include super.std_display_field$></span>"
<$setResourceInclude("std_display_field",myInclude)$>
```

The following example suppresses the returned result:

```
<$exec setResourceInclude("std_display_field",myInclude)$>
```

setValue()

The target can either be `#local` or the name of a ResultSet. If the target is `#local`, then the parameter *name* identifies the name of a local data variable whose value is to be set by the parameter *value*. If the target area is nonempty and is not `#local`, then a field in a ResultSet named by the parameter *target* is being set. If the ResultSet exists and is currently on a valid current row, then that particular column's value (column named by the parameter *name*) on that row will be set with the value in the parameter *value*. If the field is currently not a part of that ResultSet, then the field will be added as a new column to the ResultSet and the value will be set. If the target ResultSet is not on a valid row, then this function will have no effect (but an internal exception will be reported on the server output). Note that this function should be contrasted with getValue.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.18, "ResultSets"

Parameters

Takes three parameters:

- The first parameter is the target area to be set (either #local or the name of a ResultSet)
- The second parameter is the name of key that holds the value to be set (either a local variable or a ResultSet field)
- The third parameter is the value to be set.

Output

- Returns TRUE if the function is successful.
- Returns FALSE if the function fails.

Example

```
<$setValue(target, name, value)$>
```

SingleGroup

Evaluates if the current revision is in a contributor step of a Basic workflow.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.3, "Content Items"
- Section 4.2.26, "Workflow"

Output

- Returns TRUE if the revision is in a contributor step.
- Returns FALSE if the revision is not in a contributor step.

Example

Evaluates if revision is in a contributor step:

```
<$if not SingleGroup$>
```

SourceID

Provides the Content ID for the current dynamic server page.

This variable returns the same value as `ref:dID`. For more information, see Section 3.6.4, "Metadata References in Dynamic Server Pages."

Type and Usage

- Section 4.1.6.3, "Content Read-Only Variable"
- Section 4.2.3, "Content Items"

Output

Returns the Content ID as a string.

Example

Sets the *dID* variable to the value of the current HCSF page:

```
<input type=hidden name="dID" value="<$SourceID$>">
```

StatusCode

Indicates if the last function call was successful or failed. This variable is set at the end of a service call.

Certain functions set this variable to zero (0) upon success.

Certain functions set this variable to a negative numeric value (-1) upon failure. Typically, the abortToErrorPage function is called to display an error page with the StatusMessage value from the most recent function call.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.5, "Content Server"

Output

- Returns 0 if the last function call was successful.
- Returns -1 if the last function call failed.

Example

Returns the current status code:

```
<$StatusCode$>
```

See Also

- abortToErrorPage()
- getUserValue()
- hasAppRights()
- StatusMessage

StatusMessage

Defines the error message for the last function call. This variable is set at the end of a service call, but it can be set during a service call.

This variable is typically displayed on an error page when the abortToErrorPage function is called.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.5, "Content Server"

Output

- Returns OK if the last function call was successful.
- Returns a localized error message if the last function call failed.

Example

Returns the current status message:

```
<$StatusMessage$>
```

See Also

- abortToErrorPage()
- StatusCode

stdSecurityCheck()

Checks if the current user has access to the requested content item based on the standard security model.

The active data is checked to determine if the standard (or default) security model allows the user to have access to the content item. This enables a custom implementation of security to still execute the standard security model as a baseline.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"

Parameters

None.

Output

- Returns TRUE if the current user has access to the content item.
- Returns FALSE if the current user does not have access to the content item.

Example

Compares the permission level of the user to the requested content item:

```
<$stdSecurityCheck()$>
```

strCenterPad()

Pads equal space on both sides of a string. An equal number of spaces is added to each side of the string to make it at least the specified length. A character will be added to the length of the string if required.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is the string.
- The second parameter is the string length.

Output

Returns a string with spaces on both sides.

Example

Pads equal space on each side and creates a string seven characters long, using the form <space><space>inf<space><space>:

```
<$strCenterPad("inf", 7)$>
```

Pads equal space on each side and creates a string nine characters long, using the form <space><space><space>inf<space><space><space>:

```
<$strCenterPad("inf", 8)$>
```

Pads equal space on each side and creates a string nine characters long, using the form <space><space><space>inf<space><space><space>:

```
<$strCenterPad("inf", 9)$>
```

strCommaAppendNoDuplicates()

This function appends a new token to an existing string. It returns the string plus a comma plus the new token. If the token already exists in the string, it is not added.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is the string.
- The second parameter is the token.

Example

```
<$myString = strCommaAppendNoDuplicates(myString, "A")$>
<$myString = strCommaAppendNoDuplicates(myString, "B")$>
<$myString = strCommaAppendNoDuplicates(myString, "C")$>
<$myString = strCommaAppendNoDuplicates(myString, "A")$>
<$myString = strCommaAppendNoDuplicates(myString, "B")$>
```

strConfine()

Confines a string to a maximum length and appends padding if necessary.

If the string equals or is shorter than the specified length, it is unaffected.

If the string is longer than the specified length, it is shortened and three padding characters are appended to equal the specified length. The character used for padding can be specified by changing the `StrConfineOverflowChars` variable.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is the string.
- The second parameter is the maximum string length.

Output

Returns a string.

Example

Confines the string and appends three padding characters (dots) to make it a string five characters long, using the form `in<dot><dot><dot>`:

```
<$strConfine("inform", 5)$>
```

Confines the string and appends padding characters (dots) to make it a string five characters long, using the form `i<dot><dot><dot><dot>`:

```
<$strConfine("i", 5)$>
```

Confines the string to a maximum of ten characters in length, so this six-character string is unaffected:

```
<$strConfine("inform", 10)$>
```

Confines the string to a maximum of six characters in length, so this six-character string is unaffected:

```
<$strConfine("inform", 6)$>
```

See Also

- StrConfineOverflowChars

StrConfineOverflowChars

Defines a string padding character.

Defines the character used for padding by the `strConfine` Idoc Script function.

The default is a period (dot).

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.22, "Strings"

Location

IntradocDir`/config/config.cfg`

Example

Used as a configuration entry:

```
StrConfineOverflowChars=.
```

See Also

- strConfine()

strEquals()

Checks if two strings are equal, including case.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is a string.
- The second parameter is the string to be compared to the first parameter.

Output

- Returns TRUE if the strings are equal.
- Returns FALSE if strings are not equal.

Example

Evaluates whether the strings are equal and returns TRUE (1):

```
<$strEquals("Home","Home")$>
```

Evaluates whether the strings are equal and returns FALSE (0):

```
<$strEquals("home","Home")$>
```

See Also

- regexMatches()
- regexReplaceAll()
- regexReplaceFirst()
- strEquals()

strEqualsIgnoreCase()

Checks if two strings are equal, not including case.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is a string.
- The second parameter is the string to be compared to the first parameter.

Output

- Returns TRUE if the strings are equal.
- Returns FALSE if strings are not equal.

Example

Evaluates whether the strings are equal and returns TRUE (1):

```
<$strEqualsIgnoreCase("home","Home")$>
```

Evaluates whether the strings are equal and returns FALSE (0):

```
<$strEqualsIgnoreCase("home","page")$>
```

See Also

- strEquals()

strGenerateRandom()

This function generates a random string of hexadecimal characters. It outputs as many characters as specified by the `length` parameter. If length is not specified, it defaults to 16 characters.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes the length of the string as a parameter.

strIndexOf()

Determines if one string is a substring of another.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is a string.
- The second parameter is the possible substring.

Output

- If the second string is a substring of the first, returns an index value that indicates where in the first string the substring begins. The first character has an index value of 0.
- If the second string is not a substring of the first, returns a value of -1.

Example

Evaluates whether `xy` is a substring of `xyz` and returns the index value 0:

```
<$if strIndexOf("xyz","xy") >=0$> check for substring <$endif$>
```

Evaluates whether `yz` is a substring of `xyz` and returns the index value 1:

```
<$if strIndexOf("xyz","yz") >=0$> check for substring <$endif$>
```

Evaluates whether `ab` is a substring of `xyz` and returns the index value –1 to indicate that this is not a substring:

```
<$if strIndexOf("xyz","ab") >=0$> check for substring <$endif$>
```

strLeftFill()

Fills the left side of a string with characters to make it a specified length.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes three parameters:

- The first parameter is the string.
- The second parameter is the fill character.
- The third parameter is the length.

Output

Returns a string, left-filled with the specified character if necessary,.

Example

Formats the string `sleep` by left-filling with the character `Z` to ten spaces. This returns the string `ZZZZZsleep`:

```
<$strLeftFill("sleep",'Z',10)$>
```

Returns the string *sleep*:

```
<$strLeftFill("sleep",'Z',4)$>
```

strLeftPad()

Pads extra space to the left of a string to make it a specified length.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is the string.
- The second parameter is the length.

Output

Returns a string, left-filled with spaces if necessary.

Example

Pads extra space on the left to make it a string five characters long, using the form `<space><space>inf`:

```
<$strLeftPad("inf", 5)$>
```

Returns the string *information*:

```
<$strLeftPad("information", 5)$>
```

strLength()

Evaluates the length of a string.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns an integer value.

Example

Evaluates the length of the string `home` and returns the integer 4:

```
<$strLength("home")$>
```

strLower()

Formats a string in all lowercase letters.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string in all lowercase letters.

Example

Evaluates the string `Home` and returns *home*.

```
<$strLower("Home")$>
```

strRemoveWs()

Removes empty spaces from a string.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string with no spaces.

Example

Formats the string `h o m e` as the string `home`:

```
<$strRemoveWs("h o m e")$>
```

strReplace()

Replaces an existing substring with another substring.

If there are multiple occurrences of the substring to be replaced, they will all be replaced by the new substring.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes three parameters:

- The first parameter is the string on which the substitution will be performed.
- The second parameter is the substring to be replaced.
- The third parameter is the substring that will replace the existing substring.

Output

Returns a string.

Example

Replaces the word `classified` with `restricted` and results in the string `This document is restricted`.

```
<$strReplace("This document is classified.","classified","restricted")$>
```

Replaces the slashes in the date with periods, giving a date in the form `6.20.2001`:

```
<$strReplace(formatDateOnly(dateCurrent()),"/",".")$>
```

strReplaceIgnoreCase()

Replaces an existing substring with another substring, performing the search without using case sensitivity.

If there are multiple occurrences of the substring to be replaced, they will all be replaced by the new substring.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes three parameters:

- The first parameter is the string on which the substitution will be performed.
- The second parameter is the substring to be replaced.
- The third parameter is the substring that will replace the existing substring.

Output

Returns a string.

Example

```
<$hello = "Hello world! I love the world!"$>
<$strReplaceIgnoreCase(hello, "WORLD", "universe")$>
```

strRightFill()

Fills the right side of a string with characters to make it a specified length.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes three parameters:

- The first parameter is the string.
- The second parameter is the fill character.
- The third parameter is the length.

Output

Returns a string, right-filled with the specified character if necessary,.

Example

Formats the string `sleep` by right filling with the character `Z` to ten spaces. This returns the string `sleepZZZZZ`:

```
<$strRightFill("sleep",'Z',10)$>
```

Returns the string *sleep*:

```
<$strRightFill("sleep",'Z',4)$>
```

strRightPad()

Pads extra space to the right of a string to make it a specified length.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes two parameters:

- The first parameter is the string.
- The second parameter is the length.

Output

Returns a string, right-filled with spaces if necessary.

Example

Pads extra space on the right to make it a string five characters long, using the form `inf<space><space>`:

```
<$strRightPad("inf", 5)$>
```

Returns the string *information*:

```
<$strRightPad("information", 5)$>
```

strSubstring()

Retrieves a substring from a string.

The first character has an index value of 0.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

Takes either two or three parameters:

- The first parameter is the string.
- The second parameter is a number representing the start index.
- The third optional parameter is a number representing the stop index.

Output

Returns a substring.

- If the stop index parameter is specified, the substring starting with the character at the start index and ending one character before the stop index is returned.
- If the stop index parameter is not specified, the substring from the start index to the end of the string is returned.

Example

Retrieves the first two characters of the string *my*:

```
<$strSubstring("mystring",0,2)$>
```

Retrieves the string after the second character *string*:

```
<$strSubstring("mystring",2)$>
```

See Also

- regexReplaceAll()
- regexReplaceFirst()

strTrimWs()

Removes spaces from the beginning and end of a string.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string with no leading or trailing spaces.

Example

Formats the string `" homepage "` as the string `homepage`:

```
<$strTrimWs(" homepage ")$>
```

strUpper()

Formats a string in all uppercase letters.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string in all uppercase letters.

Example

Evaluates the string `Home` and returns `HOME`.

```
<$strUpper("Home")$>
```

SysAdminAddress

Defines the system administrator e-mail address.

This is the address used in workflow and subscription notification e-mails that come from the Content Server instance.

Returns a string.

There is no default value.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.24, "Users"
- Section 4.2.5, "Content Server"

Location

- System Properties, Internet tab, **Administrator Mail Address**
- Admin Server, Internet Configuration, **Administrator Mail Address**
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting:

```
SysAdminAddress=admin@example.com
```

As script, returns the value of the configuration setting:

```
<$SysAdminAddress$>
```

See Also

- "MailServer" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- "SmtpPort" in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*

TemplateClass

Retrieves the classification of the template for the current page.

This variable makes it possible to create conditional content in a template based on the identity of the template. Use this variable within a template page to determine the source of the pages delivered by the server.

For standard templates, this variable is defined in the *class* column of the `IntradocTemplates` table in the `templates.hda` file.

For search results templates, this variable evaluates to `Results`.

For report templates, this variable evaluates to `Reports`.

For dynamic server pages, this variable is typically `IdcDynamicFile`.

Type and Usage

- Section 4.1.6.1, "Template Read-Only Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.23, "Templates"

Output

Returns a string.

Example

This markup displays a table of template information on the page:

```
<TABLE>
    <TR><TD>Template Name</TD>
    <TD><$TemplateName$></TD></TR>
    <TR><TD>Template Class</TD>
    <TD><$TemplateClass$></TD></TR>
    <TR><TD>Template Type</TD>
    <TD><$TemplateType$></TD></TR>
    <TR><TD>Template File Path</TD>
    <TD><$TemplateFilePath$></TD></TR>
</TABLE>
```

See Also

- TemplateFilePath
- TemplateName
- TemplateType

TemplateFilePath

Retrieves the path from where the template was actually loaded.

This variable makes it possible to create conditional content in a template based on the identity of the template. Use this variable within a template page to determine the source of the pages delivered by the server.

Type and Usage

- Section 4.1.6.1, "Template Read-Only Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.23, "Templates"

Output

Returns the path as a string.

Example

This markup displays a table of template information on the page:

```
<TABLE>
    <TR><TD>Template Name</TD>
    <TD><$TemplateName$></TD></TR>
    <TR><TD>Template Class</TD>
    <TD><$TemplateClass$></TD></TR>
    <TR><TD>Template Type</TD>
    <TD><$TemplateType$></TD></TR>
    <TR><TD>Template File Path</TD>
    <TD><$TemplateFilePath$></TD></TR>
</TABLE>
```

See Also

- TemplateClass
- TemplateName
- TemplateType

TemplateName

Retrieves the internal name of the template for the current page. For example, DOC_INFO or CHECKIN_NEW_FORM.

This variable makes it possible to create conditional content in a template based on the identity of the template. Use this variable within a template page to determine the source of the pages delivered by the server.

Type and Usage

- Section 4.1.6.1, "Template Read-Only Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.23, "Templates"

Output

Returns the template name as a string.

Example

This markup displays a table of template information on the page:

```
<TABLE>
    <TR><TD>Template Name</TD>
    <TD><$TemplateName$></TD></TR>
    <TR><TD>Template Class</TD>
    <TD><$TemplateClass$></TD></TR>
    <TR><TD>Template Type</TD>
    <TD><$TemplateType$></TD></TR>
    <TR><TD>Template File Path</TD>
    <TD><$TemplateFilePath$></TD></TR>
</TABLE>
```

See Also

- TemplateClass
- TemplateFilePath
- TemplateType

TemplateType

Provides the template type for the current page.

This variable makes it possible to create conditional content in a template based on the identity of the template. Use this variable within a template page to determine the source of the pages delivered by the server.

For standard templates, this variable is defined in the `formtype` column of the `IntradocTemplates` table in the `templates.hda` file.

For search results templates, this variable is defined in the `formtype` column of the `SearchResultTemplates` table in the `templates.hda` file.

For report templates, this variable is defined in the `datasource` column of the `IntradocReports` table in the `reports.hda` file.

For dynamic server pages, this variable is either `hcsp` or `hcsf`.

Type and Usage

- Section 4.1.6.1, "Template Read-Only Variables"
- Section 4.2.16, "Page Display"
- Section 4.2.23, "Templates"

Output

Returns the template type as a string.

Example

This markup displays a table of template information on the page:

```
<TABLE>
    <TR><TD>Template Name</TD>
    <TD><$TemplateName$></TD></TR>
    <TR><TD>Template Class</TD>
    <TD><$TemplateClass$></TD></TR>
    <TR><TD>Template Type</TD>
    <TD><$TemplateType$></TD></TR>
    <TR><TD>Template File Path</TD>
    <TD><$TemplateFilePath$></TD></TR>
</TABLE>
```

See Also

- TemplateClass
- TemplateFilePath
- TemplateName

toInteger()

Converts a string to an integer.

> **Note:** For string concatenation, string inclusion, and simple comparison, use the string operators described in Section 3.5.2, "Special String Operators."

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns an integer value. If the string does not evaluate to a number, or evaluates to a non-integer number, an error is thrown.

Example

Converts the string 4 to an integer and returns the value 4:

```
<$toInteger("4")$>
```

trace()

Enables logging a debug or trace message to the `IsPageDebug` output. A message can also be output to the console or to the system logs.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.9, "Debugging"

Parameters

Takes one required and two optional parameters:

- The first parameter is the message which is output. The value can be one of the following:
  - A string. If the string is passed as the only parameter, the IsPageDebug variable must be set for this function to execute. The string is added to the accumulated debug trace.
  - `#local`, which prepares a dump of all current local variables.
  - `#all`, which prepares a full HDA dump of all local data, ResultSets, and environment data.
- The second (optional) parameter is the location where the message will be relayed. The values can be the following:
  - `#console`, to display to a console
  - `#log` to log a message in the HTML log files
  - the name of a variable (such as StatusMessage). In that case, the message is appended to the current value.
- The third (optional) parameter is for the appropriate tracing section and is only relevant if the location parameter is set to `#console`.

Example

The following example logs the string `message` to the system console, which is always logged:

```
<$trace("message", "#console")$>
```

The following example logs the string `message` to the system console in the `pagecreation` tracing section.

```
<$trace("message", "#console", "pagecreation")$>
```

The following example logs the string `message` to the HTML Content Server log file.

```
<$trace("message", "#log")$>
```

The following example dumps all local variables and their values to the system console.

```
<$trace("#local", "#console")$>
```

The following example dumps all local variables, ResultSets, and environment variables to the system console.

```
<$trace("#all", "#console")$>
```

The following example dumps all data to the variable `MyTraceDump`, which can then be displayed on the page. This is useful for HCSP developers who may not have the appropriate access rights to view the console logs.

```
<$trace("#all", "MyTraceDump")$>
<$MyTraceDump$>
```

See Also

- IsPageDebug

UploadApplet

Enables the multiple file Upload Applet.

> **Important:** This setting is only for use by custom legacy versions of the Upload Applet. It should not be enabled when the standard MultiUpload variable is enabled.

When set to TRUE, the Upload Applet is enabled so that multiple files can be zipped and checked in as a single content item.

When set to FALSE, the Upload Applet is disabled.

Default is FALSE.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.5, "Content Server"
- Section 4.2.2, "Clients"

Location

IntradocDir/config/config.cfg

Example

As a configuration setting, enables the Upload Applet:

```
UploadApplet=true
```

As script, evaluates the condition of the Upload Applet:

```
<$UploadApplet$>
```

See Also

- DownloadApplet
- MultiUpload

url()

Formats a string for use in a URL.

This function converts blank spaces and reserved characters to an escape sequence.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string formatted for use in a URL.

Example

Formats the string `home page` as `home%20page`:

```
<$url("home page")$>
```

Formats the string `home/page` as `home%2fpage`:

```
<$url("home/page")$>
```

Formats the string `home?page` as `home%3fpage`:

```
<$url("home?page")$>
```

urlEscape7Bit()

This function returns a URL-encoded version of the string parameter. This is similar to the `url` function but it only encodes characters that are not 7-bit clean (ASCII). Therefore this function can be called repeatedly on the same string.

If the `url` function is used to double encode a string, every % character is encoded to %25.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string formatted for use in a URL.

Example

```
<$path = "mÿ fïlë.txt"$>
20fïlë.txt = <$url(path)$>
mÿ%2520fïlë.txt = <$url(url(path))$>
m%ff%20f%efl%eb.txt = <$urlEscape7Bit(url(path))$>
m%ff%20f%efl%eb.txt= <$urlEscape7Bit(urlEscape7Bit(url(path)))$>
```

See Also

- url()

UseHtmlOrTextHighlightInfo

Checks whether full-text search keyword highlighting is enabled and the file format supports highlighting (such as PDF, HTM, or TXT).

If the `EnableDocumentHighlight` setting is `TRUE`, keyword highlighting is enabled.

Type and Usage

- Section 4.1.2, "Dynamic Variables"
- Section 4.2.20, "Searching"

Output

- Returns `TRUE` if highlighting is enabled.
- Returns `FALSE` if highlighting is not enabled.

Example

Returns search keyword highlighting status:

```
<$UseHtmlOrTextHighlightInfo$>
```

See Also

- UseXmlUrl
- EnableDocumentHighlight

UserAccounts

Retrieves a comma-delimited list of accounts the current user is assigned to.

- The `#none` entry indicates privileges to content items that have no account assigned.
- The `#all` entry indicates privileges to all accounts.

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

Returns the account list as a string.

Example

For example:

```
UserAccounts=BOS,SEA,MSP/Gen
```

This markup displays user variables on a page:

```
<$if UserName$>
    Logon Name: <$UserName$><BR>
    User Name: <$UserFullName$><BR>
    E-mail Address: <$UserAddress$><BR>
    Accounts: <$UserAccounts$><BR>
<$endif$>
```

UserAddress

Retrieves the email address of the current user.

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.24, "Users"

Output

Returns the email address as a string.

Example

This markup displays user variables on a page:

```
<$if UserName$>
    Logon Name: <$UserName$><BR>
    User Name: <$UserFullName$><BR>
    E-mail Address: <$UserAddress$><BR>
    Default Account: <$UserDefaultAccount$><BR>
<$endif$>
```

UserAppRights

Checks the application rights of the current user.

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

Returns a bit flag number specifying the user's rights:

Admin Application	Bit Flag
User Admin	1
Web Layout Editor	2
Repository Manager	4
Workflow Admin	8
Configuration Manager	16
Archiver	32

Example

Displays application rights of the current user:

```
<$UserAppRights$>
```

For example, the following combinations would result in the following numbers:

- User Admin + Web Layout Editor = 3
- Repository Manager + Workflow Admin + Archiver = 44
- All six applications = 63

See Also

- AdminAtLeastOneGroup
- UserIsAdmin

UserDefaultAccount

Retrieves the default account for the current user.

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

Returns the account name as a string.

Example

This markup displays user variables on a page:

```
<$if UserName$>
    Logon Name: <$UserName$><BR>
    User Name: <$UserFullName$><BR>
    E-mail Address: <$UserAddress$><BR>
    Default Account: <$UserDefaultAccount$><BR>
<$endif$>
```

UserFullName

The full name of the current user.

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.24, "Users"

Output

- Returns the user's full name as a string.
- If the user is not logged in, returns the string `anonymous`.

Example

This markup displays user variables on a page:

```
<$if UserName$>
    Logon Name: <$UserName$><BR>
    User Name: <$UserFullName$><BR>
    E-mail Address: <$UserAddress$><BR>
    Default Account: <$UserDefaultAccount$><BR>
<$endif$>
```

userHasAccessToAccount()

This function returns TRUE if the user has access to a named account at a specific privilege level. The privilege is a one-character representation of the access level, as follows:

- R: Read. Specified as 1.
- W: Write. Specified as 2.
- D: Delete. Specified as 4.
- A: Administration. Specified as 8.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Parameters

Takes two parameters:

- The first is the account name.
- The second is the access level to be checked.

Output

- Returns TRUE if the current user has the specified access.
- Returns FALSE if the user does not have the specified access.

Example

Evaluates whether the user has Read access to the specified account:

```
<$userHasAccessToAccount("profile_account", 1)$>
```

Evaluates whether the user has Read and Write access to the specified account:

```
<$userHasAccessToAccount("profile_account", 3)$>
```

userHasGroupPrivilege()

This function returns TRUE if the user has the specified privilege to the specified group. The privilege is a one-character representation of the access level, as follows:

- R: Read
- W: Write
- D: Delete
- A: Administration

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Parameters

Takes two parameters:

- The security group to be checked.
- The access level to be checked.

Output

- Returns TRUE if the current user does have the specified access.
- Returns FALSE if the user does not have the specified access.

Example

Evaluates whether the user has the specified role:

```
<$userHasGroupPrivilege("Public", "R")$>
<$userHasGroupPrivilege("Secure", "A")$>
```

userHasRole()

Checks if the current user has a particular role.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Parameters

The only parameter is the name of a role.

Output

- Returns TRUE if the current user does not have the specified role.
- Returns FALSE if the user does not have the specified role.

Example

Evaluates whether the user has the specified role:

```
<$userHasRole("admin")$>
```

UserIsAdmin

Checks if the current user has rights to any administration applets.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

- Returns TRUE if the user has any administrative rights.
- Returns FALSE if the user does not have any administrative rights.

Example

Evaluates administrative rights of user:

```
<$UserIsAdmin$>
```

See Also

- AdminAtLeastOneGroup
- UserAppRights

UserLanguageID

Returns the two-letter code that represents the user's preferred language, as in `en` for English, `fr` for French, or `ja` for Japanese.

This is useful when constructing URLs to localized content.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.15, "Localization"

Output

Returns the code for the preferred language.

UserLocaleId

Returns the full name for a user's locale, as in `English-US`, `English-UK` or `Japanese`. The locale contains information about language, date, and number formatting.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.2.15, "Localization"

Output

Returns the name of the preferred language.

UserName

Retrieves the username of the current user.

Type and Usage

- Section 4.1.6, "Read-Only Variables"
- Section 4.2.24, "Users"

Output

- Returns the username as a string.
- If the user is not logged in, evaluates to the string `anonymous`.

Example

This markup displays user variables on a page:

```
<$if UserName$> Logon Name: <$UserName$><BR>
    User Name: <$UserFullName$><BR>
    E-mail Address: <$UserAddress$><BR>
    Default Account: <$UserDefaultAccount$><BR>
<$endif$>
```

UserRoles

Retrieves a comma-delimited list of roles the current user belongs to.

Type and Usage

- Section 4.1.6.2, "User Read-Only Variables"
- Section 4.2.21.1, "Internal Security"
- Section 4.2.24, "Users"

Output

Returns the user role list as a string.

Example

For example:

```
PublicContributor,ClassifiedConsumer
```

References the list of user roles:

```
<$if UserRoles$>
    <$include optional_field$>
<$endif$>
```

UseSSL

Enables the secure sockets layer (SSL).

- This setting affects the variables `HttpWebRoot` and `HttpCgiPath`.
- Use the Secure Sockets Layer only if you are using an SSL-enabled web server.
- When set to `TRUE`, SSL is used (`https` instead of `http`).

Default is `FALSE`.

Type and Usage

- Configuration variables, described in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*
- Section 4.2.21.2, "External Security"
- Section 4.2.25, "Web Servers"

Location

- System Properties, Internet tab, Use Secure Sockets Layer
- *IntradocDir*`/config/config.cfg`

Example

As a configuration setting, enables SSL:

```
UseSSL=true
```

As script, returns the value of the configuration setting:

```
<$UseSSL$>
```

See Also

- HttpCgiPath
- HttpWebRoot

UseXmlUrl

Checks if search keyword highlighting for XML documents is enabled. When set to TRUE, an XML URL is constructed when performing a keyword search.

Type and Usage

- Section 4.1.1, "Conditional Dynamic Variables"
- Section 4.1.6, "Read-Only Variables"

Output

- Returns TRUE if XML keyword highlighting is enabled.
- Returns FALSE if XML keyword highlighting is not enabled.

Example

Returns the status of search keyword highlighting for XML documents.

```
<$UseXmlUrl$>
```

See Also

- UseHtmlOrTextHighlightInfo
- EnableDocumentHighlight

utGetValue()

Returns the value of a user property from a specified personalization (user topic) file.

User topic files are HDA files that are located in the *IntradocDir*/data/users/profiles/*us*/*username*/ directories.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.17, "Personalization"

Parameters

Takes two parameters:

- The first parameter is the name of a user topic HDA file.
- The second parameter is the name of a property defined in the user topic file.

Output

Returns the value of the property in the specified user topic file.

Example

Returns the value of the `emailFormat` property in the current user's `pne_portal.hda` file:

```
<$utGetValue("pne_portal","emailFormat")$>
```

See Also

- utLoad()
- utLoadResultSet()

utLoad()

Loads a personalization (user topic) file so it is available for use by the utGetValue and `utLoadResultSet` functions.

User topic files are HDA files that are located in the *`IntradocDir`*`/data/users/profiles/`*`us`*`/`*`username`*`/` directories.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.17, "Personalization"

Parameters

The only parameter is the name of a user topic HDA file.

Example

Loads the current user's `wf_in_queue.hda` file and makes it available to other user topic functions:

```
<$utLoad("wf_in_queue")$>
```

See Also

- utGetValue()
- utLoadResultSet()

utLoadDocumentProfiles()

Used to retrieve information about a user's current My Check In and My Search links for content profiles.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.4, "Content Profiles"

Parameters

Takes two parameters:

- The user topic name.
- The name of the ResultSet containing the content profiles available for the user on the personalization links.

Example

```
<$utLoadDocumentProfiles("pne_portal", "PneDocumentProfiles")
```

See Also

- loadDocumentProfile()

utLoadResultSet()

Loads a ResultSet specified in a personalization (user topic) file into the DataBinder.

User topic files are HDA files that are located in the *IntradocDir*/data/users/profiles/*us*/*username*/ directories.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.17, "Personalization"
- Section 4.2.18, "ResultSets"

Parameters

Takes two parameters:

- The first parameter is the name of a user topic HDA file.
- The second parameter is the name of a ResultSet defined in the user topic file.

Output

- Returns TRUE if the ResultSet is successfully loaded into the DataBinder.
- Returns FALSE if the function fails.

Example

Loads the `SavedQueries` ResultSet from the current user's `pne_portal.hda` file into the DataBinder:

```
<$utLoadResultSet("pne_portal","SavedQueries")$>
```

See Also

- utGetValue()
- utLoad()

valueStyle

Specifies the style of the value for the metadata field.

The following SPAN styles are typically used for values. They are defined in the *std_style_declaration* include in the *WC_CONTENT_ORACLE_HOME*/shared/config/resources/std_page.htm resource file:

- tableEntry
- xxsmall
- strongHighlight

Type and Usage

- Section 4.1.5.2.1, "Field Information Variables"
- Section 4.2.12, "Field Display"

Output

Returns the name of the value style.

Example

Sets the value style:

```
<$if isFieldInfoOnly$>
    <$if valueStyle$>
        <$fieldValueStyle = valueStyle$>
    <$else$>
        <$fieldValueStyle = "tableEntry"$>
    <$endif$>
<$endif$>
```

Uses the `xxsmall` style for the field value when the Download applet is used:

```
<$if useDownloadApplet$>
    <$valueStyle="xxsmall", fieldValue = strTrimWs(inc("download_file_by_
    applet_form_content"))$>
<$else$>
    <$fieldValue = strTrimWs(inc("doc_file_get_copy"))$>
<$endif$>
```

See Also

- fieldValueStyle

wfAction

Retrieves the action that is currently being performed on the revision.

- This variable is set after the Exit event of a workflow step, so it is normally evaluated during the Entry event of the next step.
- The possible actions are:
  - APPROVE
  - REJECT
  - CHECKIN
  - CONVERSION
  - META_UPDATE
  - TIMED_UPDATE
  - RESUBMIT

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the name of the action as a string.

Example

If the revision is in the process of being rejected, notify the original author:

```
<$if wfAction like "REJECT"$>
    <$wfSet("wfJumpName", "notifyAuthor")$>
    <$wfSet("wfJumpEntryNotifyOff", "0")$>
    <$wfNotify(dDocAuthor, "user")$>
<$endif$>
```

If the revision is currently being approved but the metadata value of `xDept` is not HR, terminate the approval and display an error page:

```
<$if wfAction like "APPROVE" and not(xDept like "HR")$>
    <$abortToErrorPage("The revision is not in HR.")$>
<$endif$>
```

wfAddActionHistoryEvent()

Inserts an event into the `WorkflowActionHistory` table, in the workflow's companion data. The three parameters to this function are required. The rest of the row's values are computed or inherited from local data.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes three parameters:

- The first parameter is the workflow action.
- The second parameter is the message associated with the action.
- The third parameter is a comma-delimited list of users for the current workflow step,. See `wfComputeStepUserList` for a function which computes the current list.

See Also

- wfComputeStepUserList()

wfAdditionalExitCondition

Retrieves the exit condition that has been defined for the current step.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the exit condition as a string.

Example

The following code in the *IdcHomeDir*/resources/core/templates/workflow_info.htm template page is used to display the exit condition on the Workflow Step Information page:

```
<$wfDisplayCondition(dWfName, currentStepName, "wfAdditionalExitCondition")$>
```

Typical exit condition output for this variable would look like:

```
dSecurityGroup like "Secure"
```

wfAddUser()

Adds a user, alias, or workflow token to the list of reviewers for a workflow step. This function can only be used inside a token.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters:

- The first parameter indicates the user name, alias, or token name.
- The second parameter indicates the type, either *user* or *alias*.

Example

Adds the user `mjones` as a reviewer:

```
<$wfAddUser("mjones", "user")$>
```

Adds the original author token and the `hr` alias as reviewers:

```
<$wfAddUser(dDocAuthor, "user")$>
<$wfAddUser("hr", "alias")$>
```

wfComputeStepUserList()

Computes the list of users from the current step in the workflow.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Output

Returns a comma-delimited list of users.

See Also

- wfAddActionHistoryEvent()

wfCurrentGet()

Retrieves a local state value from the companion file.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

The only parameter is the key.

Output

Returns the local state value from the companion file.

Example

Returns the value of the local key `<step_name>@<workflow_name>.myKey`:

```
<$wfCurrentGet("myKey")$>
```

See Also

- wfGet()

wfCurrentSet()

Sets the local state value of a key in the companion file.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters:

- The first parameter is the key.
- The second parameter is the value.

Example

Sets the key `<step_name>@<workflow_name>.myKey` to `myValue`:

```
<$wfCurrentSet("myKey", "myValue")$>
```

See Also

- wfSet()

wfCurrentStep()

Retrieves the name of a step relative to the current step.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

The only parameter is an integer that represents the number of steps relative to the current step.

Output

- Returns a step name.
- Reaching the beginning or the end of the workflow returns the name of the first or last step, respectively.

Example

Returns the current step name:

```
<$wfCurrentStep(0)$>
```

Returns the previous step name:

```
<$wfCurrentStep(-1)$>
```

Returns the next step name:

```
<$wfCurrentStep(1)$>
```

To actually move to the next step you must set up a jump, as in this example:

```
<$wfSet("wfJumpName", "move to next step")$>
<$wfSet("wfJumpTargetStep", wfCurrentStep(1))$>
<$wfSet("wfJumpEntryNotifyOff", "0")$>
```

wfDisplayCondition()

Retrieves the exit condition for a workflow step.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes three parameters:

- The first parameter is the workflow name.
- The second parameter is the step name.
- The third parameter is the exit condition to be displayed.

Output

Returns the exit condition expression.

Example

The following code in the *IdcHomeDir*/resources/core/templates/workflow_info.htm template page is used to display the exit condition on the Workflow Step Information page:

```
<$wfDisplayCondition(dWfName, currentStepName, "wfAdditionalExitCondition")$>
```

where:

- `dWfName` is the internal variable for the workflow name.
- `currentStepName` is set earlier in the template page to be equal to `dWfName`, which is the internal variable for the step name.
- `wfAdditionalExitCondition` is the internal variable for the exit condition expression. Typical exit condition output would look like:

  ```
  dSecurityGroup like "Secure"
  ```

wfExit()

Exits a workflow step. This function moves the revision to a particular step in a workflow according to the function parameters and resets the parent list information. To completely exit a workflow, use `wfExit(100,100)` or any parameters that ensure that the revision returns to the parent workflow and then gets moved past the last step in that workflow.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters:

- The first parameter indicates the number of jumps to rewind.
- The second parameter indicates the target step relative to the step determined by the first parameter.

Example

Exits to the parent step in the workflow:

```
<$wfExit(0,0)$>
```

Returns to the previous jump step:

```
<$wfExit(1,0)$>
```

Returns to the previous jump step and moves to the next step in that workflow:

```
<$wfExit(1,1)$>
```

wfGet()

Retrieves a state value from the companion file.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

The only parameter is the state key.

Output

Returns the state value from the companion file.

Example

Returns the current jump name:

```
<$wfGet("wfJumpName")$>
```

See Also

- wfCurrentGet()

wfGetStepTypeLabel

This function takes an internal workflow step type value and turns it into a human-readable label. For example, `:C:CA:CE:` is the three states `:C:` (contribution), `:CA:` (autocontribute or original author step) and `:CE:` (Edit revision). After application of the Idoc Script function, the label might become `"Auto Contributor, Edit Revision"`.

Here are the possible current states.

States	Description
:CA:	Auto contribution step or step that occurs before all administrator defined workflow steps ("original author handles document rejection").
:CE:	Edit current revision. All edits replace current revision.
:CN:	Create new revision. All edit create new revision.
:R:	Review. Document can be approved or rejected.
:C:	Contribution. The document can be edited.

A workflow document step state has all the states that are true for it combined as a single fields with multiple values separated by colons (:) (redundant colons are eliminated). So a contributor/reviewer step that creates new revisions would have the state :R:C:CE:. Order does not matter. But :C: must be present even though it can be presumed by the presence of :CE: (:CE: also implies that the step is a contributor step). The value :C: accurately describes one capability of the current workflow step and the Content Server system does not chase implication rules (:CE: -> :C:) so the Content Server system will see the absence of :C: (even with :CE: present) as a statement that the workflow step does not allow core contributor types of activities (such as checkout or undo checkout).

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Parameters

The only parameter is the internal workflow step type value.

Example

```
<$wfGetStepTypeLabel(stepType)$>
```

wfIsFinishedDocConversion()

Indicates if the document will not be in GENWWW after the current action finishes.

If this Idoc Script function returns true, the conversion process has finished and the content item is no longer in GENWWW.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Output

- Returns TRUE if the document has finished conversion.
- Returns FALSE if the content item is still in GENWWW and has not finished conversion.

Example

```
<$wfIsFinishedDocConversion()$>
```

wfIsNotificationSuppressed()

Returns TRUE if this workflow is currently suppressing all workflow notifications for this particular workflow event. Suppression includes notification into both the workflow in queue and e-mail until either the current action is ended or the `wfSetSuppressNotification` function is used to re-enable notification.

Suppression of notifications is temporary. If notification is enabled or allowed to remain on for a later workflow action, all notifications that were not sent out for the current step are then sent out and workflows in queues are appropriated updated.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Output

- Returns TRUE if notifications are suppressed.
- Returns FALSE if notifications are not suppressed.

Example

```
<$wfIsNotificationSuppressed()$>
```

See Also

- wfSetSuppressNotification()

wfIsReleasable()

Indicates if the document is released as far as the workflow is concerned. The document may be still in GENWW or DONE, but if this Idoc Script function returns true, then the workflow is not preventing the release of the document. If it returns false, then the document will not be released until the workflow allows it to be released. This Idoc Script function takes no parameters. It evaluates the active release state value for any document info that may be present (in Idoc Script terms it checks if `#active.dReleaseState` is not the value `E`).

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Output

- Returns TRUE if the document is available to be released.
- Returns FALSE if the content item is not available to be released. For example, the content item has not completed the check in and/or conversion process.

Example

```
<$wfIsReleasable()$>
```

wfJumpEntryNotifyOff

Turns the jump entry notification on and off.

- If this variable is TRUE, reviewers will not be notified when the jump is entered.
- If this variable is FALSE, reviewers will be notified when the jump is entered.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns TRUE or FALSE.

Example

The following code sets the step entry notification to 0, which means that reviewers will be notified upon jump entry:

```
<$wfSet("wfJumpEntryNotifyOff","0")$>
```

wfJumpMessage

Defines a message that will be included in the notification e-mail that is sent to users when a jump is entered.

- If no message is specified, the e-mail message will include only the information in the e-mail template.
- The jump message can include Idoc Script, which must be executed using the eval function. For example:

```
<$eval(dDocName)$> is ready for your review.
```

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the message as a string.

Example

The following code includes the message *This is the message text* in the e-mail message that notifies users upon jump entry:

```
<$wfSet("wfJumpMessage","This is the message text")$>
```

See Also

- eval()

wfJumpName

Retrieves the name of the current jump.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the jump name as a string.

Example

The following code sets the name of the current jump to `MyJump`:

```
<$wfSet("wfJumpName","MyJump")$>
```

wfJumpReturnStep

Retrieves the name of the step in the parent workflow that the revision returns to upon exiting a workflow after the current jump.

- The return step applies only if the revision properly completes the last step in the workflow that was jumped to and exits the workflow normally. Consequently, the return step is *not* applied when the revision jumps to another workflow.
- In the companion file, the return step is stored in the local key:

  ```
  <step_name>@<workflow_name>.returnStep
  ```

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the step name as a string.

Example

The following code sets the return step to be the next step in the parent workflow:

```
<$wfSet("wfJumpReturnStep",wfCurrentStep(1))$>
```

wfJumpTargetStep

Retrieves the name of the step the revision will jump to if the condition is met.

The target step can be referred to symbolically (such as `wfCurrentStep(1)`) or explicitly (such as `MyStep@MyWorkflow`). It is strongly recommended that you use symbolic references in step event scripts. They make the script easier to modify and reuse.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the step name as a string.

Example

The following code sets the target step explicitly. When the jump criteria is met, the revision jumps to a step named `step_1` in the `Marketing` workflow:

```
<$wfSet("wfJumpTargetStep","step_1@Marketing")$>
```

The following code sets the target step symbolically. When the jump criteria is met, the revision jumps to the first step in the current workflow:

```
<$wfSet("wfJumpTargetStep",WfStart)$>
```

wfLoadDesign()

This function is used to obtain information about the existing steps in a workflow or about exit conditions in a workflow.

This function loads the *IntradocDir*/data/workflow/design/*workflowname*.hda file and returns a ResultSet containing design information for a workflow. The workflowname value corresponds to the value for the dWfName variable, usually available on workflow pages, email templates, and jump scripts.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes one parameter, the name of the workflow.

Output

Returns the information as a ResultSet.

Example

```
<$wfLoadDesign(WorkflowName)$>
```

wfMailSubject

Defines the subject line of a workflow e-mail notification.

If no subject line is specified, the e-mail will use the default subject for the type of notification (review, reject, or workflow started).

Idoc Script can be included in the subject string.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the subject line as a string.

Example

Notifies the original author with the subject line *"Content ID* has been released":

```
<$wfMailSubject="File <$dDocName$> has been released"$>
<$wfNotify(dDocAuthor, "user")$>
```

See Also

- wfNotify()
- wfMessage

wfMessage

Defines a message that will be included in a workflow e-mail notification.

If no message is specified, the e-mail will use the default message for the type of notification (review, reject, or workflow started).

Idoc Script can be included in the message string.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the message as a string.

Example

Notifies the original author with the message `"Content ID has been released"`:

```
<$wfMessage="File <$dDocName$> has been released"$>
<$wfNotify(dDocAuthor, "user")$>
```

See Also

- wfNotify()
- wfMailSubject

wfNotify()

Sends an e-mail message to a specified user, alias, or workflow token.

The wfMailSubject and wfMessage variables can be set to customize the notification message.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters and an optional third parameter:

- The first parameter specifies the user name, alias, or token to be notified.
- The second parameter indicates the type, either *user, alias* or token.
- The optional third parameter specifies the name of the e-mail template to use for constructing the message. (See the *IdcHomeDir*`/resources/core/templates/templates.hda` file for template definitions.)

Example

Notifies the original author:

```
<$wfNotify(dDocAuthor, "user")$>
```

Notifies all users in the `myAlias` alias, using the *IdcHomeDir*`/resources/core/templates/reject_mail.htm` file as a template:

```
<$wfNotify("myAlias", "alias", "WF_REJECT_MAIL")$>
```

See Also

- wfMailSubject
- wfMessage

wfParentList

Retrieves a list (parent list) of the workflow steps that the revision has visited. This tells the system where jumps occurred and how to unwind the steps during an error, a reject, or an exit.

This variable can be used to create conditional statements, but it should not be hard-coded or altered.

If the parent list is unwound due to an error, reject, or exit, steps are removed from the list, so the parent list may not reflect the complete step history.

The parent list is global, and is not localized with a step name.

Steps in the parent list are listed with the most recent step first. Steps are separated with a pound sign (#). An asterisk before a step name indicates that it is a jump step. For example:

```
Step_B@Workflow_2#*Step_A@Workflow_2#Step_1@Workflow_1
```

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Output

Returns the parent list as a string.

Example

One use for the parent list is to simulate the jumps that a content item has visited when you are testing a workflow script. For example, on the Test tab or the Edit Script screen, selecting a content item and clicking Load Item's Workflow State could result in the following line:

```
wfParentList=step_1@Marketing#contribution@Marketing
```

wfReleaseDocument

Causes a workflow to release all outstanding document revisions for a document that are currently being locked by the workflow. Such released revisions are free to be evaluated by the indexing engine and one of the released revisions may be full-text indexed as a result.

This function can only be used in the context of the evaluation of Idoc Script for a workflow step going through a workflow event. The function will have no effect if the document has already been released by the workflow. Note that this function does not cause the document to exit workflow.

Any new revision that is created for the document will be back in a "locked" (unreleasable) state. In other words, this function releases current revisions but has no implications for any new revision that may be created. This function takes no parameters and returns no result. It acts on the current active workflow document.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Example

```
<$wfReleaseDocument()$>
```

wfSet()

Sets a key with a particular value in the companion file.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters:

- The first parameter is the key.
- The second parameter is the value.

Example

Sets the key `wfJumpName` to `MyJump`:

```
<$wfSet("wfJumpName", "MyJump")$>
```

See Also

- wfCurrentSet()

wfSetSuppressNotification()

Sets an internal flag indicating if workflow notification will be sent out during the current action (check in, update, resubmit, and so on).

The suppression is on both e-mail and updates to the workflow in queue. An additional use for this function is to suppress workflow notification until after a document has been converted. This prevents a document from advancing out of the auto-contributor workflow step when the document finishes a conversion.

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes one parameter, the internal flag. If set to 1 (TRUE), notifications are suppressed. If set to 0 (FALSE), notifications are not suppressed.

Example

```
<$wfSetSuppressNotification(1)$>
```

See Also

- wfIsNotificationSuppressed()

WfStart

Sends the revision to the first step in the current workflow. Note that this variable begins with an uppercase W.

Type and Usage

- Section 4.1.8.2, "Workflow Variables"
- Section 4.2.26, "Workflow"

Example

Sets the target step for a jump to restart the workflow:

```
<$wfSet("wfJumpTargetStep",WfStart)$>
```

wfUpdateMetaData()

Defines a metadata value for the current content item revision in a workflow.

The `wfUpdateMetaData` function can be used only for updating custom metadata fields. You cannot use this function to update standard, predefined fields

Type and Usage

- Section 4.1.8.1, "Workflow Functions"
- Section 4.2.26, "Workflow"

Parameters

Takes two parameters:

- The first parameter is the name of the metadata field.
- The second parameter is the value of the metadata field.

Example

Defines "This is my comment." as the value of the Comments field:

```
<$wfUpdateMetaData("xComments", "This is my comment.")$>
```

xml()

Formats a string for use in XML.

This function replaces non-alphanumeric characters with the correct XML sequence that represents the character. For example, the ampersand "&" character is replaced with the "&" sequence.

When using a double-quote character within a string, a backslash "\" must precede the double-quote to display it as a character. If the backslash is not used as an escape flag, the double-quote is interpreted as ending the string.

Type and Usage

- Section 4.1.4, "Global Functions"
- Section 4.2.22, "Strings"

Parameters

The only parameter is the string.

Output

Returns a string formatted for use in a URL.

Example

Escapes the ampersand and returns the XML formatted string, "Me & you."

```
<$xml("Me & you.")$>
```

Escapes the non-alphanumeric characters and returns the XML-formatted string, "Test the ", >, and < characters."

```
<$xml("Test the \", >, and < characters.")$>
```

B

Building a Website

This appendix describes the Web Layout Editor and how it is used to build a website.

The following topics are covered:

- Section B.1, "Planning a Website"
- Section B.2, "Working with Web Pages"
- Section B.3, "Managing Web Pages"
- Section B.4, "Working with Reports"
- Section B.5, "Writing Queries"

B.1 Planning a Website

Administrators are responsible for planning the website. Subadministrators with WebLayout rights can create directory pages for groups and accounts if they have permissions for those groups and accounts.

This section covers these topics:

- Section B.1.1, "The Web Layout"
- Section B.1.2, "Defining the Site Structure and Displaying Criteria"
- Section B.1.3, "Task Sequence"

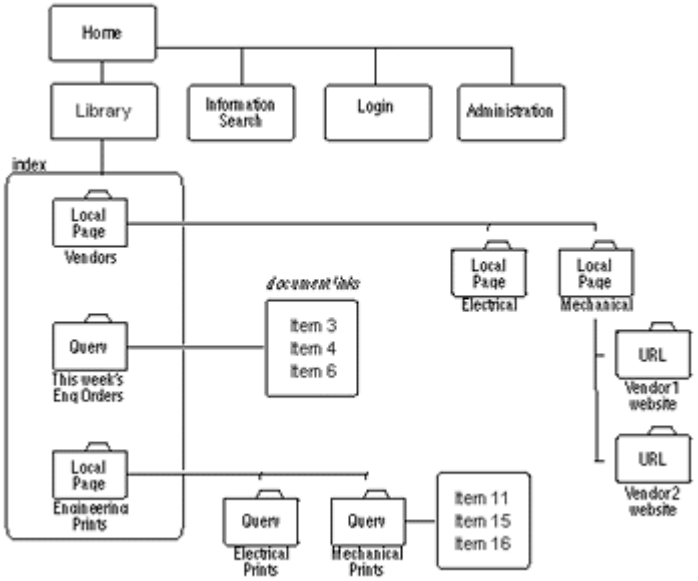
B.1.1 The Web Layout

From the Home page, the Library (Table of Contents) link displays the top level of your Web layout. Although a Web layout is not required and might not be necessary for all applications, it provides an effective means for grouping files and navigating. When a Web layout is not created, the Search function provides the only access to files in the core Oracle WebCenter Content Server. However other products like Site Studio, and extras like Folders and Categorization Folders provide other means of navigation.

Active and Historical reports are other features introduced later in this section. Web viewable files always have lowercase names.

Figure B–1 shows an example of a Web layout using Local Pages, URLs, and Queries as site-building features.

Figure B–1 Web Layout Example



On the Library Web pages, these features are displayed as links with a title next to a file-folder icon. When you click a folder that represents a query, the result produces a set of links to files that match the query's criteria.

B.1.2 Defining the Site Structure and Displaying Criteria

Define the website structure in the Web Hierarchy pane of the Web Layout Editor. Then define criteria to display specific files when the user clicks a folder (or link). The criteria for each link is based on the metadata for each file. Besides executing a query, links can jump to another page of links, go to a URL, or display a report. The following examples demonstrate how links are setup to display files.

- **Example 1:** To enable users to access engineering forms from a link named Forms, create a content type named Forms using the Configuration Manager. Then, create a query with Type equal to Forms using the Web Layout Editor.

  For information about how to create a content type. For details, see "Defining Content Types" in *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

- **Example 2:** To enable users to access specific Standard Work Procedures, create a content type called SWP, and create a query of Type equal to SWP and Content Name substring of 7200.

B.1.3 Task Sequence

The following steps demonstrate the typical sequence of tasks for creating a website with the Web Layout Editor:

1. **Gather information.** The first step is to gather information about how your users would intuitively retrieve information; what do they want and how would they typically search for it? How does this impact security?
2. **Customize metadata.** If necessary, customize your site's metadata by creating any additional fields that might be useful.

   For more information, see "Customizing Repository Fields and Metadata" in *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

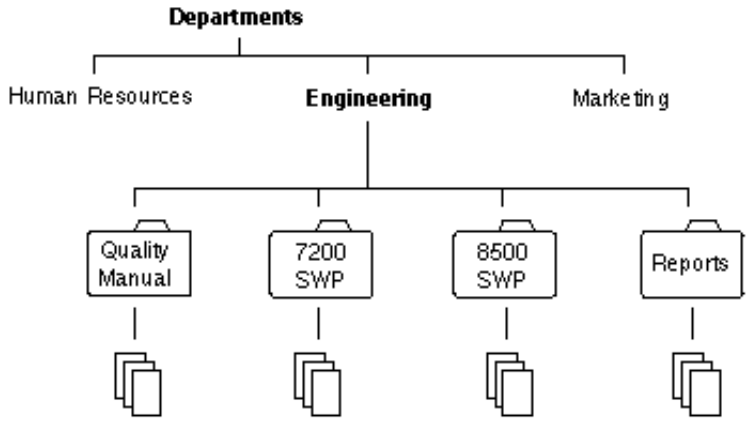
3. **Define content types.** Define the content types to support your site.

   For more information, see "Defining Content Types" in *Oracle Fusion Middleware Managing Oracle WebCenter Content*.

4. **Define security groups, users, and roles.** Create security groups and users, and assign roles to users to establish their permissions (described in the *Oracle WebCenter Content System Administrator's Guide for Content Server*).

5. **Design the website.** Create the website layout. Although a website provides a structure that allows navigation to locate and display files, it is not required. Some companies prefer users to use only the search engine to find files, others prefer to use both a navigation structure and a search engine. To design the website, it is helpful to first draw a website structure as shown in Figure B–2.

Figure B–2 Example Web Structure



B.2 Working with Web Pages

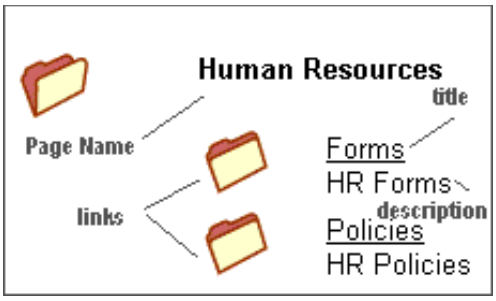
After completing the initial plan, build the website and determine how it functions. Revise it if it does not perform properly or is not manageable. Continue this process until you have a design that is going to work for you and the users.

The links on a page jump to one of these:

- Local page
- URL
- Query
- Report

The links all look similar and can be combined on the same page as shown on the local page in Figure B–3:

Figure B–3 Links in Example Local Page



A *local page* is one of two types: a directory or a report. A local page that is a directory can contain links that open another local page, open a URL, or run a query. Only administrators can create a local page that is a report. Administrators or subadministrators with appropriate rights can create a local page that is a directory.

An *external URL* is a link to a specified URL (Web address). You can link to any URL address or web page on the intranet or Internet.

A *search query* produces a page containing links to files that meet the criteria of a defined query. The page looks the same as a page resulting from a content search.

Important: Queries can inherit the security group, the account or both that the page links are on. If the security group or account is inherited, it automatically restricts the query to files in that security group or account.

Reports are either Active or Historical. Active reports appear as a file folder link and perform a database query each time they are run, generating a display of current information. Like Active Reports, Historical Reports appear as a file folder link, but they contain information that was queried at the time they were initially run. They do not perform a database query each time they are opened, and the report is only changed if it is updated.

B.3 Managing Web Pages

Subadministrators do not have access to the Query Result Pages function in the Options menu nor to any applications for which they do not have rights. Additionally, subadministrators have viewing, editing, and deleting restricted rights as described in these sections:

- Section B.3.1, "Adding a New Web Page"
- Section B.3.2, "Editing Web Page Properties"
- Section B.3.3, "Creating a Local Page Link"
- Section B.3.4, "Creating an External URL Link"
- Section B.3.5, "Editing a Hierarchical Web Page Structure"

B.3.1 Adding a New Web Page

To add a new Web page to the Web layout:

1. In the Web Page Hierarchy Pane, click **Add**.

   The Add Web Page Screen opens.

2. Enter information about the new page.
3. Click **OK**.

B.3.2 Editing Web Page Properties

To edit the properties of a Web page:

1. Select the page in the Web Page Hierarchy Pane.
2. Click **Edit** in the Page Properties Pane.

   The Edit Page Properties Screen opens.

3. Edit the properties.
4. Click **OK**.

B.3.3 Creating a Local Page Link

To create a local page link:

1. Select the page in the Web Page Hierarchy Pane under which you want to locate the new local page.
2. In the Page Links Pane, click **Add**.

   The Add Page Link Screen opens.

3. Select **Local Page**, and click **OK**.
4. Enter information about the new local page into the Edit Local Page Link Screen.
5. When done, click **OK**.

B.3.4 Creating an External URL Link

To create an external URL link:

1. Select the page in the Web Page Hierarchy Pane under which you want to locate the new URL.
2. In the Page Links Pane, click **Add**.

   The Add Page Link Screen opens:

3. Select **External URL**, and click **OK**.
4. Enter information about the URL into the Edit External URL Screen.
5. When done, click **OK**.
6. Refresh the browser to display the new page.

B.3.5 Editing a Hierarchical Web Page Structure

To edit a hierarchical Web page structure, the objective is to insert a page, making it the new parent of the hierarchical page.

For example:

1. Create a structure.
2. Select **QSTest** directory with the Page Link **PCTest** also selected.
3. Click **Page LinksDelete**.
4. Select the **Index** directory and select **Page LinksAdd**.
5. Create a new page.
   - Title the page *NewEngPage*.
   - Type a Description as *NewEngPage*.
6. Click **OK**.

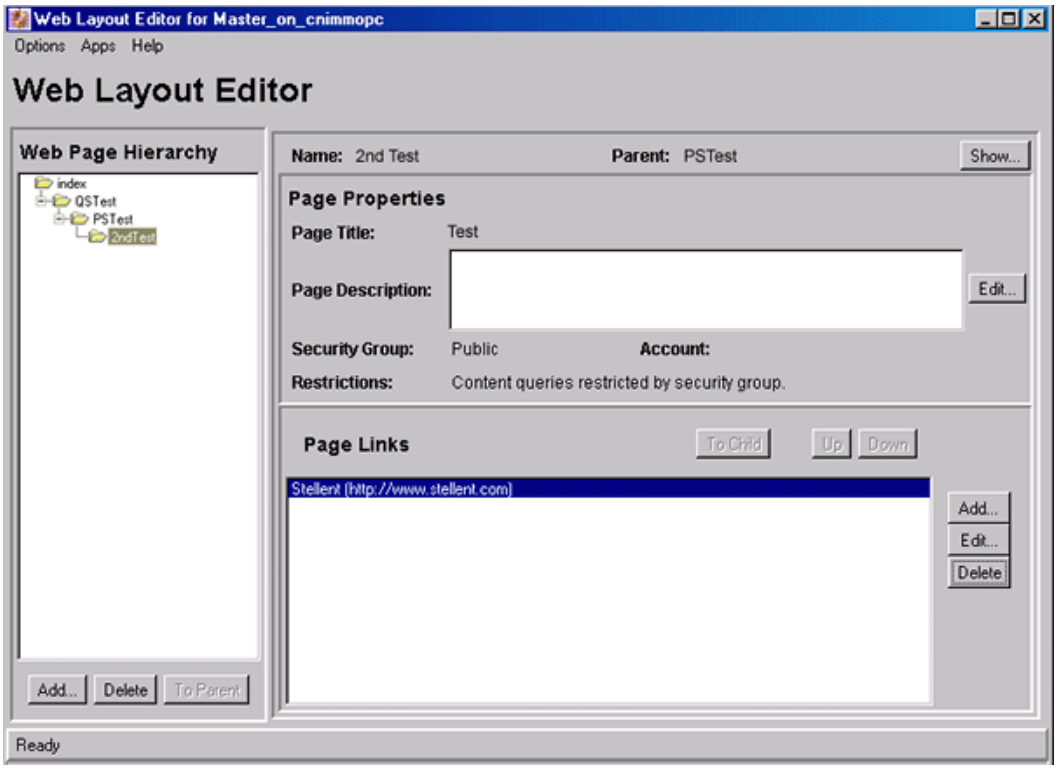
   The NewEngPage opens in the Web Page Hierarchy Pane.
7. Select **QSTest** and click **Page LinksAdd**.
8. Select **Local Page.**
9. Click **OK**.

   The NewEngPage now appears under QSTest.
10. Select NewEngPage and click **Page LinksAdd**.
11. Select **Local PageOK**.

    Note that the Page name is PCTest.

The NewEngPage has now been entered with PCTest as its child and the External URL in 2ndTest has been preserved.

The following image shows a hierarchical Web page structure created by this example procedure.



B.4 Working with Reports

This section covers these topics:

- Section B.4.1, "About Reports"
- Section B.4.2, "Defining an Active Report"
- Section B.4.3, "Defining a Historical Report"
- Section B.4.4, "Editing a Query Expression in an Active Report"

B.4.1 About Reports

You can define two types of reports with the Web Layout Editor: Active Reports and Historical Reports.

- Active Reports appear as a file-folder link and perform a database query each time they are run, generating a display of current information. You can define active reports and edit their query expressions.
- Like Active Reports, Historical Reports appear as a file-folder link, but they contain information that was queried at the time they were initially run. They do not perform a database query each time they are opened, and the report is changed only if the database is updated. The procedure for creating a Historical Report is almost the same as creating an Active Report. The only difference is the Create Historical Report screen has an extra field (Rows Per Page) to specify the number of rows each page of the report can contain.

B.4.2 Defining an Active Report

To define an active report:

1. In the Web Layout Editor, add a new Web page, and select **Active Report** as the Page Type.
2. In the Active Report Specification pane of the Web Layout Editor, click **Edit Report Query**.
3. Define the query by entering information on the Edit Active Report Query Screen.
4. Click **OK**.

B.4.3 Defining a Historical Report

To prepare an Archive Historical Report:

1. In the WebLayout Editor, add a new Web page and select **Historical Report** as the Page Type.
2. In the Historical Report Specification pane of the Web Layout Editor, click **Create Report Data**.
3. When you create the report data in the Create Historical Report Screen, specify **Archive History** for the data source.
4. Write a query for the report that returns the data you want to retrieve. For example, specify the Content ID.
5. When done specifying information for the report, click **OK**.

B.4.4 Editing a Query Expression in an Active Report

To edit the query expression in an active report:

1. In the Web Page Hierarchy Pane, select the report you want to edit.
2. In the Active Report Specification pane, click **Edit Report Query**.
3. In the Query Expression window on the Edit Active Report Query Screen, select the query line to edit.
4. Make changes to the query as necessary, and click **Update**.

> **Caution:** If you clear the Custom Query Expression check box, the expression reverts to its original definition; all modifications are lost.

5. Click **OK**. If a query is not specified, all values are returned.

B.5 Writing Queries

This section covers these topics:

- Creating a Query Link
- Editing the Query Expression in a Query Link
- Adding a Query Results Page
- Editing a Query Results Page
- Deleting a Query Results Page

You can write custom query expressions when you define query links. The method you use to write custom queries varies depending on the kind of query you write.

To write directory custom queries, use Idoc Script, a proprietary scripting language. To write report queries, you can use SQL script and Idoc Script. Idoc Script is described in detail in the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*. Basic SQL script is briefly described below.

> **Note:** Your SQL syntax is dependent on your database. Different databases expect different syntax for items like wildcards, and so forth. See your database documentation for specific syntax information.

SQL script involves operators, which are words that show logical relationships between the words in your query. The following table contains some basic operators and their use.

Operator	Use
AND	Returns files that contain the specified words.
OR	Returns files that contain at least one specified word.
=	Equal
<> or !	Not equal
<	Less than
>	Greater than
<+	Less than or equal to
>+	Greater than or equal to
IN	Finds a position in a table.
BETWEEN	Finds a value in a range.
NOT	Excludes the files that contain the specified condition.

The following are examples of SQL script:

- Finds all files that have an internal revision ID less than 50000:

```
dID < '50000'
```

- Finds all files that have a Content ID between 10000 and 50000:

```
dDocName BETWEEN '10000' AND '50000'
```

B.5.1 Creating a Query Link

To create a query link:

1. In the Web Page Hierarchy Pane, select the page where you want to locate the new query link.
2. In the Page Links pane, click **Add**.

   The Add Page Link Screen opens.

3. Select **Query**, and click **OK**.

   The Query Link Definition Screen opens.

4. Enter information into the screen.

5. When done, click **OK**.

When adding Idoc Script variables and HTML tags to the Text 1 and Text 2 fields, keep in mind that any resulting HTML tags can affect the display of the search results page. See the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content* for more information.

B.5.2 Editing the Query Expression in a Query Link

To edit the query expression in a query link:

1. In the Page Links Pane of the Web Layout Editor, select the query you want to edit.

2. Click **Edit**.

   The Query Link Definition Screen opens.

3. In the Query Expression area, select the query line to edit.

4. Make changes to the metadata, Operator, metadata Value fields as necessary, then click **Update**.

5. Click **OK**.

> **Caution:** If you clear the Custom Query Expression check box, the expression reverts to its original definition; all modifications are lost.

B.5.3 Adding a Query Results Page

Follow these instructions to add a query results page. This task is available for administrators, not for subadministrators with WebLayout rights:

1. From the Web Layout Editor menu, select **Options**, then select **Query Results Pages**.

   The Query Result Pages screen opens.

2. Click **Add** to display the Add Result Page.

3. Enter information for the new page. Observe the field descriptions for Add/Edit Query Results Page.

4. Click **OK**.

B.5.4 Editing a Query Results Page

Follow these instructions to edit a query results page. This task is available for administrators, not for subadministrators with WebLayout rights:

1. From the Web Layout Editor menu, select **Options**, then **Query Results Pages**, then select the name of the page.

2. Click **Edit**.

   The Add/Edit Query Results Page opens.

3. Make the necessary changes, and click **OK**.

B.5.5 Deleting a Query Results Page

Follow these instructions to delete a query results page. This task is available for administrators, not for subadministrators with WebLayout rights:

1. From the Web Layout Editor menu, select **Options**, then **Query Results Pages**, then select the name of the page.
2. Click **Delete**.
3. When prompted, click **OK** to verify the deletion.

C

Troubleshooting

This appendix describes how to use troubleshooting aids to resolve problems with customizing Oracle WebCenter Content Server.

This appendix includes the following sections:

C.1 About Troubleshooting Aids

Several troubleshooting aids are available to help evaluate Content Server pages as they are used.

C.2 Viewing Server Errors

Syntax errors and other mistakes in component files or dynamic server pages can cause errors in Content Server. If the Content Server instance fails, it reports the error in the following locations:

- If you run Content Server from a command prompt, you can view the error in the console window.
- If you can log in to Content Server and display the Admin Server page, you can view the Content Server log by selecting the Content Server instance and then clicking **View Server Output**.
- You can view the Content Server log files in the *DomainHome*/ucm/cs/weblayout/groups/secure/logs directory.

C.3 Viewing Page Data

The `IsJava` setting displays the local data of a Content Server web page.

- In a web browser, add the following code in the **Address** box to the end of the page's URL:

  ```
  &IsJava=1
  ```

- On a template page or in an include, use the following code:

  ```
  <$IsJava=1$>
  ```

The `IsPageDebug` setting displays a tree structure view of all includes being called on a Content Server web page. The debug trace appears at the bottom of the web page.

- In a web browser, add the following code in the **Address** box to the end of the page's URL:

  ```
  &IsPageDebug=1
  ```

- On a template page or in an include, use the following code:

  ```
  <$IsPageDebug=1$>
  ```

  Tip: You can also set the `IsPageDebug` variable in the config.cfg file if you want the setting to apply for the whole server.

- To place a marker in the script debug trace, place the following code at the point where you want to see a value or perform a step:

  ```
  <$trace("marker code")$>
  ```

  For example, you can use the following code to insert the current user name in the debug trace (the `eval` function must be used to evaluate Idoc Script):

  ```
  <$trace(eval("The user name is <$UserName$>"))$>
  ```

For more information about `IsJava` and `IsPageDebug`, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

C.4 Monitoring Resource Loading

Three configuration settings enable you to view the loading of resources when you run Content Server from a command line. Set any of these variables equal to 1 in the *IntradocDir*/config/config.cfg file:

- **TraceResourceLoad** logs all resources loaded, resource overrides, resource conflicts, and resource merges.
- **TraceResourceOverride** logs when a system resource is overridden by a component resource or a component resource is loaded twice.
- **TraceResourceConflict** logs when a system resource is overridden twice by component resources.

For more information about these configuration setting, see the *Oracle Fusion Middleware Configuration Reference for Oracle WebCenter Content*.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z