Oracle® Fusion Middleware

Developer's Guide for Oracle Universal Content Management

11*g* Release 1 (11.1.1)

E10807-03

April 2011

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle Universal Content Management, 11*g* Release 1 (11.1.1)

E10807-03

Primary Author: Bonnie Vaughan

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

Contributors: Sharmarke Aden, Daniel Lew, Scott Nelson, Rick Petty, David Truckenmiller, Ron van de Crommert, Peter Walters, Sam White

Contents

3 Working with Standard, Server, and Custom Components

4 Changing the Look and Navigation of the Oracle Content Server Interface

7 Using the IdcCommand Utility to Access Services

8 Using the COM API for Integration

9 Using Remote Intradoc Client (RIDC)

10 Using Content Integration Suite (CIS)

11 Using the Java Content Repository Adapter

12 Using Oracle UCM Web Services

13 Customizing DesktopTag

Index

Preface

While Oracle Universal Content Management (Oracle UCM) is highly functional "out of the box," there are many ways to tailor it to your site requirements. This guide provides the background information necessary to customize your Oracle UCM instance.

Audience

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

Document Organization

This guide includes the following sections:

- Chapter 1, "Introduction to Customizing Your Oracle UCM Instance," provides an introduction to the methods and tools you can use to customize Oracle UCM.
- Chapter 2, "Oracle UCM Architecture," describes the architecture of Oracle UCM and how that affects the customization you can make.
- Chapter 3, "Working with Standard, Server, and Custom Components," describes how to use components to modify or add functionality to Oracle Content Server.
- Chapter 4, "Changing the Look and Navigation of the Oracle Content Server Interface," defines the items you can adjust to change the look and navigation of the Oracle Content Server interface.
- Chapter 5, "Modifying System Functionality," describes how you can change the functionality of Oracle UCM settings, components, and configuration variables.
- Chapter 6, "Integrating Oracle UCM with Enterprise Applications," provides information about integrating Oracle Content Server with enterprise applications such as application servers, catalog solutions, and enterprise portals.
- Chapter 7, "Using the IdcCommand Utility to Access Services," provides information about using the IdcCommand utility to access Oracle Content Server services from other applications.
- Chapter 8, "Using the COM API for Integration," provides information about how Oracle Content Server utilizes a Component Object Model-based API, which provides the capability to call functionality from within a Microsoft Component Object Model (COM) environment.

- Chapter 9, "Using Remote Intradoc Client (RIDC)," describes Remote Intradoc Client (RIDC) and how you can use the RIDC API for communication with Oracle Content Server.
- Chapter 10, "Using Content Integration Suite (CIS)," describes the Content Integration Suite (CIS) API, which you can use to access Oracle Content Server services and data from a unified object model.
- Chapter 11, "Using the Java Content Repository Adapter," provides information about the Java Content Repository API, which was developed under the Java Community Process as JSR-170 and includes the Content Repository for Java API and the Java Content Repository (JCR).
- Chapter 12, "Using Oracle UCM Web Services," discusses using web services and SOAP (Simple Object Access Protocol) to manage Oracle Content Server.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Universal Content Management 11*g* Release 1 (11.1.1) documentation set:

- *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*
- *Oracle Fusion Middleware Application Administrator's Guide for Content Server*
- *Oracle Fusion Middleware Idoc Script Reference Guide*
- *Oracle Fusion Middleware Services Reference Guide for Oracle Universal Content Management*

Conventions

The following text conventions are used in this document.

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 Universal Content Management (Oracle UCM) for Oracle Content Server developers that are covered in this guide.

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

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

- 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 Developer's Guide for Content Integration Suite*
  - *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: Oracle UCM uses Oracle WebLogic Server web services. For more information, see Chapter 12, "Using Oracle UCM Web Services."
- **ComponentTool:** The ComponentTool utility has been added to provide a command-line tool for installing, enabling, and disabling components. For more information, see Chapter 3, "Working with Standard, Server, and Custom Components."
- Oracle Content Server deployment: Oracle Content Server is deployed with Oracle UCM to an Oracle WebLogic Server domain, which means changes in configuring and administering Oracle UCM. For more information, see *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*.

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

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

- Oracle UCM Directories and Files: Oracle UCM 11*g* Release 1 (11.1.1) is provided as part of a full media install of Oracle Enterprise Content Management Suite, with the Oracle UCM directories and files. The directory structure for an Oracle UCM 11*g* instance is different from an Oracle UCM 10*g* instance. The following terms and path names are important to understanding and working with the Oracle UCM structure:
  - *IdcHomeDir*: This variable refers to the ucm/idc directory in the ECM Oracle home where the Oracle UCM server media is located. The server media can run Oracle Content Server, Oracle Inbound Refinery, or Oracle Universal Records Management. This is essentially a read-only directory. The default location is *ECM_ORACLE_HOME*/ucm/idc. The variable portion of the default location can be changed, but the ucm/idc portion is required.
  - *DomainHome*: This variable refers to the user-specified directory where an Oracle UCM server is deployed to run on an Oracle WebLogic Server 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 Oracle UCM to Oracle WebLogic Server.
  - *short-product-id*: This variable refers to the type of Oracle UCM server deployed on an Oracle WebLogic Server. Possible values follow:
    * `cs` (Oracle Content Server)
    * `ibr` (Oracle Inbound Refinery)
    * `urm` (Oracle Universal Records Management)
  - *IntradocDir:* This variable refers to the root directory for configuration and data files specific to an Oracle Content Server instance that is part of an Oracle UCM application deployed to an Oracle WebLogic Server domain. This Idoc Script variable is configured for one type of Oracle Content Server instance: Oracle Content Server (`cs`), Oracle Inbound Refinery (`ibr`), or Oracle Universal Records Management (`urm`). This directory can be located elsewhere, but the default location is *DomainHome*/ucm/*short-product-id*/. The specified directory must be an absolute path to the instance directory, and it must be unique to a particular server or node. The directory files include startup files (intradoc.cfg and executables).
- SOAP: SOAP is provided with Oracle WebLogic Server, not in Oracle UCM.
- Web Form Editor: The Web Form Editor user interface and FCKEditor are not supported.

1

Introduction to Customizing Your Oracle UCM Instance

This chapter provides an overview of Oracle Universal Content Management (Oracle UCM) customization and describes the tools you need and the resources that are available.

This chapter includes the following sections:

1.1 Customization Types

Three major types of alterations can be made to the core Oracle Content Server instance:

- **Altering the look and feel of the product:** You can customize the look and feel of the Oracle Content Server interface to meet your organization's specifications. Interface modifications can be as simple as replacing the icons that appear on the standard Oracle 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 Oracle 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 Oracle Content Server into your site's current environment.

1.2 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 Oracle 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 Oracle 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**, taking into account those aspects of the Oracle 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 web site contain many types of customization. It's possible that there may be an existing component that can be used 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 your Oracle UCM products.

4. **Evaluate the problem and how essential it is to solve.** Some problems may require more effort to fix than will provide 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 provides an historical audit trail for others who may need to add to the customization later.

1.3 Recommended Skills and Tools

Oracle UCM brings together a wide variety of technologies to deliver advanced functionality. To modify the system, certain experience and skills with some or all of these technologies is required.

The technical skills required to customize your content management system 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 may need to modify Oracle Content Server:

- Oracle Content Server Architecture

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

- HTML/CSS

  You will need a good understanding of HTML and cascading style sheets (CSS) to make changes to the Oracle 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 layout.

- Idoc Script

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

- JavaScript

  The internal content of most Oracle 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 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.

- SQL

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

- Java programming

  Oracle Content Server is implemented with Java classes. You should have a thorough understanding of Java and the Oracle 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++, VBScript, and so forth may be helpful if you are doing complex customization or integrating Oracle UCM with other systems.

You may find the following tools useful when customizing Oracle 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 may cause Idoc Script tags to be converted into a string of characters that are no longer recognized by Oracle 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, Netscape, Mozilla, and Safari 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.

1.4 Troubleshooting

Several troubleshooting aids are available to help evaluate Oracle Content Server pages as they are used. The following sections discuss three broad types of troubleshooting aids:

- Section 1.4.1, "Viewing Server Errors"
- Section 1.4.2, "Viewing Page Data"
- Section 1.4.3, "Monitoring Resource Loading"

1.4.1 Viewing Server Errors

Syntax errors and other mistakes in component files or dynamic server pages can cause errors in Oracle Content Server. If the Oracle Content Server instance fails, it reports the error in the following locations:

- If you run Oracle Content Server from a command prompt, you can view the error in the console window.
- If you can log in to Oracle Content Server and display the Admin Server page, you can view the Oracle Content Server log by selecting the Oracle Content Server instance and then clicking **View Server Output**.
- You can view the Oracle Content Server log files in the *DomainHome*/ucm/cs/weblayout/groups/secure/logs directory.

1.4.2 Viewing Page Data

The `IsJava` setting displays the local data of an Oracle 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 an Oracle 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$>")$>
  ```

`IsJava` and `IsPageDebug` are discussed in detail in the *Oracle Fusion Middleware Idoc Script Reference Guide*.

1.4.3 Monitoring Resource Loading

Three configuration settings enable you to view the loading of resources when you run Oracle 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.

These configuration settings are discussed in detail in the *Oracle Fusion Middleware Idoc Script Reference Guide*.

The following example shows the command line output when TraceResourceLoad=1.

```
Loading Java Resources
Loading ConflictTester Component
Loading ConflictTester2 Component
Loading Compression Component
Merging  into Filters
*MERGE* [validateStandard, compression.ConversionParamsFilter, null, 1]
Loading Html Resources
Loading System Resource
c:/intradoc/shared/config/resources/upper_clmns_map.htm
ColumnTranslation
Loading System Resource
```

```
c:/intradoc/shared/config/resources/indexer.htm
IndexerQueryTable
IndexerStatesTable
IndexerTransitionsTable
DefaultIndexerCycles
Loading System Resource
c:/intradoc/shared/config/resources/std_page.idoc
std_html_head_declarations
std_definitions
std_html_head_definition_declarations
std_page_variable_definitions
…

Loading System Resource
c:/intradoc/shared/config/resources/std_docrefinery.htm
AdditionalRenditionsSource
DocumentConversions
ConversionSteps
Loading ConflictTester Component
c:/intradoc/custom/ConflictTester/resources/conflicttester_resource.htm
conflict_tester_include
ConflictTester_Table
Loading ConflictTester2 Component
c:/intradoc/custom/ConflictTester2/resources/conflicttester_resource.htm
*OVERRIDE* conflict_tester_include
***CONFLICT*** ConflictTester_Table
Loading Compression Component
c:/intradoc/custom/Compression/Compression_resource.htm
*OVERRIDE* searchapi_result_definitions
*OVERRIDE* searchapi_thumbnail_result_doc_href_start
*OVERRIDE* searchapi_result_table_content_begin
compression_thumbnail_img
Loading Compression Component
c:/intradoc/custom/Compression/Compression_handlers.htm
CompressionHandlers
Merging ConflictTester_Templates into IntradocTemplates
*MERGE* HOME_PAGE
Merging ConflictTester_Templates into IntradocTemplates
*MERGE* HOME_PAGE
Merging CompressionIntradocTemplates into IntradocTemplates
*MERGE* COMPRESSION_IMAGE_INFO
Merging CompressionHandlers into ServiceHandlers
*MERGE* [FileService, compression.CompressionFileServiceHandler, 100]
*MERGE* [FileService, DocCommonHandler, 100]
*MERGE* [DocService, compression.CompressionFileServiceHandler, 100]
…
```

2

Oracle UCM Architecture

This chapter describes the architecture of Oracle Universal Content Management (Oracle UCM) in the context of what you need to know before beginning a customization project. To create a customization efficiently and effectively, you should have an understanding of how Oracle UCM works.

This chapter includes the following sections:

2.1 Oracle UCM Directories and Files

When you create custom components or dynamic server pages, you work primarily with files in the bin, config, components, resources, and weblayout directories. The following sections describe these directories:

> **Caution:** Modifying the default variables in these files can cause Oracle UCM to malfunction. For more information about configuration variables, see the *Oracle Fusion Middleware Idoc Script Reference Guide*.

2.1.1 Terminology for Oracle UCM Directories

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

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

2.1.2 The bin Directory

The bin directory is the root directory for Oracle Content Server startup files. It contains the intradoc.cfg file and the executable files that run Oracle 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 Oracle Content Server, Oracle Inbound Refinery, or Oracle Universal Records Management.

Element	Description
Executables	**Services** ■ IdcServer ■ IdcServerNT **Applets** ■ IntradocApp (launches all Admin tools) **Utilities** ■ Batch Loader ■ Installer ■ IdcAnalyze ■ Component Wizard ■ System Properties ■ IdcCommand
intradoc.cfg file	Configuration file that contains the settings for Oracle Content Server services, applets, and utilities.

Note: If Oracle Content Server is set up as an automatic service and you attempt to start an Oracle 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 Oracle Content Server, including directory, Internet, and refinery settings. Several of these variables can be set using the Oracle Content Server System Properties utility.

A typical intradoc.cfg file follows:

```
<?cfg jcharset="Cp1252"?>
#Oracle 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
```

2.1.3 The config Directory

The config directory stores global configuration information for Oracle Content Server. This directory can be located elsewhere, but the default location is *DomainHome*/ucm/*short-product-id*/config.

Element	Description
config.cfg file	Defines system configuration variables.

The config.cfg file is used to define global variables for the Oracle Content Server system. Several of these variables can be set using the Oracle Content Server System Properties utility or by modifying the variables on the Admin Server General Configuration page.

A typical config.cfg file follows:

```
<?cfg jcharset="Cp1252"?>
#Oracle 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
```

2.1.4 The components Directory

The *IntradocDir*/data/components directory contains the files that Oracle Content Server uses to configure system components.

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

The following example file is a *component*.hda file that defines the configuration status for a component called *help*.

```
<?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
```

2.1.5 The resources Directory

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

The resources/core directory contains files that Oracle Content Server uses to assemble web pages.

Element	Description
config	Holds base configuration files for Oracle Content Server.
idoc	Holds IdocScript `dynamichtml` and `dynamicdata` definitions.
install	Holds files used by the installer and related applications.
javascript	Holds files which 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 Oracle Content Server.
reports	Holds templates for Oracle Content Server reports.
resources	Holds resource definitions (queries, page resources, services, and other resource data) for Oracle Content Server.
tables	Holds IdocScript resource table definitions.
templates	Holds templates for all Oracle Content Server pages (except reports).

The resources/admin directory contains the following resource types.

Element	Description
idoc	Holds IdocScript `dynamichtml` and `dynamicdata` definitions.
tables	Holds IdocScript resource table definitions.
templates	Holds templates for all Oracle Content Server pages (except reports).

2.1.6 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 Oracle Content Server web site.

Element	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.

2.2 Resources

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

Resources are a critical part of the Oracle 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 distinct categories, although the first five types listed in the following table are also called *Resource*-type resources.

Resource Type	Description	Example of Standard Resource
HTML Include	Defines pieces of HTML markup and Idoc Script code that are used on multiple Oracle 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 IdocScript 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 Oracle Content Server.	*IdcHomeDir*/resources/core/datastoredesign/columnIndexdList.hda

Resource Type	Description	Example of Standard Resource
Static Table (HTML format)	Provides static (seldom changed) content in table format to Oracle Content Server.	*IdcHomeDir*/resources/core/std_locale.htm
Query	Defines database queries.	*IdcHomeDir*/resources/core/tables/query.htm
Service	Defines scripts for services that can be performed by Oracle Content Server.	*IdcHomeDir*/resources/core/tables/std_services.htm
Template	Defines templates, which contain the code that Oracle Content Server uses to assemble a particular web page.	*IdcHomeDir*/resources/core/templates/checkin_new.htm
Environment	Defines configuration settings for Oracle Content Server.	*IntradocDir*/config/config.cfg

2.3 Oracle Content Server Behavior

The following sections describe how Oracle Content Server behaves in different situations:

- Section 2.3.1, "Startup Behavior"
- Section 2.3.2, "Resource Caching"
- Section 2.3.3, "Oracle Content Server Requests"
- Section 2.3.4, "Page Assembly"
- Section 2.3.5, "Database Interaction"
- Section 2.3.6, "Localized String Resolution"

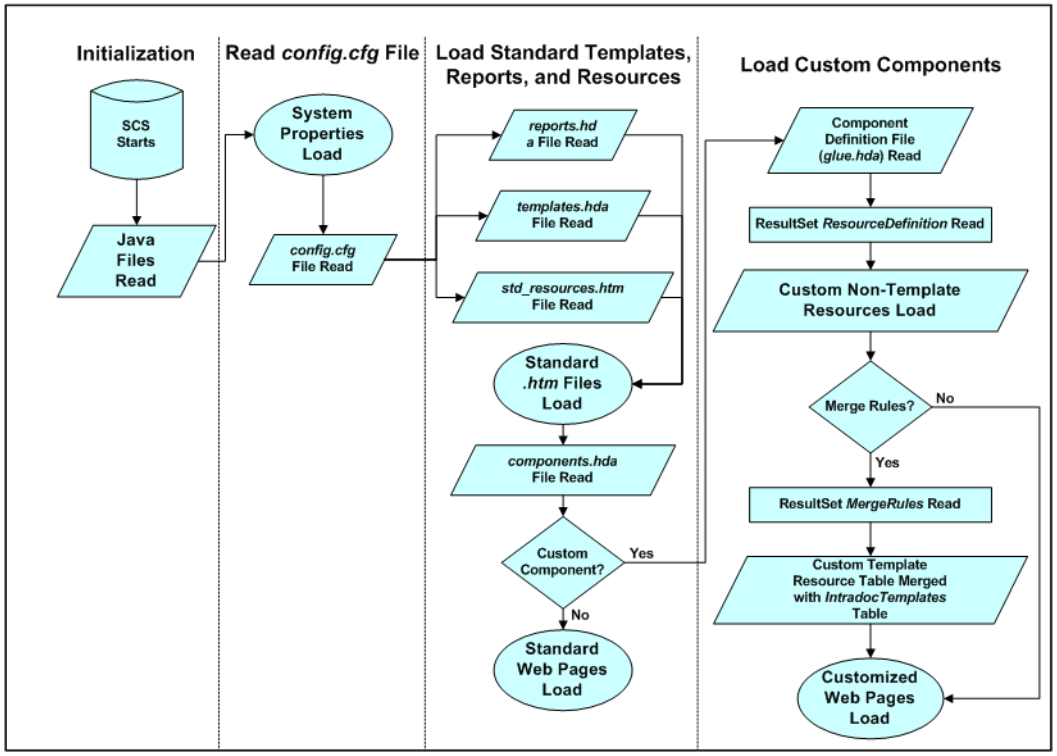
2.3.1 Startup Behavior

The following steps occur during Oracle Content Server startup:

1. Internal initialization occurs.
2. Configuration variables load.
3. Standard templates, resources, and reports load.
4. Custom components load (templates, resources, configuration variables, and reports).

Figure 2–1 illustrates these steps in a flowchart. Section 2.3.1.1, "Startup Steps," describes each step in more detail.

Figure 2–1 Oracle Content Server Startup Behavior



2.3.1.1 Startup Steps

Here are descriptions of the steps that an Oracle Content Server instance goes through during startup:

1. **Internal Initialization Occurs:** When Oracle Content Server initializes internally, the Java class files from Oracle Content Server are read and the Java Virtual Machine (JVM) is invoked. Any variables in the *DomainHome*/ucm/*short-product-id*/intradoc.cfg file initialize as well.
2. **Configuration Variables Load:** After initializing, Oracle 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 Content Server installation process, but you can modify this file in several ways:

   - By using the Admin Server General Configuration page, accessible from the **Administration** menu.
   - By using the **System Properties** option, which is available from the **Start** menu (in Windows) or by running the SystemProperties script, located in the bin directory of your installation (on a UNIX operating system).
   - By editing the configuration files directly.
   - By using a custom component.

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

3. **Standard Resources, Templates, and Reports Load:** For Oracle Content Server to function properly, many standard resources, templates, and reports must be loaded. After the configuration settings have been loaded, Oracle 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 Oracle Content Server which templates to load, which in turn loads any standard resources referenced in the template and report pages.
4. **Custom Components Load:** Oracle Content Server loads resources in the order specified in the *IntradocDir*/custom/components.hda file.

2.3.1.2 Effects of Configuration Loading

It is important to understand the effect of the load order of the different configuration files because if a variable is set in more than file, the last variable loaded takes precedence. For example, the *IntradocDir*/config/config.cfg file is loaded first, and the *IntradocDir*/data/components/*component_name*/config.cfg file is loaded last, 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. *IntradocDir*/config/config.cfg.
2. *DomainHome*/ucm/*short-product-id*/custom/*component_name*/*_environment.cfg. Some components may not have this file or it may be named environment.cfg.
3. *IntradocDir*/data/components/*component_name*/install.cfg.
4. *IntradocDir*/data/components/*component_name*/config.cfg.
5. *DomainHome*/ucm/*short-product-id*/bin/intradoc.cfg is reread at the end of startup to allow overrides to other settings.

If, for example, a variable was set in each of the files listed previously, the variable in *component_name*/config.cfg takes precedence.

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

To change a component variable, use the Advanced Component Manager and select **Update Component Configuration**. This displays values in the *component_name*/config.cfg file that are editable. Make the desired changes, and click **Update**.

You can also edit the configuration file manually. If a component is not displayed in the drop-down list or if the variables displayed do not include the one to change, make the change directly in one of the configuration files.

2.3.2 Resource Caching

Oracle Content Server handles caches template pages and resources as follows:

- When Oracle 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 config variable DisableSharedCacheChecking.

- If you change any other component files (including services, queries, environment variables, tables, components.hda file, and template.hda file), you must restart Oracle Content Server before the changes go into effect. This is because such changes could cause Oracle Content Server to malfunction if they were implemented immediately. You do not need to restart Oracle Content Server when changing strings; however, you must republish the ww_strings.js files by clicking **publish dynamic files** in the Weblayout Publishing area of the Admin Actions page. For more information, see Chapter 3, "Working with Standard, Server, and Custom Components."

2.3.3 Oracle Content Server Requests

When a web browser client sends an Oracle Content Server request to the web server, the instructions are typically communicated through URLs or form fields. The web server routes the request to Oracle Content Server, which then performs one or more of the following actions:

- Retrieves pages

  For more information, see Section 2.3.3.1, "Page Retrieval."

- Runs an Oracle Content Server service

  For more information, see Section 2.3.3.2, "Oracle Content Server Services."

- Runs a search engine service

  For more information, see Section 2.3.3.3, "Search Services."

> **Note:** Oracle Content Server uses the web server provided by Oracle WebLogic Server.

When an Oracle Content Server web page is requested, all of the necessary information can be sent to Oracle Content Server through the URL. A typical Oracle Content Server URL follows, the URL for the Home page:

```
http://cs.example.com/instancename/idcplg?IdcService=GET_DOC_
PAGE&Action=GetTemplatePage&Page=HOME_PAGE
```

- `http://cs.example.com/`*`instancename`* is the web address of the Oracle Content Server instance.
- `IdcService=GET_DOC_PAGE` tells Oracle Content Server to execute the GET_DOC_PAGE service.
- `Action=GetTemplatePage` tells Oracle Content Server to return the results using a specified template page.
- `Page=HOME_PAGE` tells Oracle Content Server which template page to use.
- The question mark (**?**) indicates the end of the web server path and the beginning of Oracle Content Server instructions.
- Ampersands (**&**) are used as separators between Oracle Content Server instructions.

- You can include some Idoc Script variables in a URL to affect page display at the time of the page request. This is useful for troubleshooting or for temporary pages. For example, the following variables can be used for customization:
  - &StdPageWidth=1000
  - &dDocAuthor:isHidden
  - &dDocType=HRForm

Necessary information can also be sent to Oracle Content Server through form fields on the page. A typical Oracle Content Server form follows:

```
<form name=SubscriptionForm action="<$HttpCgiPath$>" Method="GET"">
<input type=hidden name=dID value="<$dID$>">
<input type=hidden name=dDocName value="<$dDocName$>">
<input type=hidden name=subscribeService value=SUBSCRIBE>
<input type=hidden name=exitUrl value="<$HttpCgiPath$>?IdcService=DOC_
INFO&dID=<$dID$>&dDocName=<$dDocName$>">
<input type=hidden name=title value="Subscriptions">
<input type=hidden name=unsubscribeService value=UNSUBSCRIBE>
<$if ClientControlled$>
<input type=hidden name=ClientControlled value="<$ClientControlled$>">
<$endif$>
<$if DocHasSubscription$>
<input type=hidden name=IdcService value="UNSUBSCRIBE_FORM">
<input type=submit value="<$lc("wwUnsubscribe")$>">
<$else$>
<input type=hidden name=IdcService value="SUBSCRIBE_FORM">
<input type=submit value="<$lc("wwSubscribe")$>">
<$endif$>
</form>
```

2.3.3.1 Page Retrieval

When a web page is requested from Oracle Content Server, the page it returns is either static dynamic:

- Static page

  The content of a static web page is preformatted, and does not change from one request to the next. On some Oracle Content Server web sites, the only static page is the Guest Home page (*DomainHome*/ucm/*short-product-id*/weblayout/portal.htm).

- Dynamic page

  A dynamic web page is assembled at the time of the web server request, using Oracle Content Server services and templates to determine the content and formatting. For example, each user's portal design page is generated using an Oracle Content Server service called GET_PORTAL_PAGE and a template called PNE_PORTAL_DESIGN_PAGE.

2.3.3.2 Oracle Content Server Services

When a web browser requests a dynamic page from Oracle Content Server, the browser is actually placing a request for an Oracle Content Server *service*.

For example:

1. When a user clicks the **Administration** link in the navigation area, a request for the GET_ADMIN_PAGE service is sent to the web server.
2. The web server recognizes this request as an Oracle Content Server function, and sends the specific request to Oracle Content Server.
3. When Oracle Content Server has processed the request, it passes the result back to the web server. In the case of the **Administration** link, the GET_ADMIN_PAGE service:
   - Provides a login prompt if the user is not currently logged in.
   - Verifies that the user has the `admin` privilege.
   - Assembles the Administration page, using the ADMIN_LINKS template.
   - Returns the assembled web page to the web server.
4. The web server delivers the results of the Oracle Content Server service to the originating web browser client.

2.3.3.3 Search Services

A search request is a special kind of Oracle Content Server service. When Oracle Content Server receives a search request, it sends the request on to the search engine, using a search engine API. This service makes it possible to use different search engines with Oracle Content Server.

For example:

1. When a user clicks the **Search** button on the standard Search page, a request for the GET_SEARCH_RESULTS service is sent to the web server.
2. The web server recognizes the request as an Oracle Content Server function and sends the specific request to Oracle Content Server.
3. Oracle Content Server passes the request to the search engine that is configured for the Oracle Content Server instance.
4. The search engine returns the search results to Oracle Content Server.
5. Based on the user login and security permissions, Oracle Content Server assembles the search results page and returns it to the web server.
6. The web server delivers the results to the originating web browser client.

2.3.4 Page Assembly

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

- The structure and format of a web page is defined by a particular HTML template file in either the resources/admin/templates directory or the resources/core/reports directory.
- The templates reference *resources*, which 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:

- A standard page header
- A standard page beginning
- A standard page ending

Because all of the Oracle Content Server resources are cached in memory at startup, Oracle Content Server has a definition for the standard pieces that appear on the page. Oracle 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 Oracle Content Server. When one of these pages is requested by a web browser, Oracle 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 Oracle Content Server.

2.3.5 Database Interaction

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

Because of this case mapping issue, custom components created for an Oracle Content Server instance using one database might not work correctly on an Oracle 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 Oracle Content Server database fields (for example, if you create a component that accesses a custom table within the Oracle Content Server database).

For information about using the upper_clmns_map.htm file, see Section 5, "Modifying System Functionality."

2.3.6 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. Oracle 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.

3

Working with Standard, Server, and Custom Components

This chapter describes how to work with Oracle Universal Content Management (Oracle UCM) components, which are programs used to modify Oracle Content Server functionality.

This chapter includes the following sections:

3.1 Components Overview

Components are modular programs designed to interact with Oracle Content Server at runtime. **Standard components**, **system components**, and **custom components** are included with Oracle Content Server to add or change the core functionality of the standard Oracle Content Server instance. You can create and use **custom components** to modify an Oracle Content Server instance without compromising the system integrity. Custom components can alter defaults for your system, add new functionality, or streamline repetitive functions.

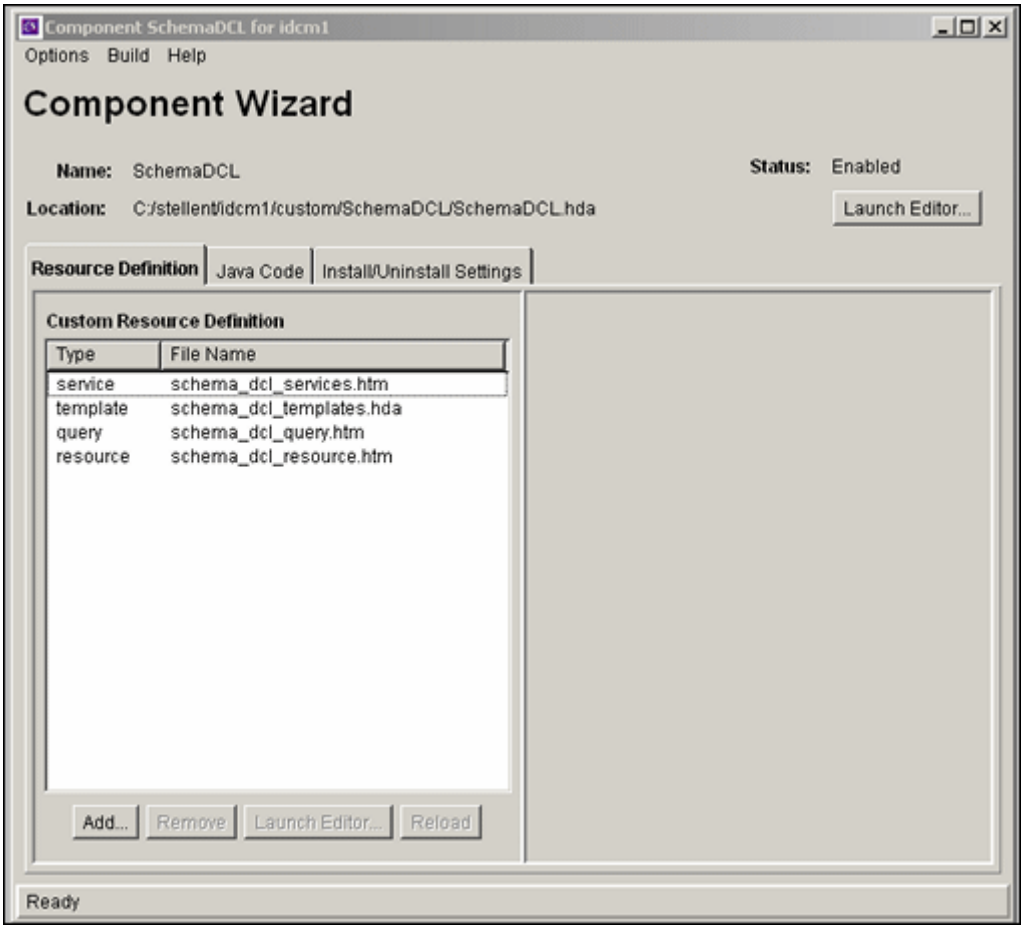
This section provides an overview of component management and the files and directory structure associated with components. It covers these topics:

3.1.1 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 Oracle Content Server instances.

The Component Wizard is discussed in more detail in the *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*.

Figure 3–1 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.

3.1.2 Advanced Component Manager

The Advanced Component Manager provides a way to manage custom components in Oracle Content Server. By using the Advanced Component Manager, you can easily enable or disable components or add new components to Oracle Content Server.

The Advanced Component Manager is discussed in more detail in the *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*.

To use the Advanced Component Manager:

1. In the **Administration** tray or menu, choose **Admin Server**.

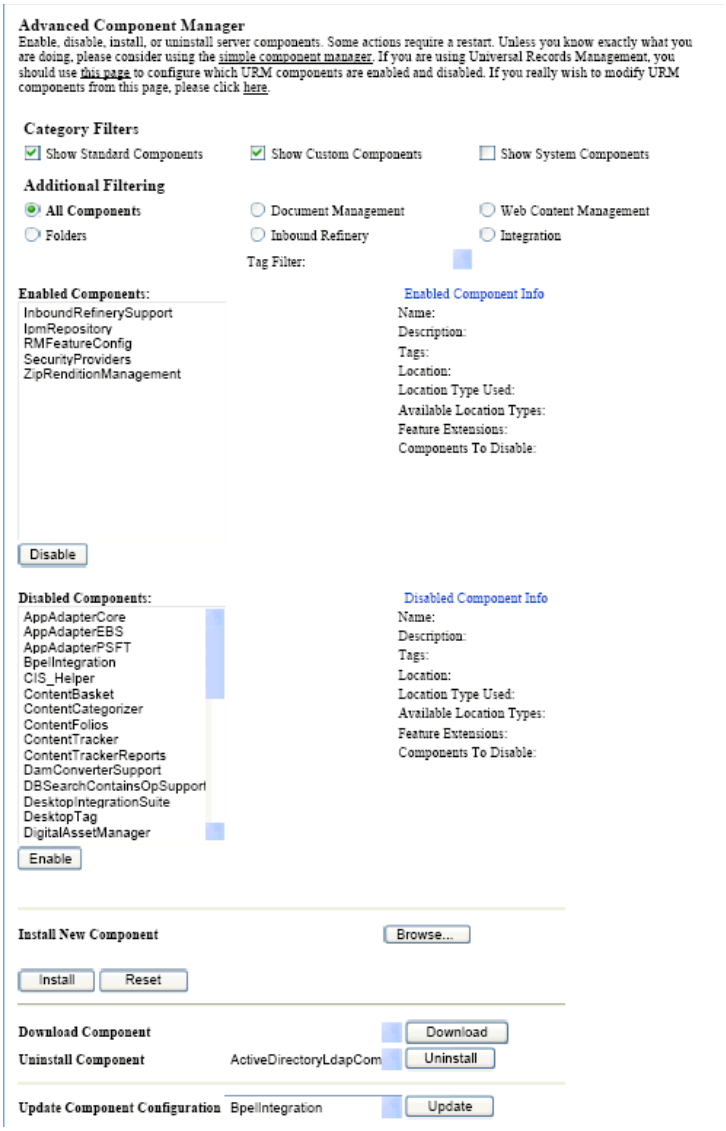
   The Admin Server displays the Component Manager page, which has lists of enabled and disabled components.

2. In the first paragraph on the Component Manager page, click **advanced component manager**.

3. On the Advanced Component Manager page, you can do these tasks:

   - View lists of enabled and disabled components by categories and other filters
   - Select a component to view details about it
   - Enable components
   - Disable components
   - Install custom components
   - Uninstall custom components

Figure 3–2 Advanced Component Manager Page



3.1.3 ComponentTool

ComponentTool is a command-line utility for installing, enabling, and disabling components in Oracle Content Server. After installing a component, ComponentTool automatically enables it. ComponentTool is located in the *DomainHome*/ucm/cs/bin directory.

3.1.4 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 Oracle Content Server what components are enabled and where to find the definition file for each component.
- The component definition (or *glue*) file, which tells Oracle Content Server where to find the resources for the custom component.
- Different custom resource files, which define your customization to standard Oracle Content Server resources.
- Template files, which define custom template pages.
- Other files which contain customization to Oracle Content Server graphics, Java code, help files, and so on.

For more detailed information about these files, see Section 3.2, "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 Oracle Content Server instances.
- A manifest.hda file that tells Oracle Content Server where to place the files that are unpackaged or uploaded from a component ZIP file.

3.1.5 Enabling and Disabling Components

By definition, a component is **enabled** when it is properly defined in the Components ResultSet in the idc_components.hda file. A component is **disabled** if there is no entry or the entry is not formatted correctly.

There are several ways to enable or disable a component:

- ComponentTool: Run *DomainHome*/ucm/*short-product-id*/bin/ComponentTool to enable or disable a component. For example:

```
ComponentTool -enable component_name
```

- Component Wizard: Choose **Enable** or **Disable** from the **Options** menu. For more information, see *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*.
- Component Manager: Select the checkbox next to a component name to enable a server component specified on the Component Manager screen. 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 System Administrator's Guide for Oracle Content Server*.
- Advanced Component Manager: On the Advanced Component Manager page, select a component name, and click **Disable** to disable the component, or click **Enable** to enable the component.

3.2 About Directories and Files

This section provides information about the files used in component creation and the directory structure used to store those files. It includes the following sections:

3.2.1 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 Oracle 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 Oracle Content Server.

The HDA file type is used to define the following component files:

- Components file (idc_components.hda)
- Component definition file
- Manifest file
- Dynamic table resource file
- Template resource file

The following example file is an idccs_components.hda file that points to a component called `customhelp`.

```
<?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
```

3.2.1.1 Elements in HDA Files

Each HDA file contains a **header line** and one or more **sections**. The header line identifies the Oracle 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 Oracle 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.

  An example of a `Properties` section in an HDA file follows:

  ```
  @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=Oracle 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`.

  The following example shows a ResultSet called `Scores` that has 4 columns and 3 rows.

```
@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

  Oracle Content Server uses some predefined ResultSets with the following names, which should not be used for a custom component table.

ResultSet Name	Location	Purpose
`Components`	*IntradocDir*/data/components/idc*custom-name*_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 *custom-name*.

ResultSet Name	Location	Purpose
IntradocReports	*IdcHomeDir*/resources/core/reports/reports.hda	Specifies the default report templates for Oracle Content Server.
IntradocTemplates	*IdcHomeDir*/resources/core/templates/templates.hda	Specifies all of the default templates for Oracle 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.
SearchResultTemplates	*IdcHomeDir*/resources/core/templates/templates.hda	Specifies the default search results templates for Oracle Content Server.

3.2.1.2 The idc_components.hda File

The idc_components.hda file is a text file that tells Oracle Content Server which components are enabled and where to find the definition file for each component.

The idc_components.hda file is always stored in the *IntradocDir*/data/components directory. The Component Wizard, Component Manager, and ComponentTool can be used 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-name*_components.hda. If the Admin Server does not find the idc*short-product-name*_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-name*_components.hda file containing the appropriate data.

The following example of an idccs_components.hda file lists several enabled components, such as schema, configuration migration, and SOAP.

```
@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
```

3.2.1.3 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 is 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.

> **Note:** Do not confuse the idc*short-product-name*_components.hda file with the *component_name*.hda file. The idc*short-product-name*_components.hda file is used to track all installed components. The *component_name*.hda file contains information that is specific to a single component.

The following example of a component definition file points to an environment resource file called customhelp_environment.cfg.

```
<?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
```

3.2.2 Custom Resource Files

Custom resource files define your Oracle 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.

The following table describes these resources.

Resource Type	File Type	Contents
HTML include	HTM	"Include" definitions
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 3.5, "Resources Detail."

In addition, a template.htm page is used by Oracle Content Server to assemble web pages. For more detailed information about the template.hdm file, see Section 3.5.8, "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.

3.2.3 Data Binder

Oracle 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 the *Oracle Fusion Middleware Idoc Script Reference Guide*.

3.2.3.1 LocalData

The `@Properties LocalData` section in an HDA file maps to the LocalData of the DataBinder. The LocalData information consists of name/value pairs.

LocalData information is maintained only during the lifetime of the Oracle 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.

3.2.3.2 ResultSets

Each `@ResultSet` section of an HDA file maps to a named result in the DataBinder. Some result sets can be made *active*, thus 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. 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.

3.2.3.3 Environment

Environment values are placed in the DataBinder as name/value pairs, which are defined in configuration files such as *IntradocDir*/config/config.cfg, intradoc.cfg, and environment-type resource files.

3.2.4 Manifest File

Manifest files are used to upload or unpackage a component ZIP file on Oracle Content Server. This file tells Oracle 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
`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 Oracle UCM 11*g*. Oracle UCM 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 Oracle 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 Oracle Content Server instances with the same installation directory path.

An example of a manifest.hda file follows:

```
@ResultSet Manifest
2
entryType
location
component
MyComponent/MyComponent.hda
componentExtra
MyComponent/readme.txt
images
MyComponent/
@end
```

3.2.5 Other Files

Your custom components can include any type of file that Oracle Content Server uses for functionality or to generate its look and feel.

3.2.5.1 Customized Site Files

You can add customized files for your site to change the look or actions of Oracle 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 Oracle 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 Oracle 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 Oracle UCM 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.

3.2.5.2 Component ZIP File

A component ZIP file contains all files that define an Oracle Content Server component. It can be unpackaged to deploy the component on other Oracle Content Server instances.

3.2.5.3 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.

3.2.6 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.

3.3 Development Recommendations

The following sections provide some guidelines to assist you in developing custom components:

- Section 3.3.1, "Creating a Component"
- Section 3.3.2, "Working with Component Files"
- Section 3.3.3, "Using a Development Instance"
- Section 3.3.4, "Component File Organization"
- Section 3.3.5, "Naming Conventions"

For more detailed information about creating or modifying components, see *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server* or online help.

3.3.1 Creating a Component

To create and enable a custom component, follow this basic procedure:

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 Oracle 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 Oracle Content Server files to create your custom resource files.
5. Test and revise your customization as necessary. You may need to restart Oracle Content Server to apply your changes.
6. If you want to package the component for later use or for deployment on other Oracle Content Servers instances, build the component and create a component ZIP file.

3.3.2 Working with Component Files

Two tools are available for working with component files:

- Component Wizard

  The Component Wizard is an Oracle 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 System Administrator's Guide for Oracle Content Server*.

- 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, thus helping you to 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 System Administrator's Guide for Oracle Content Server*.

3.3.3 Using a Development Instance

Whenever you are customizing Oracle Content Server, you should isolate your development efforts from your production system. Remember to include the same custom metadata fields on your development instance as you have defined for your production instance.

When you have successfully tested your modifications on a development instance, use the Component Wizard to build a component ZIP file and then unpackage the component on your production system.

Remember to restart Oracle Content Server after enabling or disabling a component.

If you are having problems with your Oracle Content Server instance after you have installed a custom component, disable the component and restart the instance. 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 the Oracle Content Server instance.

3.3.4 Component File Organization

To keep your custom components organized, follow these file structure guidelines. For more information, see Section 3.2.6, "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

Keep the following points in mind when considering files and their organization:

- 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.
- Oracle 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 Oracle Content Server, but all component files must be accessible to your Oracle Content Server instance.
- Images and other objects that are referenced by Oracle Content Server web pages must reside somewhere in the *DomainHome*/ucm/*short-product-id*/weblayout directory (so they can be accessed by the web server).

3.3.5 Naming Conventions

To keep your component files organized and ensure that the files work properly in Oracle 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 Oracle 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 Oracle 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 an .htm extension, and HDA file types should have an .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 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.
- Oracle Content Server is case sensitive even if your file system is not. For example, if a file is named My_Template, Oracle 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 Oracle 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 2.3.6, "Localized String Resolution."

3.4 Component File Detail

This section discusses the HDA file type and the component definition (glue) file in more detail, in the following subsections:

The information in this section is intended as reference material and should not be used to create files manually. You should always use the Component Wizard to create your component files.

3.4.1 The idc_components.hda File

The idc_components.hda file tells Oracle Content Server 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 Oracle UCM products: idccs_components.hda (for Oracle Content Server), idcibr_components.hda (for Oracle Inbound Refinery), and idcurm_components.hda (for Oracle Universal Records Management). The file is always stored in the *IntradocDir*/data/components directory.

3.4.1.1 Contents of idc_components.hda

The idc_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 Section 3.1.5, "Enabling and Disabling Components."

In the following example of an idccs_components.hda file, two components called `MyComponent` and `CustomHelp` are enabled.

```
<?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
```

3.4.1.2 Components ResultSet

The order that components are listed in the `Components` ResultSet determines the order that components are loaded when you start Oracle 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 Oracle 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 Oracle Content Server installation directory.

> **Note:** Always use forward slashes in the `location` path.

3.4.2 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 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. The following example shows a typical 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
```

3.4.2.1 ResourceDefinition ResultSet

The `ResourceDefinition` ResultSet table defines the type, file name, table names, and load order of custom resources.

3.4.2.1.1 Example of ResourceDefinition ResultSet The following example shows the structure of a `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
```

3.4.2.1.2 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.

3.4.2.2 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.

3.4.2.2.1 Example of MergeRules ResultSet The following example shows a `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
```

3.4.2.2.2 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 Oracle Content Server tables, such as `IntradocTemplates` or `QueryTable`.
- The `column` column specifies the name of the table column that Oracle Content Server uses to compare and update data.
  - Oracle 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.

3.4.2.3 Filters ResultSet

The `Filters` ResultSet table defines filters, which are used to execute custom Java code when certain Oracle Content Server events are triggered, such as when new content is checked in or when the server first starts. The following example shows a typical `Filters` ResultSet.

```
@ResultSet Filters
4
type
location
parameter
loadOrder
loadMetaOptionsLists
intradoc.server.ExecuteSubServiceFilter
GET_CHOICE_LIST_SUB
1
@end
```

3.4.2.4 ClassAliases ResultSet

The `ClassAliases` ResultSet table points to custom Java class files, which are used to extend the functionality of an entire Oracle Content Server Java class. The following example shows a typical `ClassAliases` ResultSet.

```
@ResultSet ClassAliases
2
classname
location
WorkflowDocImplementor
WorkflowCheck.CriteriaWorkflowImplementor
@end
```

3.5 Resources Detail

The information in this section is intended as reference material and should not be used to create any resource files manually. You should always use the Component Wizard to create your resource files.

Resources are the files that define and implement the actual customization you make to Oracle Content Server. Resources can be snippets of HTML code, dynamic page elements, queries that gather data from the database, services that perform Oracle 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.

There are two ways to create and edit a resource file:

- **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. Each resource type described in this section includes step-by-step instructions for using the Component Wizard to create and edit that type of resource.
- **Manual editing:** 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 *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server* or online help.

Note: You must restart Oracle Content Server after changing a resource file.

The following sections discuss these resource categories:

- Section 3.5.1, "HTML Include"
- Section 3.5.2, "Dynamic Data Tables"
- Section 3.5.3, "String"
- Section 3.5.4, "Dynamic Tables"
- Section 3.5.5, "Static Tables"
- Section 3.5.6, "Query"
- Section 3.5.7, "Service"
- Section 3.5.8, "Templates"
- Section 3.5.9, "Environment"

3.5.1 HTML Include

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.

3.5.1.1 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 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 3–1 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@>
```

3.5.1.2 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. Select 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, select 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, select the include, click **Delete**, and then click **Yes** to confirm.

3.5.2 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 Oracle 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 as follows in any resource file that can contain `dynamichtml` constructions:

```
<@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.

3.5.2.1 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 in this example:

```
<@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 (##). For example:

```
<@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 Oracle Content Server. For example:

```
<@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@>
```

3.5.2.2 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. Select 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. 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.

3.5.2.3 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. For example, here is a typical table declaration:

```
<@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`. For example, the following declaration specifies a table that is not to trim its values:

```
<@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

3.5.2.3.1 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 (using comma as the separator) the new value to the current value. Each of the subvalues in the comma separated list is assumed to be of the form key=value with =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 instead 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 Oracle 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.

3.5.2.3.2 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/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.

3.5.2.3.3 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.

3.5.2.3.4 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 result set, 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 result set 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 result set, 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).

3.5.2.4 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`

3.5.3 String

A `string` resource defines locale-sensitive text strings that are used in error messages and on Oracle Content Server web pages and applets. Strings are resolved by Oracle 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 Oracle 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 non-white space character.
\<	< (less than)
\>	> (greater than)
\&sp;	space (ASCII 32)
\&#xxx;	ASCII character represented by decimal number *xxx*

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. For example:

```
<@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 Oracle Content Server reads them correctly.

3.5.3.1 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}

Syntax	Meaning	Examples
{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}
{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}.

Syntax	Meaning	Examples
{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.

3.5.3.2 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. Select the resource in 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.

3.5.4 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 3.2.1.1, "Elements in HDA Files."

3.5.4.1 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.

3.5.4.2 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. Select the resource file in 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.

3.5.5 Static Tables

Static tables, HTML includes, and strings can be present in the same HTM file.

3.5.5.1 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.

3.5.5.2 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. Select the resource file in 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.

3.5.6 Query

A query resource defines SQL queries, which are used to manage information in the Oracle 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 Oracle Content Server queries are defined in the *QueryTable* table in the *IdcHomeDir*/resources/core/tables/query.htm file. You 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.

3.5.6.1 Query Example

The following example shows the standard *QdocInfo* query as defined in the *IntradocDir*/shared/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.

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.

Figure 3–3 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@>

```
<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>
```

3.5.6.2 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. Select the resource in the **Custom Resource Definition** list.
3. If there are multiple tables in the resource, select 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.

3.5.7 Service

A service resource defines a function or procedure that is performed by Oracle 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 an Oracle 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 Oracle Content Server or perform a specified function.

Important: This section provides an overview of custom service resources. For more information about Oracle Content Server services, see *Oracle Fusion Middleware Services Reference Guide for Oracle Universal Content Management*.

The standard Oracle 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.

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.

The following table row is an example of a service definition.

Figure 3–4 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)`
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 2.3.6, "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 appear in a row).	5:QdocInfo:**DOC_INFO**:6:!csUnableToDeleteItem(dDocName)!csRevisionNoLongerExist
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 2.3.6, "Localized String Resolution."	5:QdocInfo:DOC_INFO:6:**!csUnableToDeleteItem(dDocName)!csRevisionNoLongerExist**

3.5.7.1 Service Example

The DOC_INFO service provides a good example of how services, queries, and templates work together. The following figures show the DOC_INFO service definition from the *IntradocDir*/config/resources/std_services.htm file.

Figure 3–5 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@>

```
    <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>DocService
        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>
```

3.5.7.1.1 Attributes The following table describes the attributes of the DOC_INFO service shown previously.

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 on 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.

Attribute	Value	Description
Error Message	!csUnableToGetRevInfo	If this service fails on an English Oracle Content Server system, it returns this error message string: Unable to retrieve information about the revision

3.5.7.1.2 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.
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 Oracle 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.

Action Definition	Description
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 Oracle 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.

Action Definition	Description
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 Oracle Content Server system, it returns the error message string: `Unable to retrieve revision history for ''{dDocName}.''`

3.5.7.2 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. Select the resource in the **Custom Resource Definition** list.
3. If there are multiple tables in the resource, select 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.

3.5.8 Templates

A template resource defines the names, types, and locations of custom template files to be loaded for the component.

The actual template pages (.htm files) are separate files that are referenced in the template resource file. **Template HTM** files contain the code that the Oracle 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 Oracle 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 3.2.1.1, "Elements in HDA Files."

Merge rules are required to merge the new template definition into the **IntradocTemplates** table or the **SearchResultTemplates** table. Typically, the merge is on the **name** column. The following example shows a MergeRules ResultSet for a template.

```
@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 Oracle 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 Oracle 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 Oracle 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 Oracle 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.

The following example 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).

```
<?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
```

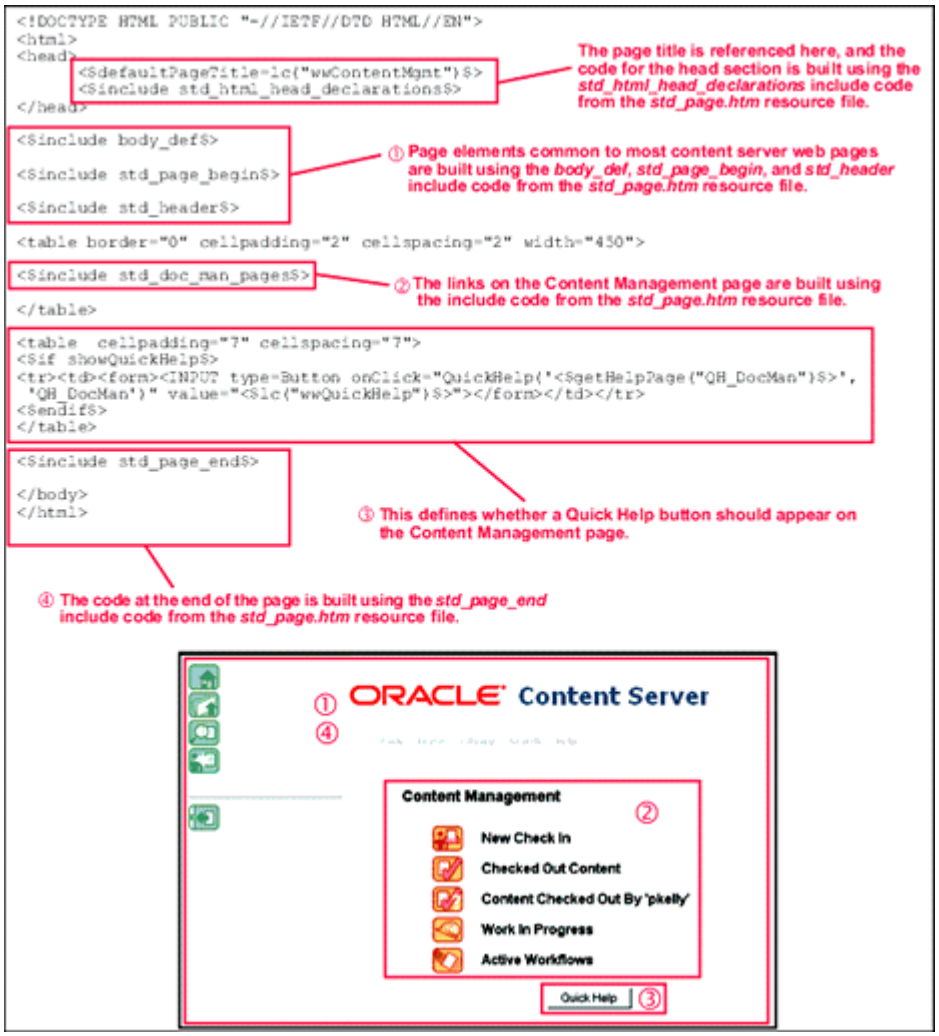
3.5.8.1 Template and Report Pages

Template pages and report pages are also called **presentation pages**, because Oracle 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.

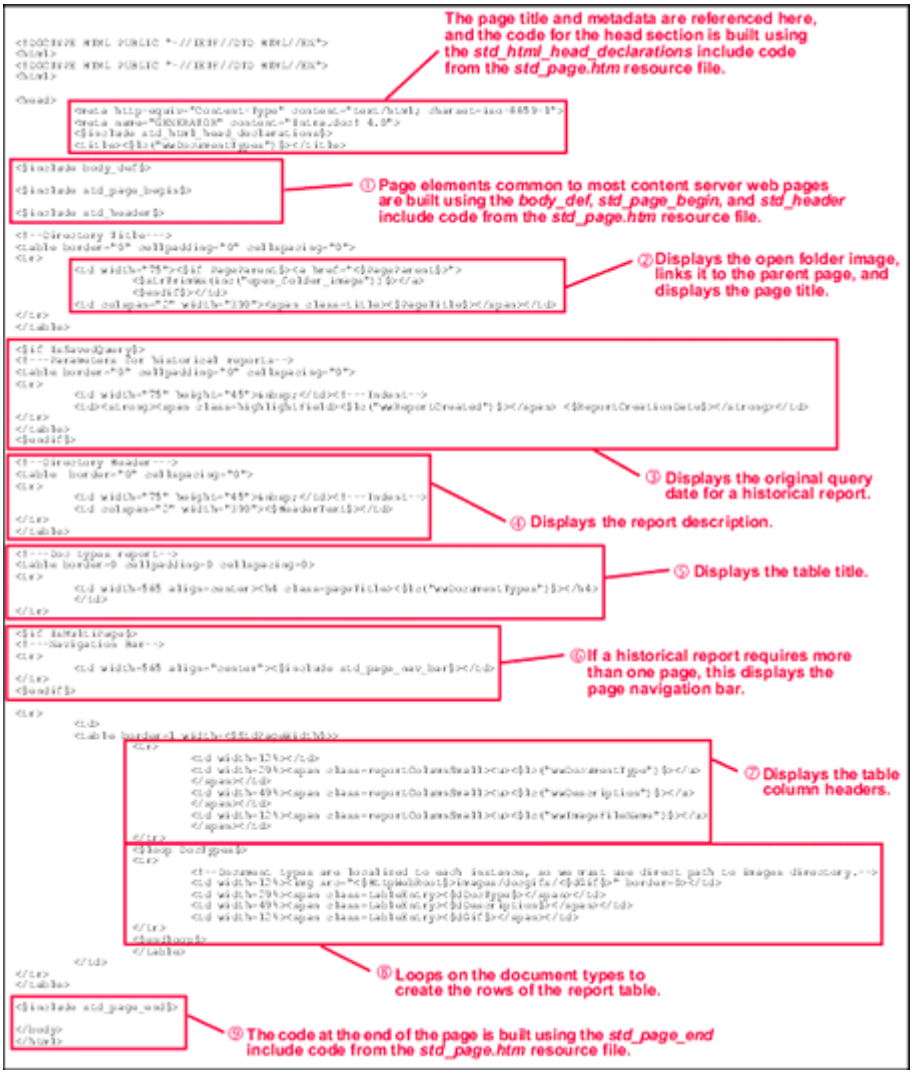
3.5.8.1.1 Template Page Example The following example shows the template file for the standard Content Management page (doc_man.htm).

Figure 3–6 Template Page Example



3.5.8.1.2 Report Page Example The following example shows the template file for the standard Document Types report page (doc_types.htm).

Figure 3–7 Report Page Example



3.5.8.2 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. Select the resource in 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, select 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.

3.5.9 Environment

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.
```

3.5.9.1 Environment Resource Example

Example 3–2 shows the contents of an environment resource file.

Example 3–2 Environment Resource

```
# 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 Idoc Script Reference Guide*.

3.5.9.2 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. Select the resource file in 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.

3.6 Installing Components

Server components for Oracle 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 Section 3.1.5, "Enabling and Disabling Components."

You can install components using one the methods that the following subsections describe:

- Section 3.6.1, "Installing a Component with Component Manager"
- Section 3.6.2, "Installing a Component with Component Wizard"
- Section 3.6.3, "Installing a Component with ComponentTool"

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.

3.6.1 Installing a Component with Component Manager

Follow these steps to install the component using the Component Manager:

1. Select **Admin Server** from the **Administration** tray.

   The Admin Server page is displayed with the Advanced 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 Oracle 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.

3.6.2 Installing a Component with 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 Content Server**, then your server instance, 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 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.

3.6.3 Installing a Component with ComponentTool

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
```

4

Changing the Look and Navigation of the Oracle Content Server Interface

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

This chapter includes the following sections:

- Section 4.1, "Modifying the Oracle Content Server Interface"
- Section 4.2, "Using Dynamic Server Pages to Alter the Navigation of Web Pages"

> **Tip:** In addition to the methods discussed here, you can also alter the metadata fields that are presented to users and modify the type of presentation used for check-in pages, search pages, and other user interfaces. For information about creating and modifying metadata fields and creating content profiles, see "Managing Repository Content" in *Oracle Fusion Middleware Application Administrator's Guide for Content Server*.

4.1 Modifying the Oracle Content Server Interface

This section describes how to modify the Oracle Content Server interface, in these subsections:

- Section 4.1.1, "Skins and Layouts"
- Section 4.1.2, "Customizing the Interface"
- Section 4.1.3, "Optimizing the Use of Published Files"

4.1.1 Skins and Layouts

Skins and layouts provide alternate color schemes and alternate navigation designs. The following sections describe the skins and layouts provided by default with Oracle Content Server:

- Section 4.1.1.1, "Types of Skins and Layouts"
- Section 4.1.1.2, "Selecting Skins and Layouts"
- Section 4.1.1.3, "Configuration Entries"
- Section 4.1.1.4, "Anonymous User Interface"

4.1.1.1 Types of Skins and Layouts

Some skins and layouts are provided by default with Oracle 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 4.1.1.3, "Configuration Entries."

4.1.1.1.1 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.

4.1.1.1.2 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.

4.1.1.2 Selecting Skins and Layouts

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

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

To change the skin or layout, follow these steps:

1. On the Oracle Content Server Home page, click ***your_user_name*** in the top menu bar. The User Profile page displays.
2. On the Oracle Content Server User Profile page, select the desired skin and layout.
3. Click **Update**, and view the changes.

4.1.1.3 Configuration Entries

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

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

4.1.1.4 Anonymous User Interface

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

When Oracle 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 Oracle 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 Oracle Content Server repository so it can be referenced by the Oracle Content Server templates.

The following files in the *IntradocDir*/data/users directory can be altered:

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

Use the following procedure to update the look-and-feel of the web site based on user login:

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

4.1.2 Customizing the Interface

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

The following sections provide an overview of this process.

- Section 4.1.2.1, "About Dynamic Publishing"
- Section 4.1.2.2, "Creating New Layouts"

4.1.2.1 About Dynamic Publishing

When the Oracle 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 Oracle 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 Oracle 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 used to create it did not have any access to the current service's DataBinder or any information about the current user.

This does limit the type of Idoc Script you can write in these files, so 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 thus increases the amount of bandwidth used.

4.1.2.2 Creating New Layouts

This section describes the general steps needed to create and publish new layouts.

1. Merge a table into the `LmLayouts` table in *IdcHomeDir*/resources/core/tables/std_resources.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_resources.htm. This new table describes the files that are created from Oracle 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 `PublishStaticFiles` table in std_resources.htm. This lists the directories that contain files, such as images, that should be published to the weblayout directory.

4.1.3 Optimizing the Use of Published Files

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

The following sections describe how to optimize the use of published files:

- Section 4.1.3.1, "Bundling Files"
- Section 4.1.3.2, "Referencing Published Files"

4.1.3.1 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/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 core Oracle Content Server JavaScript resources 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 UCM 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. Therefore, 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, Oracle Content Server can bundle multiple published files so they are delivered as one. This feature can be disabled 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_resources.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.

4.1.3.2 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.

To correctly include these files on a page, use the following Idoc Script and include it from somewhere within the HEAD of the page:

```
<$exec createPublishedResourcesList("javascript:common")$>
<$loop PublishedResources$>
<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` result set 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.

4.2 Using Dynamic Server Pages to Alter the Navigation of Web Pages

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

This chapter includes the following sections:

4.2.1 About Dynamic Server Pages

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

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

Dynamic server pages include the following file formats:

- **IDOC**: A proprietary scripting language
- **HCST**: Hypertext Content Server Template, similar to a standard Oracle 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, Oracle Content Server assembles web pages dynamically using a custom template (HCST, HCSP, or HCSF file) that you have checked in to Oracle Content Server. The template calls HTML includes from a text file (IDOC file) you have also checked in to Oracle 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.

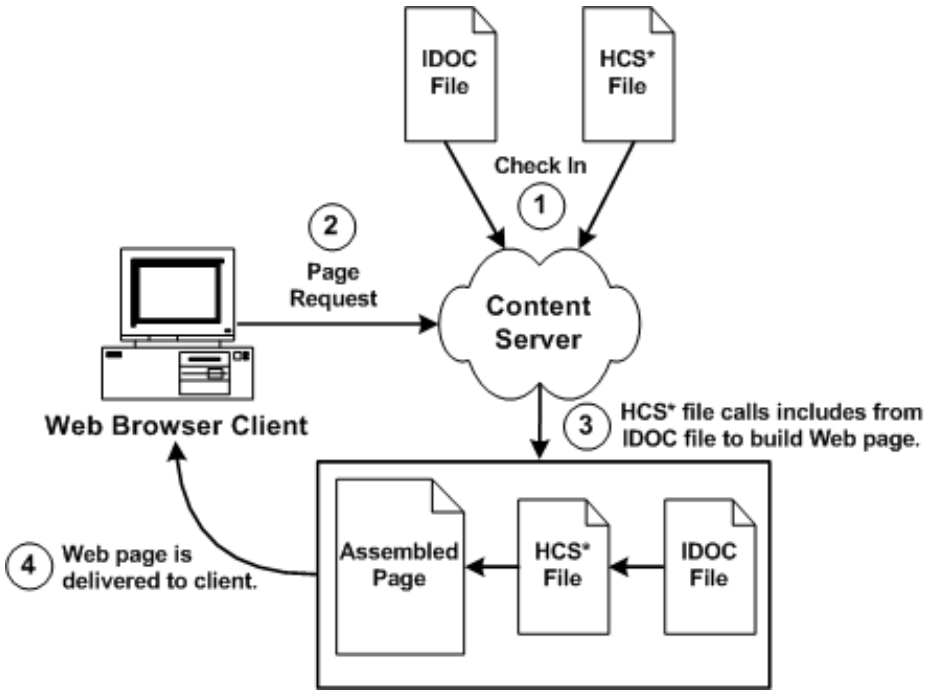
Using dynamic server pages with Oracle Content Server gives you these advantages:

- **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 Oracle Content Server.
- **Your web pages can make use of functionality not found in standard HTML.** For example, HTML forms can be submitted directly to Oracle Content Server without the need for CGI scripts. Also, Idoc Script enables you to work directly with environment and state information about Oracle 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 Oracle 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** Oracle 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 4–1 The Dynamic Server Page Process



4.2.2 Page Types

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

4.2.2.1 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 3, "Working with Standard, Server, and Custom Components."

4.2.2.2 HCST File

A Hypertext Content Server Template (HCST) file is a template page, similar to a standard Oracle 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 Oracle 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 Oracle Content Server.

4.2.2.3 HCSP File

A Hypertext Content Server Page (HCSP) file is a published web page that displays actual web site 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 Oracle Content Server through an HCSF page.
- Because this type of page contains web-viewable content, HCSP files are indexed in Oracle Content Server.

4.2.2.4 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 tags in the HCSF page.
- Because this type of page contains web-viewable content, HCSF files are indexed in Oracle Content Server.

  For more information about HCSF pages, see Section 4.2.2.4, "HCSF File."

4.2.3 Creating Dynamic Server Pages

Although dynamic server pages are implemented in Oracle Content Server differently than custom components, you must be familiar with Oracle UCM component architecture concepts, particularly Oracle Content Server templates and HTML includes. For more information, see Chapter 3, "Working with Standard, Server, and Custom Components."

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

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

4.2.4 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 Tags	<$ … $>	<$ … $>	<!--$ … --> [!--$ … --]	<!--$ … --> [!--$ … --]
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 uses standard HTML include coding. For more information, see Section 3.5.1, "HTML Include."

HCST uses standard Oracle Content Server template coding. For more information, see Section 3.5.8.1, "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.

4.2.4.1 Idoc Script Tags

For HCSP and HCSF pages, Idoc Script expressions are generally placed between HTML comment tags. When viewed statically, this allows 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.

For example:

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

In some situations, you may 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 of an Idoc Script expression.

For example:

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

In the preceding example, 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 (< >) in the standard HTML comment tags. This allows an XHTML parser to properly identify all the script when viewed statically.

For example:

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

4.2.4.2 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 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-->`

4.2.4.3 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/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 4.2.5.1, "docLoadResourceIncludes Function."

For example, in Idoc Script, a quotation mark can be included in a string by preceding it with a backslash escape character. However, 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 tag, but the second is not.

4.2.4.4 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`:

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

4.2.5 Idoc Script Functions

The following sections describe two special Idoc Script functions that are required for dynamic server pages:

- Section 4.2.5.1, "docLoadResourceIncludes Function"
- Section 4.2.5.2, "executeService Function"

4.2.5.1 docLoadResourceIncludes Function

To be able to use the HTML includes in an IDOC file, an HCS* file must call the docLoadResourceIncludes function. This function loads all the includes from the specified IDOC file for use in assembling the current page.

For example:

HCST file:

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

HCSP or HCSF file:

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

4.2.5.1.1 Requirements for Calling the docLoadResourceIncludes Function

- The native file for the specified content item must have an .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 4.2.4.3, "Special Characters."

4.2.5.1.2 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 of the IDOC file. This parameter should always be present when the Content ID is known. Error messages assume that it is present, as do 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 if no *Rendition* is specified. **Web—**The web-viewable file. **Alternate—**The alternate file.

4.2.5.2 executeService Function

The `executeService` function executes an Oracle 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 6, "Integrating Oracle UCM with Enterprise Applications."
- For a list of standard Oracle 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 Idoc Script Reference Guide*.
- For more information about services, see the Chapter 6, "Integrating Oracle UCM with Enterprise Applications."

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

4.2.6 Development Recommendations

This section provides some guidelines to assist you in developing dynamic server pages. It includes the following sections:

- Section 4.2.6.1, "General Tips"
- Section 4.2.6.2, "HCSF Tips"

4.2.6.1 General Tips

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 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 web site 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.

4.2.6.2 HCSF Tips

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 Content Server, and then passed back to the web browser, change the METHOD attribute in the include code from a POST to a 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 Content Server when it processes the form.

4.2.7 HCSF Pages

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

- Section 4.2.7.1, "Load Section"
- Section 4.2.7.2, "Data Section"
- Section 4.2.7.3, "Form Section"

For an example of a complete HCSF page, see Section 4.2.2.4, "HCSF File."

4.2.7.1 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. The following example shows a typical load section:

```
<!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 includes the items described in the following subsections:

- Section 4.2.7.1.1, "HTML Declaration"
- Section 4.2.7.1.2, "The docLoadResourceIncludes Function"
- Section 4.2.7.1.3, "Meta Tag"
- Section 4.2.7.1.4, "Variables and Includes"

4.2.7.1.1 HTML Declaration The HTML declaration identifies the file as an HTML file using the following syntax:

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

4.2.7.1.2 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 4.2.7.1.2, "The docLoadResourceIncludes Function."

4.2.7.1.3 Meta Tag The meta tag is used by Content Publisher to identify that this is a special type of page.

- This tag is not required if the form is not being published through Content Publisher.
- The meta tag 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">
  ```

4.2.7.1.4 Variables and Includes The <HEAD> section of your HCSF page can contain variable definitions and HTML includes as necessary. For example, the following lines define the default page title and load the `std_html_head_declarations` code:

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

4.2.7.2 Data Section

The data section 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 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 3.2.3, "Data Binder," and Section 3.2.1.1, "Elements in HDA Files."

The following subsections describe the Data Section:

- Section 4.2.7.2.1, "Data Section Structure"
- Section 4.2.7.2.2, "The idcformrules Tag"
- Section 4.2.7.2.3, "Metadata Tags"
- Section 4.2.7.2.4, "Nested Tags"
- Section 4.2.7.2.5, "Referencing XML Tags"
- Section 4.2.7.2.6, "Form Elements"
- Section 4.2.7.2.7, "ResultSets"

4.2.7.2.1 Data Section Structure The data section consists of XML tags that are placed between idcbegindata and idcenddata Idoc Script tags. For example:

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

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

4.2.7.2.2 The idcformrules Tag The idcformrules tag defines Oracle Content Server rules in the data section. This tag 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 format to specify that the form can be submitted again:

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

  - Use the following format to specify that the form cannot be submitted again. This results in a read-only form:.

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

- **resultsets Attribute**: The resultsets attribute indicates which XML tags 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, Oracle Content Server reads the resultsets attribute and, if necessary, places empty ResultSets with the specified names into the DataBinder so they are available for merging.

    For more information about ResultSet formatting in the data section, see Section 4.2.7.2.7, "ResultSets."

4.2.7.2.3 Metadata Tags Metadata tags specify the metadata values that appear in the form fields when the form is displayed in a browser. For example:

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

Each metadata tag can be assigned a `content` attribute that indicates which type of content the tag 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 tag should be interpreted strictly as text. HTML indicates that the content of the tag should be interpreted as HTML code.
- If the `content` attribute is not specified for a metadata tag, it defaults to `html`.
- Content Publisher ignores all other attributes except the `content` attribute.

4.2.7.2.4 Nested Tags If you are not publishing HCSF pages through Content Publisher, you can use nested XML tags (also called nodes) within the data section. In the following example, the `<section>` tag is nested in the `<chapter>` tag:

```
<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 tags are not allowed in Content Publisher.

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

```
chapter!section
```

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

  ```
  chapter!section:title
  ```

- If you reference a tag in the data section, the tag value can be merged back into the data section upon form submission only if one of the following are true:
  - The root tag has already been referenced in the data area.
  - The root tag 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 may contain Idoc Script for further evaluation. For example, to specify a default *dDocTitle*:

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

4.2.7.2.6 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.
- For example, the following form elements add the `mychapters!chapter` tag as a valid ResultSet if it is not already defined in the `idcformrules resultsets` attribute. It also adds, if necessary, the root tag `mychapters`.

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

4.2.7.2.7 ResultSets You can define a ResultSet using XML tags within the data section.

- You must use the `resultsets` attribute of the `idcformrules` tag to specify a ResultSet.
- The tags must be completely qualified, and the full reference path from the root node must be used.
- The columns in the ResultSet are the tag content and the tag attributes.
- For information about limitations on repeating and nesting XML tags in a ResultSet, see Example 4–2 and Example 4–3.

Example 4–1 Two ResultSets Defined by XML Tags

In the following example, two ResultSets named `volume` and `chapter` are defined by XML tags:

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

This evaluates into two ResultSets with two columns each:

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

Example 4–2 Repeated Tags in a ResultSet

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

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

In the following example, the `chapter` tag is repeated in the `chapter` ResultSet:

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

This evaluates into a ResultSet with two columns and two rows:

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

Example 4–3 Nested Tags in a ResultSet

A ResultSet can have nested tags, but the nested tags may not be repeated within a parent tag. For example, an additional <section> tag would not be allowed within the first <chapter> tag:

```
<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 evaluates into a ResultSet with four columns and four rows (the last two cells are blank):

```
@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
```

Example 4–4 Editing a ResultSet

- Updating a specific field in a ResultSet requires that you indicate the ResultSet row number in the request parameter. The # character is used by Oracle 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 empty rows are added sufficiently to provide a row to be edited.

  For example, to update the first row (row 0) of the ResultSet, you might use the following code:

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

- Insert new fields into a ResultSet by using the exclamation point character (!). For example, to insert author and title fields into the comment ResultSet, name the input fields comment!author and comment!title. If those fields are not in the ResultSet, they are added when the form is submitted.

- To delete a row in a ResultSet, empty all the values so they are blank. For example, to delete the first row entirely:

```
<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 name and row number. 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.
```

4.2.7.3 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 4.2.8.3, "Common Code for Forms."

4.2.7.3.1 Form Begin The form section begins with the following Idoc Script:

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

The std_html_form_submit_start include in the std_page.idoc resource file contains the following code, which creates a standard HTML form using a POST method, sets the IdcService to SUBMIT_HTML_FORM, and sets the dID variable to the value of the current 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="<$SourceID$>">
```

4.2.7.3.2 Form Properties The form table typically begins with the following property definitions, which create the fields as form fields, allow the fields to be edited, and set the size of the field caption area:

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

4.2.7.3.3 Form Fields The following lines are typically used to create each input field:

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

Note: Some fields may 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:

```
<@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 Oracle Content Server when it processes the form.

Example 4–5 Changing a Value in a Specific Column and Row in a Second Table When You Update a Row in the First Table

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:

```
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.

4.2.7.3.4 Form Buttons The following lines are typically used to create the form submission and reset buttons:

```
<input type=submit name=Submit value=" Submit ">
<input type=reset name=Reset value="Reset">
```

4.2.7.3.5 Form End After all the form elements and default values have been defined, the form must end with a `</form>` tag.

4.2.8 Working with Dynamic Server Pages

The following subsections present examples that show how the dynamic server pages work together to modify Oracle Content Server behavior:

- Section 4.2.8.1, "HCST and HCSP Example"
- Section 4–7, "HCSF Example"
- Section 4.2.8.3, "Common Code for Forms"

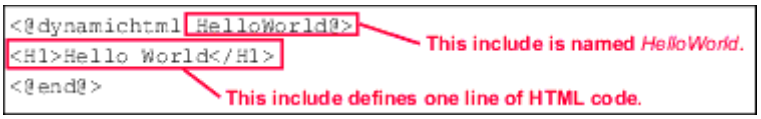
4.2.8.1 HCST and HCSP Example

Example 4–6 shows how to create simple HCST and HCSP pages.

Example 4–6 Creating an HCST Page and HCSP Page

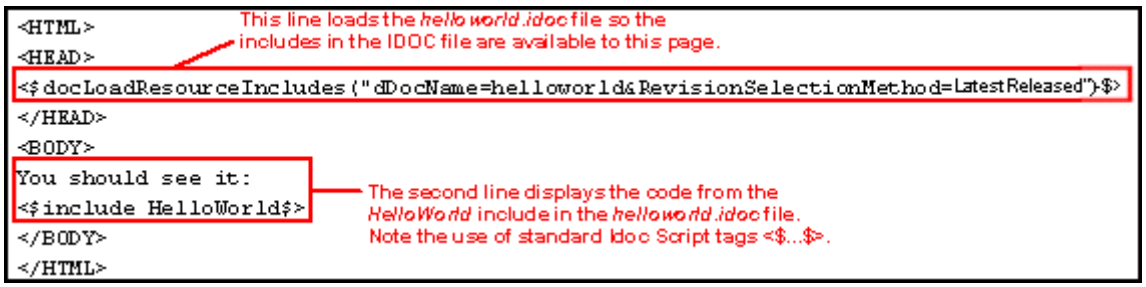
1. Create an IDOC file with a custom include.

Figure 4–2 Custom Include



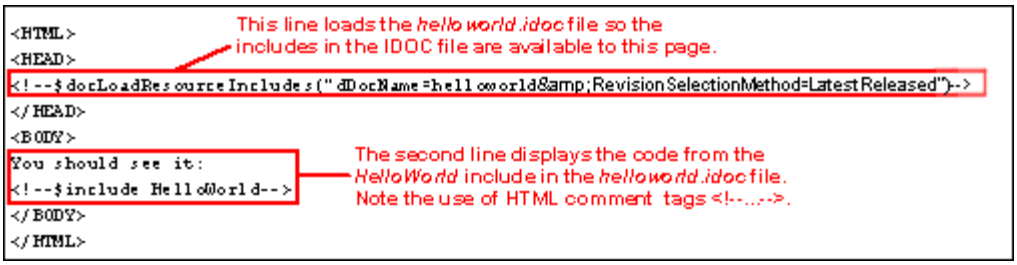
2. Save the file as `helloworld.idoc`.
3. Check in the IDOC file to Oracle Content Server with a Content ID of `helloworld`. The IDOC file is now available to any HCS* pages that reference it.
4. Create an HCST file that references the HelloWorld include:

Figure 4–3 HCST File Referencing Custom Include



5. Save the file as `helloworld.hcst`.
6. Check in the HCST file to Oracle Content Server.
7. Create an HCSP file that references the HelloWorld include:

Figure 4–4 HCSP File Referencing Custom Include



8. Save the file as `helloworld.hcsp`.
9. Check in the HCSP file to Oracle Content Server.
10. Search for the `helloworld` content items in Oracle Content Server.
11. Display the HCST file and HCSP files in your web browser. They should both look like the example in Figure 4–5.

Figure 4–5 HelloWorld Content Item Displayed in a Web Browser

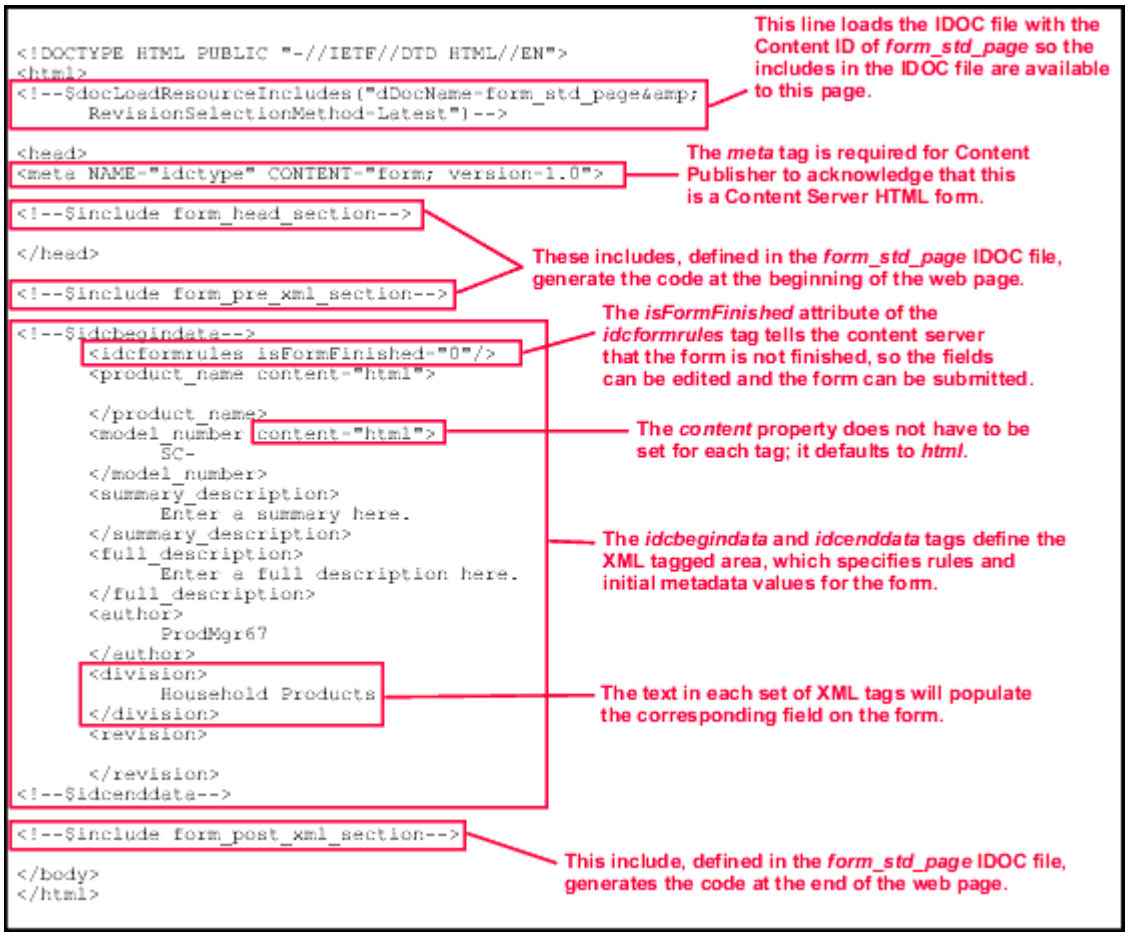
4.2.8.2 HCSF Example

Example 4–7 shows a typical HCSF page and its associated IDOC file. This example creates a form that users can fill out and submit to enter product descriptions as content items.

Example 4–7 HCSF Example

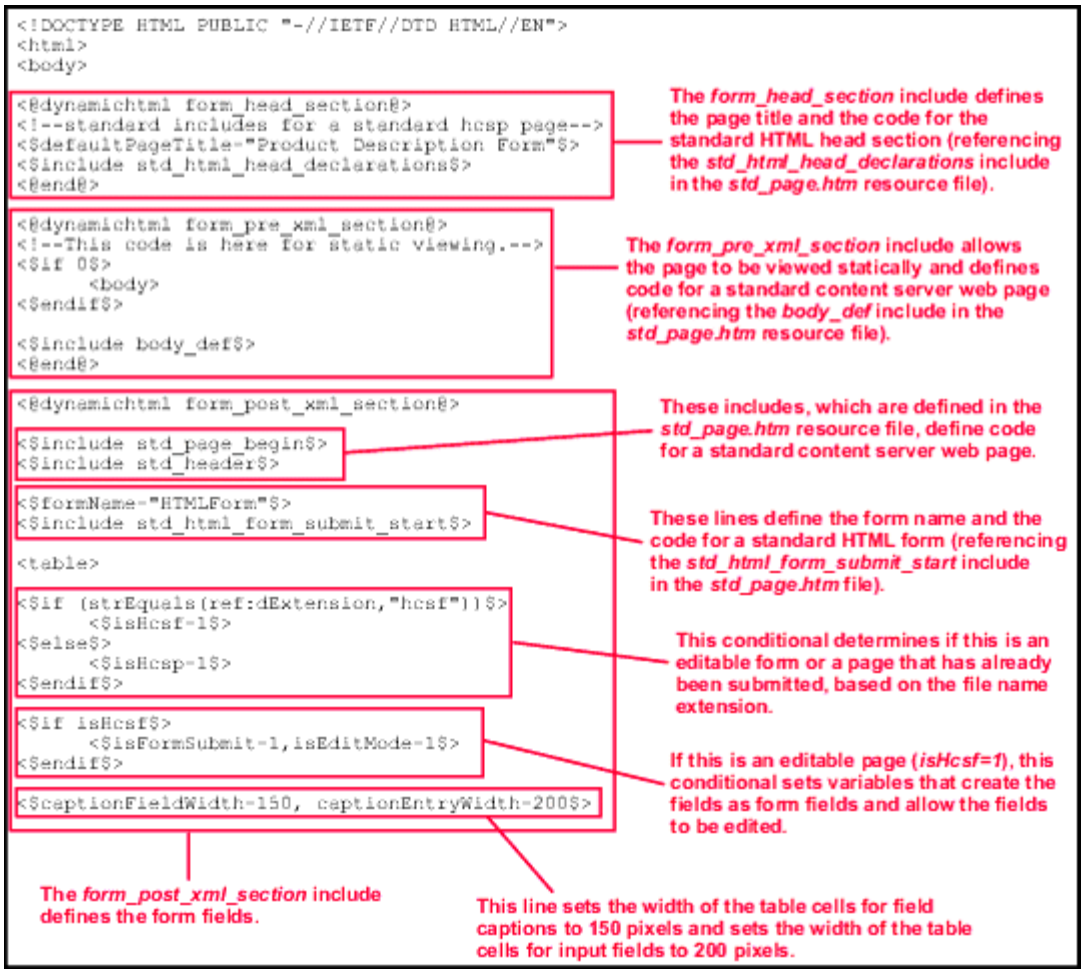
1. Create an HCSF file that references an IDOC file named `form_std_page`, as Figure 4–6 shows.

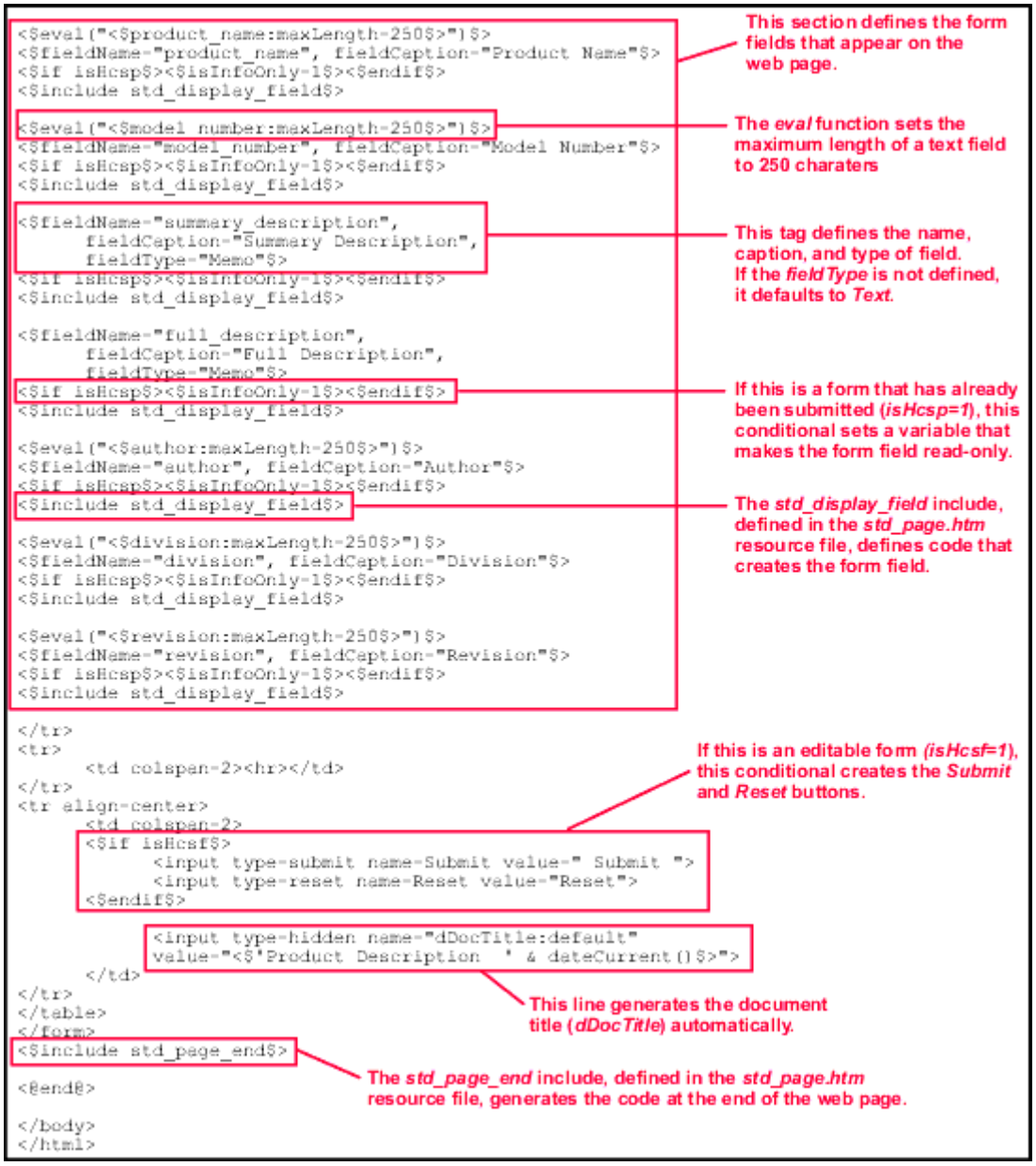
Figure 4–6 Product Description Form HCSF File



2. Save the file as `product_form.hcsf`.
3. Check in the HCSF file to Oracle Content Server.
4. Create an IDOC file with custom includes, as Figure 4–7 shows.

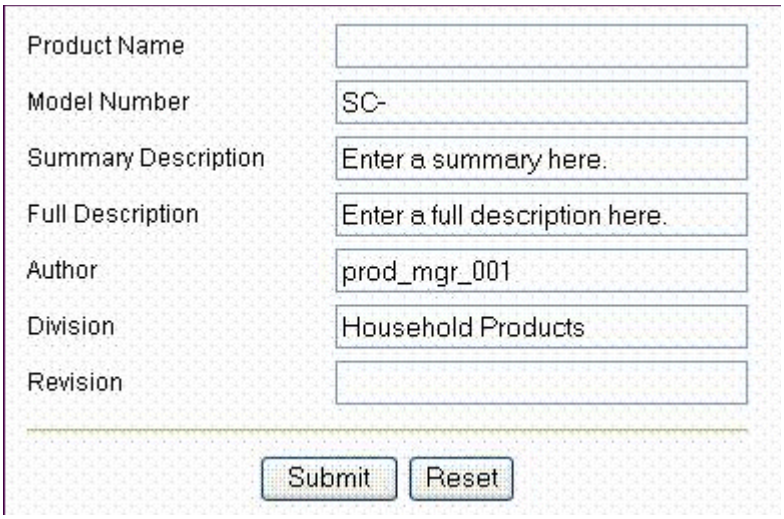
Figure 4–7 IDOC File with Custom Includes





5. Save the file as `form_std_page.idoc`.
6. Check in the IDOC file to Oracle 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 Oracle 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 4–8.

Figure 4–8 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 Oracle Content Server.

11. Click the link to display the HCSP page in your web browser. Figure 4–9 shows how it should look.

Figure 4–9 Link Displaying an 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

4.2.8.3 Common Code for Forms

This section describes some of the features that are commonly used in HCSF pages and associated IDOC files.

4.2.8.3.1 Retrieving File Information Executing the service DOC_INFO_SIMPLE makes metadata from a specific file available to the page. For example:

```
<$dID=SourceID$>
<$executeService("DOC_INFO_SIMPLE")$>
```

4.2.8.3.2 Referencing the File Extension Use the following statement to determine whether the form is submitted (hcsp) or unsubmitted (hcsf):

```
<$if (strEquals(ref:dExtension,"hcsf"))$>
    <$isHcsf=1$>
<$else$>
    <$isHcsp=1$>
<$endif$>
```

For information about the *ref:* prefix, see Section 4.2.4.4, "Referencing Metadata."

4.2.8.3.3 Defining Form Information The following code defines the form name and the standard include to start an HTML form:

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

The following is typical code that defines form properties:

```
<table border=0 width=100%>
<$isEditMode=1,isFormSubmit=1$>
<$captionFieldWidth="25%", captionEntryWidth="75%"$>
```

4.2.8.3.4 Defining Form Fields Use standard Idoc Script variables and the std_display_field include to display the form fields. For example:

```
<$fieldName="news_
author",fieldDefault=dUser,fieldCaption="Author",isRequired=1,requiredMsg =
"Please specify the author."$>
<$include std_display_field$>
```

Some fields might require extra code to display the field correctly. For example, 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:

- Standard std_memo_entry Include

```
<@dynamichtml std_memo_entry@>
    <textarea name="<$fieldName$>" rows=3 cols=40 wrap=virtual> <$fieldValue$></textarea>
<@end@>
```

- Custom std_memo_entry Include

```
<@dynamichtml std_memo_entry@>
    <textarea name=<$fieldName$> rows=15 cols=50 wrap=virtual><$fieldValue$></textarea>
<@end@>
```

4.2.8.3.5 Defining Hidden Fields You can specify metadata for a submitted form (hcsp) by defining a hidden field, which contributors cannot change. For example, use the following code to assign the document type News_Forms to each submitted form:

```
<input type=hidden name="dDocType" value="News_Forms">
```

To specify the security group of the submitted forms:

```
<input type=hidden name="dSecurityGroup" value="Public">
```

4.2.8.3.6 Submitting the 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)">
```

5 Modifying System Functionality

This chapter describes how to change the basic functionality of Oracle Content Server.

This chapter includes the following sections:

5.1 Changing System Settings

Oracle 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 Oracle Content Server to customize your content management system settings:

- Admin Server: The Admin Server is a collection of web pages that you can use to configure systemwide settings for Oracle Content Server. To access these pages, click **Admin Server** from the **Administration** tray in the portal navigation bar to display 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.
- System Properties: The System Properties administration application is used to configure systemwide Oracle Content Server 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 the Oracle Content Server instance
  - 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

Oracle WebLogic Server is the primary tool for setting system properties for Oracle UCM; 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.

- 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 web sites outside your local site. For detailed information, see *Oracle Fusion Middleware Application Administrator's Guide for Content Server*.
- User administration: You can define security groups, aliases, roles, and accounts for the users at your site using the User Admin function. To access this screen, select **Admin Applets** from **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.
- Other administration customizations: In addition to the system settings that are discussed here, other settings can be changed to match your site's needs:
  - 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

5.2 Using Components

Components are modular programs that are designed to interact with Oracle Content Server at runtime. The **component architecture** model is derived from object-oriented technologies, and encourages the use of small modules to customize Oracle 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 Content Server. For example, you could use a component could to perform any of these tasks:

- Modify the standard security features
- Change the way search results are requested and returned
- Enable Oracle Content Server to work with a particular system (such as a Macintosh client or a proprietary CAD program)

Using component architecture with Oracle Content Server gives you these advantages:

- **You can modify source code without compromising the integrity of the product.** Oracle 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 Content Server instances. Many custom components can work on Oracle 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 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 Content Server **instance without compromising customizations.** Custom components override existing product resources rather than replace them. Replacing the standard Oracle 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 Content Server **core functionality.** Because new functionality may change the way your components behave, customizations are not guaranteed to work for future Oracle 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 Content Server. Components provided with Oracle 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 Content Server instances. For more information, see Section 3.1.1, "Component Wizard."

- The Advanced Component Manager provides a way to manage custom components in Oracle Content Server. By using the Advanced Component Manager, you can add new components and enable or disable components for Oracle Content Server. For more information, see Section 3.1.2, "Advanced Component Manager."
- The ComponentTool is a command-line utility for installing, enabling, and disabling components for Oracle Content Server.

For information about component architecture and creation, see Chapter 3, "Working with Standard, Server, and Custom Components."

5.3 Changing Configuration Information

For advanced customizations and integration with other business systems, Oracle Content Server supports several development tools and technologies, such as the following:

- VBScript
- ASP
- J++
- JavaScript
- ASP+
- J2EE
- Java
- JSP
- COM
- Visual Basic
- DreamWeaver
- .Net
- C++
- Visual InterDev

In addition to these tools, the proprietary Idoc Script is a server-side custom scripting language for Oracle Content Server. It is used to reference variables, to conditionally include content in HTML pages, and to 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.

Idoc Script is primarily used in the following situations:

- For **include** code, an include defines pieces of code used to build Oracle 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 Content Server web site.

  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 Content Server behavior. Variable values can be stored in an environment resource, such as the config.cfg file and many are predefined in Oracle Content Server. You can also define your own custom variables.
- For **functions**, many built-in global functions are used in Oracle 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 Idoc Script Reference Guide*.

5.4 Customizing Services

Oracle Content Server services are functions or procedures performed by Oracle Content Server. Calling an Oracle Content Server service (making a service request) is the only way to communicate with Oracle Content Server or to access the database.

Any service can be called externally (from outside Oracle Content Server) or internally (from within Oracle 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 Oracle 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**. The attributes define the following aspects of the service:
  - the **service class**, which specifies which Java class the service has access to. This determines what actions can be performed by the service.
  - the **access level**, which assigns a user permission level to the service.
  - a **template page** that specifies the template that displays the results of the service.
  - the **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.
  - the **error message** that 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 thus customizing Oracle Content Server. For more information, see Chapter 6, "Integrating Oracle UCM with Enterprise Applications."

5.5 Generating Action Menus

In previous versions of Oracle 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 Oracle 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 result sets of data then calling specific resource includes which use that data to display the page. Result sets 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.

5.5.1 Creating Display Tables

Different display tables are used for the search results page for each display type (Headline or Thumbnail), with an API for each, as the following sections describe:

- Section 5.5.1.1, "Headline View Tables"
- Section 5.5.1.2, "Thumbnail View Tables"

One of the first steps in any table setup is to retrieve documents to display, as in this example:

```
<$QueryText = "dDocAuthor <matches> `sysadmin`"$>
<$executeService("GET_SEARCH_RESULTS")$>
```

5.5.1.1 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 result set that describes the columns of the table, as in this example:

```
<$exec rsCreateResultSet("ColumnProperties",
  "id,width,headerLabel,rowAlign")$>

<$exec rsAppendNewRow("ColumnProperties")$>
<$ColumnProperties.id = "dDocName"$>
<$ColumnProperties.width = "150px"$>
<$ColumnProperties.headerLabel = lc("wwDocNameTag")$>
<$ColumnProperties.rowAlign = "center"$>

<$exec rsAppendNewRow("ColumnProperties")$>
<$ColumnProperties.id = "dDocTitle"$>
<$ColumnProperties.width = "auto"$>
<$ColumnProperties.headerLabel = lc("wwTitle")$>
<$ColumnProperties.rowAlign = "left"$>

<$exec rsAppendNewRow("ColumnProperties")$>
<$ColumnProperties.id = "actions"$>
<$ColumnProperties.width = "75px"$>
<$ColumnProperties.headerLabel = lc("wwActions")$>
<$ColumnProperties.rowAlign = "center"$>
```

A result set 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 determine what data will be displayed in each row of the table.

```
<$exec rsCreateResultSet("RowData","dDocName,dDocTitle,actions")$>
<$exec rsAppendNewRow("RowData")$>
<$RowData.dDocName = "<$dDocName$>"$>
<$RowData.dDocTitle = "<$dDocTitle$>"$>
<$RowData.actions = "<$include doc_info_action_image$>"$>
```

The ColumnProperties result set technically has a row for each column in the table, while in RowData, there is only one row. Data entered into this result set is of the following form:

```
<$RowData.%COLUMN_ID% = "%IDOCSCRIPT%"$>
```

Each column in the `RowData` result set refers to an actual column that will appear in the final table. Each column in this result set has a corresponding "ID" in the `ColumnProperties` result set declared earlier. An Idoc Script expression is assigned to each cell in this result set. 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` result set 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` result set.
2. Set `rowIncludeName` to the name of the resource include to create.
3. Include `create_slim_table_row_include` again.

The following code displays the 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 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. For example, a custom include similar to the following can be declared from 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.

5.5.1.2 Thumbnail View Tables

The table for the Thumbnail View is created differently. The `ColumnProperties` or `RowData` result sets are not constructed. Instead, the number of columns are set and an Idoc Script include name is used to "paint" each cell. This is less easy to customize and less data-driven than the other methods, but this type of table is also much less structured.

```
<$numDamColumns = 4$>
<$damCellIncludeName = "my_sample_dam_cell"$>
<$include dam_table_header$>
<$loop SearchResults$>
  <$include dam_table_item$>
<$endloop$>
<$include dam_table_footer$>
```

5.5.2 Customizing Action Menus

The first step in customization is to add the action menu icon to the `Actions` column. The following example incorporates an action menu into each row of the Headline View sample table used previously.

```
<$RowData.actions = "<$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.

The following code creates the data to be used to construct this 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 result set 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 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 action 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, set set `<$actionPopupContainerIncludeName = new_include_name$>` in the script.

Use the following code to have this include called for each row of the Headline View table.

```
<$exec rsCreateResultSet("PopupData", "actions")$>
<$exec rsAppendNewRow("PopupData")$>
<$PopupData.actions="<$include action_popup_container_include$>"$>
```

This code creates a `PopupData` result set similar to the `RowData` result set. 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 action 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` result set 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. For example:

```
<$disabledActionPopupClasses = "workflow,folders"$>
<$disabledActionPopupIds = "getNativeFile,alertDocName"$>
```

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 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`

6

Integrating Oracle UCM with Enterprise Applications

This chapter describes how to integrate Oracle Universal Content Management (Oracle UCM) with enterprise applications.

This chapter includes the following sections:

6.1 Overview of Integration Methods

Several easy, flexible methods are available for integrating Oracle Content Server with enterprise applications such as application servers, catalog solutions, personalization applications, and enterprise portals, and client-side software.

Oracle Content Server not only serves as a content management solution for content-centric web sites, 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.

In general, these integration methodologies serve to translate or pass methods and associated parameters with the goal of executing Oracle Content Server services. The various Oracle Content Server services are the "window" for accessing the content and content management functions within Oracle Universal Content Management. For example, one simple integration option is to reference content that is managed within Oracle UCM by persistent URL. Other integration options are 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 Universal Content Management within network system environments using various protocols, interfaces, and mapping services.

For information about using the IdcCommand utility to access Oracle Content Server services from other applications, see Chapter 7, "Using the IdcCommand Utility to Access Services."

For information about the COM interface, see Chapter 8, "Using the COM API for Integration."

For information about Remote Intradoc Client (RIDC) integration, see Chapter 9, "Using Remote Intradoc Client (RIDC)."

6.2 JSP Integration

You can access the Oracle Content Server core functionality from JavaServer Pages (JSP) to deliver forms and custom pages using any of these methods:

- Through the JSP page execution functionality using the built in Apache Jakarta Tomcat Server
- Through a separate product, *Content Integration Suite*

  For more information, see Chapter 10, "Using Content Integration Suite (CIS).".

The following subsections describe JSP integration with Oracle Content Server:

- Section 6.2.1, "JSP Execution"
- Section 6.2.2, "Tomcat"
- Section 6.2.3, "Features"
- Section 6.2.4, "Configuring JSP Support"

6.2.1 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 Oracle Content Server.

The Apache Jakarta Tomcat Server is a free, open-source server of Java Servlet and JavaServer Pages that is run inside of Oracle Content Server when the feature is enabled. The integration of Tomcat Server with Oracle UCM provides the benefit of increased performance for content delivery.

Using JSP Execution functionality enables developers to access and modify Oracle 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 Oracle Content Server.

Important: JSP pages can execute Idoc Script functions only when the JSP page is being served on Oracle 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 Content Server provides revision control but does not provide dynamic execution of Idoc Script functions on the presentation tier (JSP server).

6.2.2 Tomcat

The capability for JSP to call services is provided by integrating the Tomcat 5.025 server with the Oracle 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 Oracle 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 Content Server core capabilities.
- Although Tomcat is embedded in Oracle 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/`).

6.2.3 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 Enterprise Content Management Suite distribution media also includes the current Java EE SDK.

6.2.4 Configuring JSP Support

Use the following procedure to enable and configure JSP support.

1. In Oracle 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 Oracle Content Server.

6.2.5 Loading Example Pages

Use either of the following procedures to load example pages into Oracle 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.

6.3 Java 2 Enterprise Edition Integration (J2EE)

The J2EE integration for Oracle Content Server is available with Content Integration Suite, a separate product.

Content Integration Suite (CIS) enables communication with Oracle Content Server and is deployable on a number of J2EE application servers, in addition to working in non-J2EE environments. A supported version of Oracle Content Server is required.

For more information, see Chapter 10, "Using Content Integration Suite (CIS)."

6.4 Web Services

The following subsections provide an overview of web services, general information about WSDL files and the SOAP protocol, and several basic implementation architectures.

6.4.1 Web Services Framework

Web services reside as a layer on top of existing software systems such as application servers, .NET servers, and Oracle Content Server. Web services can be used as a bridge to dissimilar operating systems or programming languages.

Web services are adapted to the Internet as the model for communication and rely on the HyperText Transfer Protocol (HTTP) as the default network protocol. Thus, using web services, you can build applications using a combination of components.

Oracle Content Server provides some web services built into the core product. Oracle WebLogic Server provides SOAP capabilities, and Oracle Content Server supports several SOAP requests through Oracle WebLogic Server. Additionally, the WSDL Generator component is installed (enabled) by default with Oracle Content Server. For more information, see Chapter 12, "Using Oracle UCM Web Services."

The core enabling technologies for web services are XML, WSDL, SOAP, and UDDL:

- 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 differing programming languages, middleware, and database management systems.
- SOAP: Communication: The Simple Object Access Protocol (SOAP) provides the Oracle Content Server communication for web services interfaces to communicate to each other over a network. SOAP is an XML-based communication protocol used to access web services. Web services receive requests and return responses using SOAP packets encapsulated within an XML document.
- 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 registration and lookup of web services.

Public or private UDDI sources are not published. However this does not prevent users from integrating Oracle Content Server with other applications using web services.

The XML, WSDL, SOAP, and UDDI technologies work together as layers on the web services protocol stack. The web services protocol stack consists of these layers:

- The service **transport** layer between applications (HTTP). While several protocols are available as a transport layer (for example, HTTP, SMTP, FTP, BEEP), the HTTP protocol is most commonly used. The WSDL Generator component relies on the HTTP protocol as the transport layer.
- The **messaging** layer that provides a common communication method (XML and SOAP).
- The service **description** layer that describes the public interface to a specific web service (WSDL).
- The service **discovery** layer that provides registry and repository services for storing and retrieving web services interfaces (UDDI).

6.4.2 Virtual Folders and WebDAV Integration

The Folders/WebDAV component is available as an extra component for download from the support site. You can use the Folders component to set up an interface to Oracle Content Server in the form of virtual folders that enable you to create a multilevel folder structure and also use the WebDAV component to remotely author and manage your content using clients that support the WebDAV protocol.

- The Folders component provides a hierarchical folder interface to content in Oracle Content Server. The component is required for WebDAV functionality, and the WebDAV Client product.

- The WebDAV component enables WebDAV (Web-Based Distributed Authoring and Versioning) functionality 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.

The option to install the WebDAV component is provided during the Folders/WebDAV installation process. For more information, see the *Oracle Fusion Middleware Application Administrator's Guide for Content Server*.

6.4.2.1 Virtual Folders

The Folders component sets up an interface to Oracle 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 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.

6.4.2.2 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 web site 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/move pages on a web server)
- Access control

When WebDAV is used with a content management system such as Oracle 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 Oracle Content Server web browser interface or a WebDAV client.

In Oracle Content Server, the WebDAV interface is based on the hierarchical Folders interface. For more information, see Section 6.4.2.1, "Virtual Folders."

6.4.2.2.1 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 Oracle UCM WebDAV Client is a different product that enhances the WebDAV interface to Oracle 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 Content Server repository.

The Desktop software package also includes a WebDAV Client component and a Check Out and Open component.

6.4.2.2.2 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 Oracle Content Server, the WebDAV server is used only as an interpreter between clients and Oracle Content Server.

6.4.2.2.3 WebDAV Architecture WebDAV is implemented in Oracle Content Server by the WebDAV component. The architecture of a WebDAV request follows these steps:

1. The WebDAV client makes a request to Oracle Content Server.
2. The message is processed by the web server (through a DLL in IIS).
3. On Oracle 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 Content Server.
   - Converts the client request from a WebDAV request to the appropriate Oracle Content Server request.
   - Connects to the core Oracle Content Server and executes the Oracle Content Server request.
4. The WebDAV component converts the Oracle Content Server response into a WebDAV response and returns it to the WebDAV client.

7

Using the IdcCommand Utility to Access Services

This chapter describes how to use the IdcCommand utility to access Oracle Content Server services from other applications

This chapter includes the following sections:

7.1 Overview of IdcCommand Utility

The IdcCommand utility is a standalone Java application that executes Content Server services. Almost any action you can perform from the Oracle Content Server browser interface or administration applets can be executed from IdcCommand.

The program reads 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 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.
- An Oracle Content Server user name. This user must have permission to execute the services being called.
- A path and file name for a log file.
- The connection mode (auto, server, or standalone).

There are certain commands that cannot be executed in standalone mode. In general, these commands are performed asynchronously by the server 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 3, "Working with Standard, Server, and Custom Components." and the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management*.

7.2 IdcCommand Setup and Execution

To set up IdcCommand, you must specify the following two things:

- A Command File, which specifies the services to be executed and any service parameters.
- 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 such as:

    ```
    IdcCommandFile=newfile.hda
    IdcCommandUserName=sysadmin
    IdcCommandLog=C:/domain/newlog.txt
    ConnectionMode=server
    ```

  - On the command line when running IdcCommand, specifying option flags such as:

    ```
    -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 Configuration Options either from the command line or in a configuration file. For more information, see Section 7.5, "Running IdcCommand."

7.3 Command File

The command file defines the service commands and parameters that are executed by the IdcCommand utility. The following subsections describe the rules for command files:

- Section 7.3.1, "Command File Syntax"
- Section 7.3.2, "Precedence"
- Section 7.3.3, "Special Tags and Characters"

7.3.1 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.
- For example, the following command file executes the ADD_USER service and defines attributes for two new users:

```
<?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>>
```

7.3.2 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.

7.3.3 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 7.3.1, "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.

7.4 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.

7.4.1 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 7.3, "Command File."

7.4.2 User

You must specify an Oracle Content Server user name. This user must have permission to execute the services being called.

7.4.3 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.

7.4.4 Connection Mode

You can specify the connection mode for executing the IdcCommand services.

Connection Mode	Description
auto	IdcCommand attempts to connect to the Oracle Content Server instance. If this fails, services are executed in standalone mode. This is the default connection mode.
server	IdcCommand executes services only through Oracle 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.

7.5 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 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. For more information, see Section 7.4, "Configuration Options."

   ```
   IdcCommandFile=newfile.hda
   IdcCommandUserName=sysadmin
   IdcCommandLog=C:/domain/newlog.txt
   ```

5. Run the IdcCommand stored in the *DomainHome*/ucm/cs/bin directory:

   ```
   IdcCommand.exe
   ```

7.6 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 which 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 the 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 Oracle 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` 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 Oracle 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*.

7.6.1 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.

7.6.2 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`
  - Examples:

    ```
    TYPE_PASSWD_FILE=file
    PASSWD_FILE_sys5=/etc/passwd
    PASSWD_FILE_bsd=/etc/master.passwd
    ```

    The 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`
  - Examples:

```
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` id 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 Content Server. 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`
  - Examples:

```
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`
  - Examples:

```
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`
  - Examples:

```
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`
  - Examples:

```
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` 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 quoted and assigned to `VDK_PLATFORM`.

- lookuplist
  - Examples:

```
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` class 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 quote the final value. In this example, assume the current value of `STARTUP_CLASS` is *version*. `STARTUP_CLASS` is replaced with the value `Installer --version`.

7.6.3 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 may change or be removed in future releases of Oracle 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.

7.6.4 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 given 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.

7.6.5 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 the 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.

7.6.6 Configuration File Example

Configuration file example entries:

```
<?cfg jcharset="Cp1252"?>
#Oracle 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
```

This is sufficient to launch nearly all Content Server applications. Others, such as Oracle 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 the Oracle Content Server instance, *IntradocDir*, and alter the `JDBC_JAVA_CLASSPATH_customjdbc` entry in the intradoc.cfg file with the location of the driver.

For example, to run Oracle Content Server with the IBM virtual machine on a Windows operating system, the command line would look like this:

```
#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 Oracle 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.
`-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 *service-name* 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 Oracle Content Server, you can specify the Oracle 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 Oracle Inbound Refinery, the last flag is required so 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 Oracle Content Server service without using that Oracle 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.

7.7 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 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

8

Using the COM API for Integration

This chapter describes Microsoft Component Object Model (COM) integration. Oracle 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:

8.1 Introduction to COM Integration

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 Oracle Content Server. Additionally, you can communicate with ODMA-aware applications through a COM interface.

Performance Tip: Calling services from a command line on the local server using the IdcCommandUX ActiveX Command Utility provides faster execution of commands than calling services remotely using the IntradocClient OCX component.

8.2 ActiveX Interface

The IdcCommandUX ActiveX Command Utility is an ActiveX control that enables a program to execute Oracle Content Server services and retrieve file-path information. The control serves as a COM wrapper for the standard IdcCommand services used by Oracle Content Server. IdcCommandUX works with multibyte languages.

Note: A Visual Basic or Visual C++ development environment is required for using IdcCommandUX.

When executing services using the IdcCommandUX ActiveX Command Utility, keep these items in mind:

- IdcCommandUX must be initialized with a valid user and the intradoc.cfg directory.

  Outside of the `init` and `connection` managing methods, all methods use the serialized HDA format for communication.

- IdcCommandUX attempts to establish a connection to a running server.

  If a connection is not made, IdcCommandUX fails.

- The returned serialized HDA format string contains information about the success or failure of the command.

  The `StatusCode` value will be negative if a failure occurs, and the `StatusMessage` value will indicate the error.

The following subsections describe how to configure and use IdcCommandUX:

8.2.1 Setting Up IdcCommandUX

To set up IdcCommandUX, run the IdcCommandUX setup file, which is stored in extras/IdcCommandUX/setup.exe in the media.

8.2.2 Calling IdcCommandUX from a Visual Basic Environment

To call IdcCommandUX from a Visual Basic environment:

1. Add IdcCommandUX as a control to the Visual Basic project.
2. Create the control as follows:

   ```
   Set idcCmd=CreateObject("Idc.CommandUX")
   ```

3. Define and initialize the connection by calling the `init` (deprecated) function and defining the *UserName* and *DomainDir* parameters:

   ```
   Dim idcCmd
   idcCmd.init("UserName", "DomainDir")
   ```

   - The *UserName* parameter specifies a user that has permission to execute the services being called by IdcCommandUX.
   - The *DomainDir* parameter specifies the complete path to the Oracle Content Server directory that contains the intradoc.cfg configuration file.

Example:

```
Dim idcCmd
idcCmd.initRemote("sysadmin", "c:\domain\bin")
```

8.2.3 Calling IdcCommandUX from a Visual C++ Environment

To call IdcCommandUX from a Visual C++ environment:

1. Add the IdcCommandUX control to the project.
2. Call the desired IdcCommandUX class.

8.2.4 Executing Services

When executing services using IdcCommandUX, keep these points in mind:

- IdcCommandUX must be initialized with a valid user name and the location of the intradoc.cfg file.
- Functions that must use HDA format for communication include computeWebFilePath, computeNativeFilePath, and computeURL. For more information about HDA formats, see Chapter 3, "Working with Standard, Server, and Custom Components."
- executeCommand can take HDA format or SOAP commands. To use SOAP, you must use the initRemote function instead of the init (deprecated) function.
- IdcCommandUX attempts to establish a connection to a running Oracle Content Server instance. If a connection is not made, it fails.
- The returned HDA-format string contains information about the success or failure of the command, using the `StatusCode` and `StatusMessage` variables.
  - If the command is successful, `StatusCode` is zero (0), and `StatusMessage` is a login message ("You are logged in as sysadmin").
  - If the command fails, `StatusCode` is negative (-1), and `StatusMessage` is an error message.

  For more information, see the *Oracle Fusion Middleware Idoc Script Reference Guide*.

  For information about using the Launcher (a native C++ application that allows a Java program to start as a Windows service), see Section 7.6, "Using the Launcher."

8.2.5 Calling IdcCommandUX from an Active Server Page (ASP)

Calling IdcCommandUX from an Active Server Page (ASP) consists of these steps:

1. Creating the COM Object
2. Initializing the Connection
3. Defining Services and Parameters
4. Referencing Custom Resources
5. Executing the Service
6. Retrieving Results

The following examples show how to do these steps.

Example 8–1 SOAP Example

In this SOAP sample:

- The GET_SEARCH_RESULTS service is called.
- The parameters for the service are defined using field/value pairs:
  - The *ResultCount* parameter sets the number of returned results to 5.
  - The *SortField* parameter sorts the returned results by release date.
  - The *SortOrder* parameter orders the returned results in descending order.
  - The *QueryText* parameter defines the query expression as "Content Type matches *research*."

The initRemote function must be used and `isSOAP` must be set to TRUE for a SOAP-formatted request, which is shown in the following example.

```
' Create COM object
Set idcCmd = CreateObject("Idc.CommandUX")
' Initialize the connection to the server
x = idcCmd.initRemote("/domain/ ", "sysadmin",
"socket:localhost:4444", true)
' Create the SOAP envelope
cmd = cmd & "<?xml version='1.0' ecoding='UTF-8'?>" + Chr(10)
cmd = cmd & "<SOAP-ENV:Envelope xmlns:SOAP-ENV=""http://
schemas.xmlsoap.org/soap/envelope/"">" + Chr(10)
cmd = cmd & "<SOAP-ENV:Body>" + Chr(10)
' Define the service
cmd = cmd & "<idc:service xmlns:idc=""http://www.oracle.com/
IdcService/""" + Chr(10)
cmd = cmd & "IdcService=""GET_SEARCH_RESULTS"">" + Chr(10)
' Define the service parameters
cmd = cmd & "<idc:document>" + Chr(10)
cmd = cmd & "<idc:field name=""NoHttpHeaders"">1</idc:field>" +
Chr(10)
cmd = cmd & "<idc:field name=""ClientEncoding"">UTF8</idc:field>"
+ Chr(10)
cmd = cmd & "<idc:field name=""QueryText"">dDocType
&lt;matches&gt; research</idc:field>" + Chr(10)
cmd = cmd & "<idc:field name=""ResultCount"">5</idc:field>" +
Chr(10)
cmd = cmd & "<idc:field name=""SortOrder"">Desc</idc:field>" +
Chr(10)
cmd = cmd & "<idc:field name=""SortField"">dInDate</idc:field>" +
Chr(10)
cmd = cmd & "</idc:document>" + Chr(10)
cmd = cmd & "</idc:service>" + Chr(10)
cmd = cmd & "</SOAP-ENV:Body>" + Chr(10)
cmd = cmd & "</SOAP-ENV:Envelope>" + Chr(10)
' End SOAP envelope and execute the command
results= idcCmd.executeCommand(cmd)
' Retrieve results
Response.Write(results)
```

Example 8–2 HDA Sample

```
' Create COM object
Set idcCmd = CreateObject("Idc.CommandUX")
' Initialize the connection to the server
x = idcCmd.initRemote("/domain/", "socket:localhost:4444", "sysadmin", true)
```

```
' Define the service
cmd = "@Properties LocalData" + Chr(10)
cmd = cmd + "IdcService=GET_SEARCH_RESULTS" + Chr(10)
' Define the service parameters
cmd = cmd + "ResultCount=5" + Chr(10)
cmd = cmd + "SortField=dInDate" + Chr(10)
cmd = cmd + "SortOrder=Desc" + Chr(10)
cmd = cmd + "QueryText=dDocType=research" + Chr(10)
' Reference a custom component
cmd = cmd + "MergeInclude=ASP_SearchResults" + Chr(10)
cmd = cmd + "ClassStyle=home-spotlight" + Chr(10)
cmd = cmd + "@end" + Chr(10)
' Execute the command
results = idcCmd.executeCommand(cmd)
' Retrieve results
Response.Write(results)

' Create COM object
Set idcCmd = CreateObject("Idc.CommandUX")
```

Example 8–3 Creating the COM Object

The first line of code creates the COM object:

```
' Create COM object
Set idcCmd = CreateObject("Idc.CommandUX")
```

Example 8–4 Initializing the Connection

To initialize the connection to Oracle Content Server, call the initRemote function:

```
' Initialize the connection to the server
x = idcCmd.initRemote("/domain/", "socket:localhost:4444", "sysadmin", false)
```

This example uses these parameters:

- The `HttpWebRoot` parameter specifies a value for the web root as defined in the config/config.cfg file.
- The `idcReference` parameter specifies a string containing information about connection to the Oracle Content Server instance. This is specified as "socket" followed by the IntradocServerHostName value and the IntradocServer Port address.
- The value of the`idcUser` parameter, `"sysadmin"`, specifies the user who is connecting to Oracle Content Server.
- The `isSoap` parameter is a Boolean value indicating if the request is in SOAP XML format or HDA format. In this case, it is `false` because it is in HDA format.

For information about all the parameters, see Section 8.3.11, "initRemote."

Example 8–5 Defining Services and Parameters

To define the service and parameters, build an HDA-formatted string that contains with the following lines:

```
@Properties LocalData
service
parameters
@end
```

The required and optional parameters vary depending on the service being called. For more information, see the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management*.

In this example, the `@end` string is created after the optional custom component reference. For more information, see Section 8.2.6, "Formatting with a Resource Include."

Example 8–6 Referencing Custom Resources

You can reference custom resources and pass parameters to a resource include from your ASP as follows:

- To reference a custom resource include, set the `MergeInclude` parameter to the name of the include.

  In this example, the `ASP_SearchResults` include is used to format the output as HTML rather than a ResultSet. For more information, see Section 8.2.6, "Formatting with a Resource Include."

- To pass a parameter to a resource include, set the variable as name/value pair.

  In this example, the `ClassStyle` variable with a value of *home-spotlight* is available to the `ASP_SearchResults` include.

> **Note:** The `@end` code is required to close the `@Properties LocalData` section in an HDA-formatted string. For more information, see Section 8–5, "Defining Services and Parameters."

```
' Reference a custom component
cmd = cmd + "MergeInclude=ASP_SearchResults" + Chr(10)
cmd = cmd + "ClassStyle=home-spotlight" + Chr(10)
cmd = cmd + "@end" + Chr(10)
```

Example 8–7 Executing the Service

To execute the service, call the executeCommand method.

After executing the service, you could use the closeServerConnection method to make sure that the connection is closed.

```
' Execute the service
results = idcCmd.executeCommand(cmd)
```

Example 8–8 Retrieving Results

The results can either be formatted HTML or a ResultSet.

In this example, the result of the service call is formatted HTML.

```
' Retrieve results
Response.Write(results)
```

8.2.6 Formatting with a Resource Include

This section provides an example of a custom resource include that is used to format the output of a service executed by IdcCommandUX.

In the example described in Section 8.2.5, "Calling IdcCommandUX from an Active Server Page (ASP),", the *ASP_SearchResults* resource include is used to format the output of a search function and return HTML rather than a ResultSet:

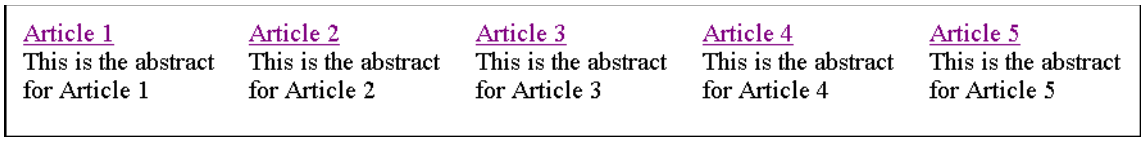
```
<@dynamichtml ASP_SearchResults@>
<table border=0>
    <$loop SearchResults$>
    <tr class="site-default">
    <td class="<$ClassStyle$>">
    <a href="<$URL$>" target=new><$dDocTitle$></a><br>
    <$xAbstract$>
    </td>
    </tr>
    <$endloop$>
</table>
<@end@>
```

- The `<@dynamichtml ASP_SearchResults@>` entry defines the name of the resource include. The `<@end@>` entry ends the resource definition.
- The code defined between the `<$loop SearchResults$>` and `<$endloop$>` entries is executed for each content item in the SearchResults ResultSet, which includes all documents that matched the query defined for the GET_SEARCH_RESULTS service.
- The `<td class="<$ClassStyle$>">` entry displays the value of the `<$ClassStyle$>` Idoc Script variable. In this example, the `ClassStyle` value was passed in on the API call.
- The `<a href="<$URL$>" target=new><$dDocTitle$></a>` entry displays the Title of the current content item as a link to the file.
- The `<$xAbstract$>` entry displays the Abstract value for the current content item.

The HTML generated and returned to the Active Server Page from this resource include would have this format:

```
<table border=0>
<tr class="site-default">
<td class="home-spotlight">
<a href="/domain/dir/dir/xyz.htm"  target=new>Article 1</a><br>
This is the abstract for Article 1
</td>
<td class="home-spotlight">
<a href="/domain/dir/dir/xyz.htm"  target=new>Article 2</a><br>
This is the abstract for Article 2
</td>
<td class="home-spotlight">
<a href="/domain/dir/dir/xyz.htm"  target=new>Article 3</a><br>
This is the abstract for Article 3
</td>
<td class="home-spotlight">
<a href="/domain/dir/dir/xyz.htm"  target=new>Article 4</a><br>
This is the abstract for Article 4
</td>
<td class="home-spotlight">
<a href="/domain/dir/dir/xyz.htm"  target=new>Article 5</a><br>
This is the abstract for Article 5
</td>
</tr>
</table>
```

Displaying this HTML page in a browser would look like the following example.



8.2.7 Connecting to Oracle Content Server from a Remote System

This section describes how to establish a connection to an Oracle Content Server instance from a remote system using IdcCommandUX from an Active Server Page. These steps are required:

1. Creating Variables
2. Creating a COM Object
3. Initializing the Connection
4. Returning the Connection Status
5. Defining the Service and Parameters
6. Executing the Service
7. Retrieving Results

The following examples show how to do these steps.

Example 8–9 Coding the ASP Page

This example calls the CHECKIN_UNIVERSAL service to provide a check-in function from a remote system. This code does not check for an error condition.

```
' Create variables
Dim idccommand, sConnect, str
' Create COM object
Set idccommand = Server.CreateObject("idc.CommandUX")
' Initialize the connection to the server
x = idccommand.initRemote ("/domain/ ", "sysadmin", "socket:localhost:4444",
false)
' Return connection status (optional)
sConnect = idccommand.connectToServer
if sConnect then
Response.Write "Connected"
else
Response.Write "Not Connected"
end if
str = "@Properties LocalData" & vbcrlf
' Define the service
str = str + "IdcService=" & "CHECKIN_UNIVERSAL" & vbcrlf
' Define the service parameters
str = str + "doFileCopy=1" & vbcrlf
str = str + "dDocName=RemoteTestCheckin23" & vbcrlf
str = str + "dDocTitle=Test1" & vbcrlf
str = str + "dDocType=ADACCT" & vbcrlf
str = str + "dSecurityGroup=Public" & vbcrlf
str = str + "dDocAuthor=sysadmin" & vbcrlf
str = str + "dDocAccount=" & vbcrlf
str = str + "primaryFile:path=C:/inetpub/Scripts/query2.asp" & vbcrlf
str = str + "@end" & vbcrlf
```

```
' Execute the command
res=idccommand.executeCommand(str)
' Return connection status
sClosed = idcCmd.closeServerConnection
if sClosed then
Response.Write "Server connection closed"
else
Response.Write "Failed to close server connection"
end if
' Retrieve results
Response.Write(res)
```

Example 8–10 Creating Variables

The following variables must be created:

- **idccommand**: The name of the COM object.
- **sConnect**: The status of the connection to the Oracle Content Server instance.
- **str**: The HDA-formatted string that defines the service and its parameters.

```
' Create variables
Dim idccommand, sConnect, str
```

Example 8–11 Creating a COM Object

The following variables must be created:

- **idccommand**: The name of the COM object.
- **sConnect**: The status of the connection to the Oracle Content Server instance.
- **str**: The HDA-formatted string that defines the service and its parameters.

```
' Create variables
Dim idccommand, sConnect, str
```

Example 8–12 Initializing the Connection

Initialize the connection to the Oracle Content Server instance.

```
' Initialize the connection to the server
x = idccommand.initRemote ("/domain/ ", "sysadmin", "socket:localhost:4444", false)
```

Example 8–13 Returning the Connection Status

In this example, the connectToServer and closeServerConnection methods are used to return connection status information before and after the service is executed.

```
' Return connection status
sConnect = idccommand.connectToServer
if sConnect then
Response.Write "Connected"
else
Response.Write "Not Connected"
end if
...
' Return connection status
sClosed = idcCmd.closeServerConnection
if sClosed then
Response.Write "Server connection closed"
else
Response.Write "Failed to close server connection"
end if
```

Example 8–14 Defining the Service and Parameters

To define the service and parameters, build an HDA-formatted string that contains the following lines:

```
@Properties LocalData
service
parameters
@end
```

The required and optional parameters vary depending on the service being called. For more information, see the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management*.

In this example:

- The CHECKIN_UNIVERSAL service is called.
- The parameters for the service are defined using field/value pairs:
  - The `doFileCopy` parameter is set to TRUE (1), so the file will not be deleted from hard drive after successful check in.
  - The `dDocName` parameter defines the Content ID.
  - The `dDocTitle` parameter defines the Title.
  - The `dDocType` parameter defines the Type.
  - The `dSecurityGroup` parameter defines the Security Group.
  - The `dDocAuthor` parameter defines the Author.
  - The `dDocAccount` parameter defines the security account. (If accounts are enabled, this parameter is required.)
  - The `primaryFile` parameter defines original name for the file and the absolute path to the location of the file as seen from the server.

Important: The required parameters vary depending on the service called. For more information, see the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management*.

```
str = "@Properties LocalData" & vbcrlf
' Define the service
str = str + "IdcService=" & "CHECKIN_UNIVERSAL" & vbcrlf
' Define the service parameters
str = str + "doFileCopy=1" & vbcrlf
str = str + "dDocName=RemoteTestCheckin23" & vbcrlf
str = str + "dDocTitle=Test1" & vbcrlf
str = str + "dDocType=ADACCT" & vbcrlf
str = str + "dSecurityGroup=Public" & vbcrlf
str = str + "dDocAuthor=sysadmin" & vbcrlf
str = str + "dDocAccount=" & vbcrlf
str = str + "primaryFile:path=C:/inetpub/Scripts/query2.asp" & vbcrlf
str = str + "@end" & vbcrlf
```

Example 8–15 Executing the Service

To execute the service, call the executeCommand method.

```
' Execute the service
res=idccommand.executeCommand(str)
```

Example 8–16 Retrieving Results

In this example, the result of the CHECKIN_UNIVERSAL service call is formatted HTML.

```
' Retrieve results
Response.Write(res)
```

8.3 IdcCommandUX Methods

The following subsections describe the IdcCommandUX methods:

Important: All parameters are required unless otherwise indicated.

8.3.1 addExtraheadersForCommand

This command adds extra HTTP-like headers to a command.

- For security reasons, some parameters can only be passed in the headers.
- The most common use for this command is to set the values for `EXTERNAL_ROLES` and `EXTERNAL_ACCOUNTS` in a request.
- Values must be all on one string and separated by a carriage return and a line feed.

Example

The following is an ASP example:

```
extraHeaders = "EXTERNAL_ROLES=contributor" _
    + vbcrlf _
    + "EXTERNAL_ACCOUNTS=my_account"
idcCmd.addExtraHeadersForCommand(extraHeaders)
```

8.3.2 closeServerConnection

```
Public Sub closeServerConnection()
```

Description

Closes the server connection.

- This method does not have to be called, because the executeCommand method automatically closes a connection after executing a service. It is provided only as a convenience for managing the state of the connection.

Parameters

None

Output

- Returns TRUE if the connection is closed.
- Returns FALSE if the connection failed to close.

Example

This ASP example passes the result of the `closeServerConnection` method to a variable and uses an if/else statement to return a connection status message:

```
sClosed = idcCmd.closeServerConnection
if sClosed then
Response.Write "Server connection closed"
else
Response.Write "Failed to close server connection"
end if
```

8.3.3 computeNativeFilePath

```
Public Function computeNativeFilePath(Data As String) As String
```

Description

HDA-only function.

Returns the URL of a native file as a string.

- This function is generally used for processing native files to perform actions such as bulk file loading or retrieval.
- To determine the values for the required parameters (such as *dDocType* and *dID*), you can reference the ResultSet returned from a DOC_INFO or SEARCH_RESULTS service call.
  - The DOC_INFO service can be used to specify previous revisions (DOC_INFO returns a list of previous revision labels).
  - The SEARCH_RESULTS service returns only enough data to specify the most recent revision of a content item.

Parameters

- Data: An HDA-formatted string that defines the content item:
  - **dDocType**: The content item Type, such as `ADACCT` or `FILES`.
  - **dID**: The generated content item revision ID.

- **dExtension**: The file extension, such as HCSF, DOC, or TXT.
- **dDocAccount**: The account for the content item. If accounts are enabled, this parameter must be defined.

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 revision of a content item.

Output

- Returns a string that defines *NativeFilePath* as the value of the string passed in as a parameter. For example:

  ```
  NativeFilePath=c:\domain\vault\adacct\1.doc
  ```

- Returns an HDA string containing `StatusCode` and `StatusMessage`.
  - If the command is successful, `StatusCode` is zero (`0`), and `StatusMessage` is a login message (`"You are logged in as sysadmin"`).
  - If the command fails, `StatusCode` is negative (`-1`), and `StatusMessage` is an error message.
  - Returns `FALSE` if there is a connection failure.

Example

This is an example of an HDA-formatted string:

```
String str = "@Properties LocalData\n"+
"dDocType=ADACCT\n"+
"dID=67\n"+
"dExtension=DOC\n"+
"dDocAccount=mainaccount\n"+
"@end\n";
```

8.3.4 computeURL

```
Public Function computeURL(Data As String, IsAbsolute As Boolean) As String
```

Description

HDA-only function.

Returns the URL of a content item as a string.

- A relative or absolute URL can be supplied to Oracle Content Server.
  - When a relative URL is defined, the function evaluates the URL as a location valid on the local server.
  - For example:

    ```
    /domain/groups/Public/documents/FILE/doc.txt
    ```

  - When an absolute URL is defined, the function returns the absolute URL path.
  - For example:

    ```
    http://server/domain/groups/Public/documents/FILE/doc.txt
    ```

- To determine the values for the Oracle Content Server parameters (`HttpRelativeWebRoot` and `HttpServerAddress`), you can reference the properties data returned from a GET_DOC_CONFIG_INFO service call.
- To determine the values for the required content item parameters (such as *dSecurityGroup* and *dDocType*), you can reference the ResultSet returned from a DOC_INFO or SEARCH_RESULTS service call.
  - The DOC_INFO service can be used to specify previous revisions (DOC_INFO returns a list of previous revision labels).
  - The SEARCH_RESULTS service returns only enough data to specify the most recent revision of a content item.
- To return the URL for a specific revision and rendition, use the content item revision label (*dRevLabel*) and the file extension (*dWebExtension*) entries. For example:

```
dDocName=test10
dRevLabel=2
dWebExtension=pdf
```

- To return the URL for the most recent revision, the content item revision label (*dRevLabel*) entry can be omitted. For example, defining just the Content ID (*dDocName*) and the file extension (*dWebExtension*) returns the most recent revision:

```
dDocName=test11
dWebExtension=html
```

Parameters

- Data: An HDA-formatted string that defines the content item:
  - **HttpRelativeWebRoot**: The web root directory as a relative path, such as `/stellent/`. This entry is required for a relative URL, and is optional for an absolute URL.
  - **HttpServerAddress**: The domain name of the Oracle Content Server instance, such as `testserver17` or `example.com`. (The server address is specified as a partial URL, such as `example.com`, rather than a full address, such as `http://www.example.com/`). This entry is required for an absolute URL, and it is optional for a relative URL.
  - **dSecurityGroup**: The security group, such as *Public* or *Secure*.
  - **dDocType**: The Type, such as *ADACCT* or *FILES*.
  - **dDocName**: The Content ID, such as *test10* or *hr_0005467*.
  - **dWebExtension**: The file extension of the web-viewable file, such as *xml*, *html*, or *txt*.
  - **dDocAccount**: The account for the content item. If accounts are enabled, this parameter must be defined.
  - **dRevLabel** (optional): The revision label for the content item. If defined, the specific revision will be referenced.

- IsAbsolute: Set to TRUE (1) to define an absolute URL address.

> **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 revision of a content item.

Output

- Returns a string that defines *URL* as the value of the string passed in as a parameter. For example:

  ```
  URL=http://server/domain/groups/public/documents/FILE/doc.txt
  ```

- Returns an HDA string containing `StatusCode` and `StatusMessage`.
  - If the command is successful, `StatusCode` is zero (0), and `StatusMessage` is a login message ("You are logged in as sysadmin").
  - If the command fails, `StatusCode` is negative (-1), and `StatusMessage` is an error message.
  - Returns FALSE if there is a connection failure.

Example

This is an example of an HDA-formatted string:

```
String str = "@Properties LocalData\n"+
"HttpServerAddress=testserver17\n"+
"HttpRelativeWebRoot=/domain/\n"+
"dDocAccount=mainaccount\n"+
"dSecurityGroup=Public\n"+
"dDocType=ADACCT\n"+
"dDocName=test11\n"+
"dWebExtension=html\n"+
"@end\n";
```

8.3.5 computeWebFilePath

```
Public Function computeWebFilePath(Data As String) As String
```

Description

HDA-only function.

Returns the path of a web-viewable file as a string.

- This function is generally used for processing web-viewable text files (such as XML) to perform actions such as bulk file loading or retrieval.
- Using `computeWebFilePath` instead of computeNativeFilePath provides the advantage of needing only the Content ID (*dDocName*) rather than the specific revision ID (*dID*) to return the most recent revision.
- To determine the values for the required parameters (such as *dSecurityGroup* and *dDocType*), you can reference the ResultSet returned from a DOC_INFO or SEARCH_RESULTS service call.
  - The DOC_INFO service can be used to specify previous revisions (DOC_INFO returns a list of previous revision labels).
  - The SEARCH_RESULTS service returns only enough data to specify the most recent revision of a content item.

Parameters

- Data: An HDA-formatted string that defines the content item:
  - **dSecurityGroup**: The security group, such as *Public* or *Secure*.
  - **dDocType**: The content item Type, such as *ADACCT* or *FILES*.
  - **dDocName**: The Content ID, such as *test10* or *hr_0005467*.
  - **dWebExtension**: The file extension of the web-viewable file, such as *xml*, *html*, or *txt*.
  - **dDocAccount**: The account for the content item. If accounts are enabled, this parameter must be defined.

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 revision of a content item.

Output

- Returns a string that defines *WebFilePath* as the value of the string passed in as a parameter. For example:

```
WebFilePath=http:\\testserver17.example.com\domain\groups\main\documents\test.xml
```

- Returns an HDA string containing `StatusCode` and `StatusMessage`.
  - If the command is successful, `StatusCode` is zero (0), and `StatusMessage` is a login message ("You are logged in as sysadmin").
  - If the command fails, `StatusCode` is negative (-1), and `StatusMessage` is an error message.
  - Returns FALSE if there is a connection failure.

Example

This is an example of an HDA-formatted string:

```
String str = "@Properties LocalData\n"+
"dDocAccount=mainaccount\n"+
"dSecurityGroup=Public\n"+
"dDocType=ADACCT\n"+
"dDocName=test11\n"+
"dWebExtension=xml\n"+
"@end\n";
```

8.3.6 connectToServer

```
Public Function connectToServer() As Boolean
```

Description

Establishes a connection to the server.

- The connection is held open until a command is executed. After a command is executed, the connection is closed automatically.
- This method does not have to be called, because the executeCommand method automatically opens a connection to execute a service. It is provided only as a convenience for managing the state of the connection.

Parameters

None

Output

- Returns TRUE if the connection is opened.
- Returns FALSE if there is a connection failure.

Example

This ASP example passes the result of the *connectToServer* method to a variable and uses an if/else statement to return a connection status message:

```
sConnect = idcCmd.connectToServer
if sConnect then
Response.Write "Connected"
else
Response.Write "Not Connected"
end if
```

8.3.7 executeCommand

```
Public Sub executeCommand(Data As String)
```

Description

Executes an Oracle Content Server service.

- This method evaluates whether a connection has already been established with a *connectToServer* call. If a connection exists, it will use the open connection. If a connection does not exist, it will establish a connection.
- On completion of the command, the connection will be closed.

Parameters

- Data: An HDA-formatted string that defines the `IdcService` command and any service parameters. For example:

  ```
  @Properties LocalData
  IdcService=GET_SEARCH_RESULTS
  ResultCount=5
  SortField=dInDate
  SortOrder=Desc
  QueryText=dDocType=research
  @end@
  ```

  This can also be a SOAP-formatted message, as shown in Example 8–1. For more information, see Section 8.3.11, "initRemote."

Output

- Returns a string representing an HDA file that holds the original request and the results.
- Returns an HDA string containing `StatusCode` and `StatusMessage`.
  - If the command is successful, `StatusCode` is zero (0), and `StatusMessage` is a login message ("You are logged in as sysadmin").
  - If the command fails, `StatusCode` is negative (-1), and `StatusMessage` is an error message.
  - Returns FALSE if there is a connection failure.
- The return string is SOAP-formatted XML if a SOAP request was sent.

Example

This ASP example executes the command specified in the data string defined by the *cmd* variable:

```
results = idcCmd.executeCommand(cmd)
```

8.3.8 executeFileCommand

```
executeFileCommand (requestString)
```

Description

This function is used to execute a service request, then pipe the raw response to the client. This command is identical to `executeCommand` but can only be called on an Active Server Page (ASP).

- The response from Oracle Content Server is redirected back to the client's browser (this is different from the response through `executeCommand`, in which the response is given as a string which can then be manipulated on the ASP).
- This is useful for GET_FILE and similar services in which you need to transfer binary files from Oracle Content Server to a client browser through an ASP.
- This function returns extra headers unless the request parameters are passed as environment variables.
- *requestString* is the name of the service request.
- For more information, see Section 8.3.7, "executeCommand."

Parameters

None

8.3.9 forwardRequest

```
forwardRequest()
```

Description

This function is used to forward a multipart form post to Oracle Content Server. This is useful for executing check-ins.

Parameters

None

8.3.10 getLastErrorMessage

```
getLastErrorMessage()
```

Description

This method retrieves the specific error details for a communication or configuration error. For example, if you do not put in the correct hostname for making a connection, this method returns the connection error. It does not return a value if the error is returned by Oracle Content Server as part of the return value for a request.

Parameters

None

Example

This example creates an object and initializes a connection to the server.

```
Set idcCmd = Server.CreateObject("Idc.CommandUX")

x = idcCmd.init("sysadmin", "c:\domain\bin")
If x = false Then
y = idcCmd.getLastErrorMessage()
Response.Write(y)
End If
```

8.3.11 initRemote

```
initRemote(HttpWebRoot, idcReference, idcUser, isSoap)
```

Description

This function initializes the module to connect to an Oracle Content Server instance. Note that you must first declare `idcCmd`.

Required Parameters

- **HttpWebRoot**: The Idoc Script value for `HttpWebRoot`.
- **idcReference**: A string containing information about how to connect to the Oracle Content Server instance, in the form `socket:hostname:port`. This is typically `socket:localhost:4444`. The `hostname` should be identical to *IntradocServerHostName* and `port` identical to *IntradocServerPort*.
- **idcUser**: The user you are connecting as.
- **isSoap**: A Boolean value indicating if the request is in SOAP XML format or HDA format. If this is set to TRUE, it indicates the SOAP XML format.

Example

```
Dim idcCmd
idcCmd.initRemote("domain", "socket:test204:4444", "sysadmin", "false")
```

8.4 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 Oracle Content Server. The OCX integration is designed to call services in a visual development environment, or to connect to a remote Oracle 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 Guide for Universal Content Management*. This guide also contains information about the IntradocClient OCX API specifications listing the properties, methods, and events.

8.5 IdcClientOCX Component

An Object Linking and Embedding Control Extension (OCX) control is provided for connecting to a remote Oracle Content Server instance and executing Oracle 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 Oracle Content Server.

This section provides a description of the IdcClient OCX control, setup instructions, and lists the events, methods, and properties. The `IdcClient.ocx` control is used to connect to a remote Oracle Content Server instance and perform typical server functions.

The following subsections describe the IdcClientOCX component and how to set it up:

- Section 8.5.1, "IdcClient OCX Description"
- Section 8.5.2, "IdcClient OCX Control Setup"

Note: A Visual Basic or Visual C++ development environment is required for using the IdcClient OCX component.

8.5.1 IdcClient OCX Description

This section provides a general description of the IdcClient OCX control and basic information about events, methods, and properties. The IdcClient OCX interface is also discussed.

8.5.1.1 General 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 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 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.

For more information, see *Oracle Fusion Middleware Services Reference Guide for Universal Content Management*.

8.5.1.2 Events, Methods, and Properties

The IdcClient OCX control is used to connect to a remote Oracle Content Server instance and perform server functions. This section provides a basic overview on Visual Basic events, methods, and properties.

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.

  For more information, see Section 8.5.1.2, "Events, Methods, and Properties."

Example 8–17 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.

For more information, see Section 8.5.1.2, "Events, Methods, and Properties."

Example 8–18 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.

For more information, see Section 8.5.1.2, "Events, Methods, and Properties."

8.5.1.3 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 Oracle Content Server. The OCX integration is designed to call services in a visual development environment, or to connect to a remote Oracle 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.

8.5.2 IdcClient OCX Control Setup

This section provides a the steps required to setup the IdcClient OCX component and also provides information about creating a visual interface in the Microsoft Visual Basic development environment.

8.5.2.1 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 8.5.2.2, "Creating a Visual Interface," to build a basic visual interface.

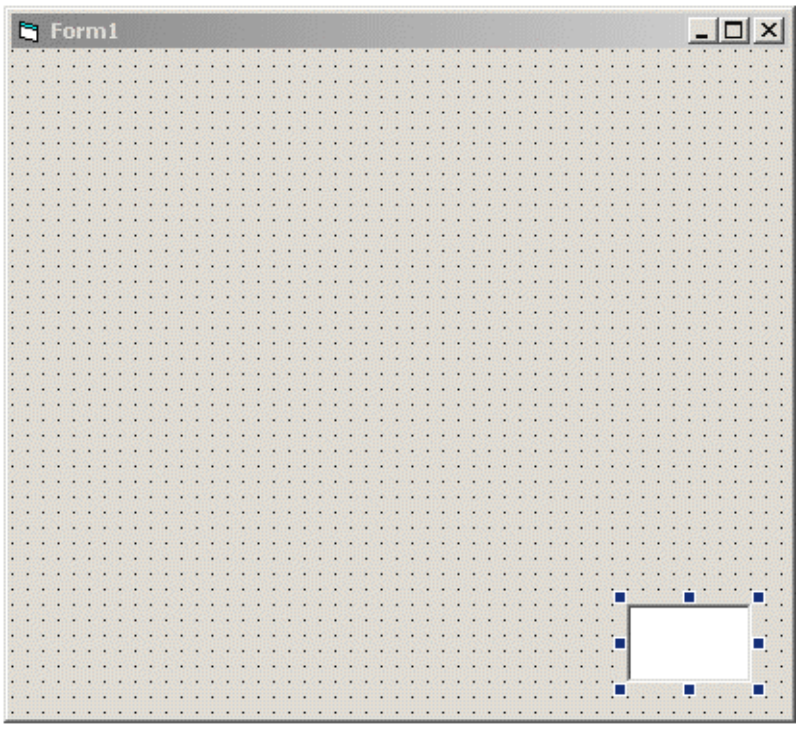
8.5.2.2 Creating a Visual Interface

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 8.5.2.1, "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, as Figure 8–1 shows.

Figure 8–1 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.

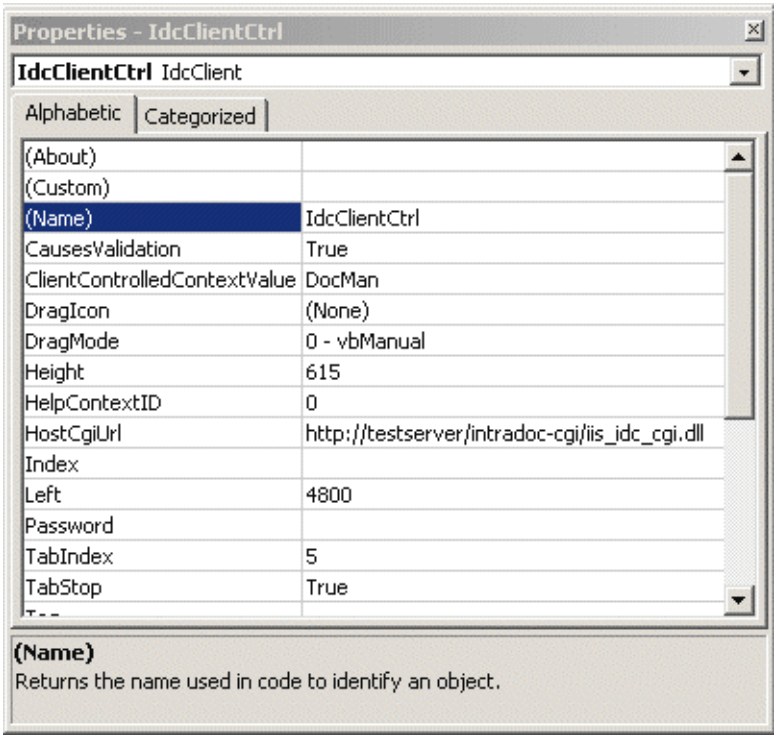
3. Rename the **IdcClient OCX** control **IdcClientCtrl**.

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 8–2 Edited IdcClient Properties

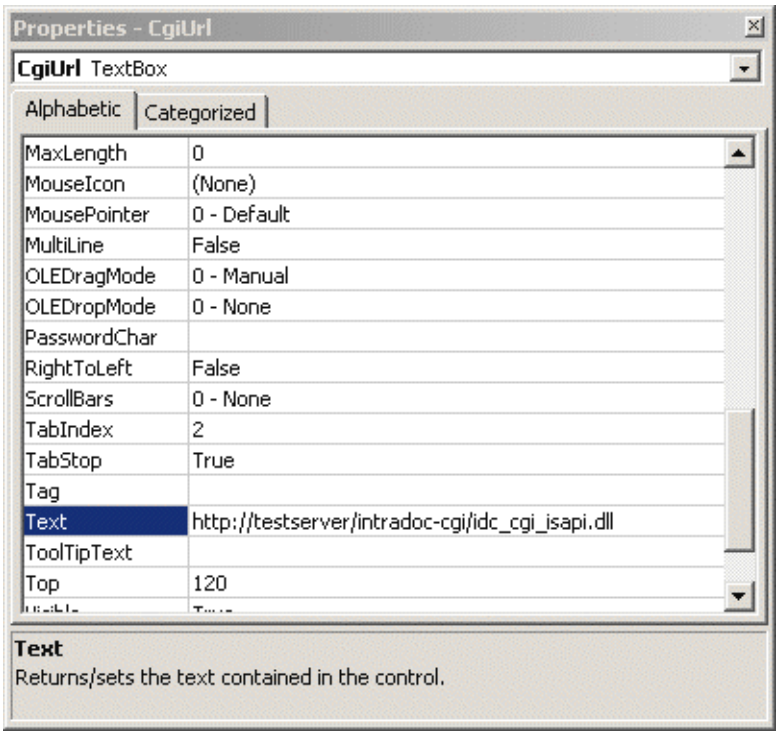


5. On the form, draw a text box, and name it `CgiUrl`.
6. For the text field, enter the **HostCgiUrl** value as the text to be displayed.

   For example:

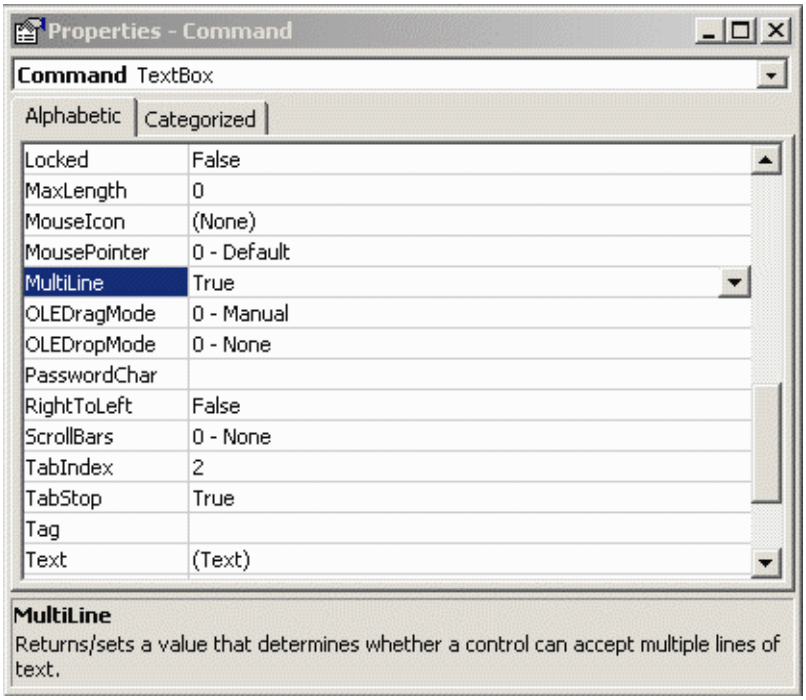
   ```
   http://testserver/intradoc-cgi/iss_idc_cgi.dll
   ```

Figure 8–3 Edited CgiUrl TextBox Properties



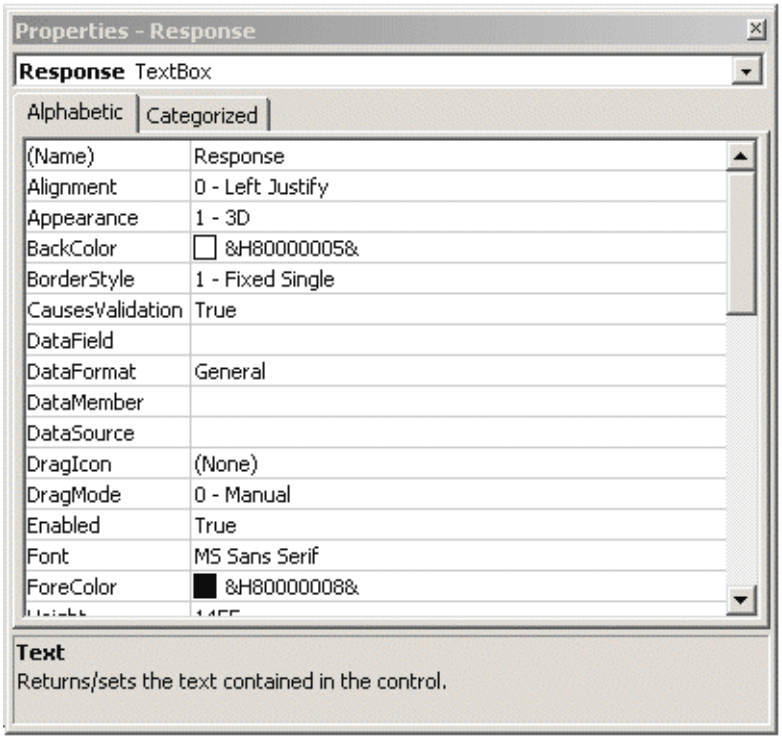
7. On the form, draw a text box, and name it `Command`.
8. Clear the entry for the text field (leave blank), and set **MultiLine** to `True`.

Figure 8–4 Edited Command TextBox Properties



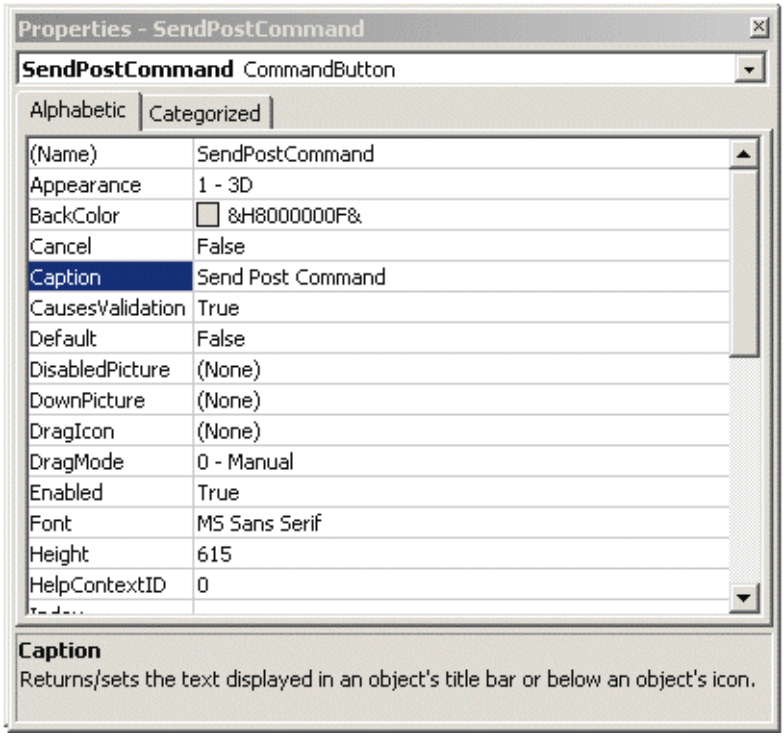
9. On the form, draw a text box, and name it `Response`.
10. Clear the entry for the text field (leave blank).

Figure 8–5 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 8–6 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.

    For example:

```
Dim R As String
IdcClientCtrl.HostCgiUrl = CgiUrl.Text
R = IdcClientCtrl.1.SendPostCommand(Command.Text)
R = Replace(R, vbLf, vbCrLf
Response.Text = R
```

Figure 8–7 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 Content Server instance:

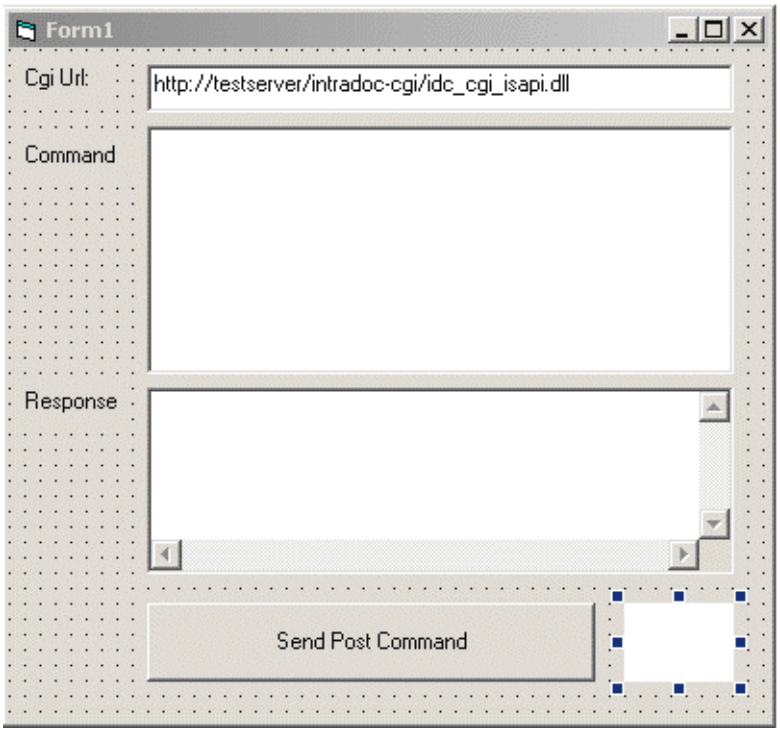
```
IdcClientCtrl.UseBrowserLoginPrompt = True
IdcClientCtrl.UseProgressDialog = True
```

Figure 8–8 Edited Form_Load Code

```
Private Sub Form_Load()
   IdcClientCtrl.UseBrowserLoginPrompt = True
   IdcClientCtrl.UseProgressDialog = True
End Sub
```

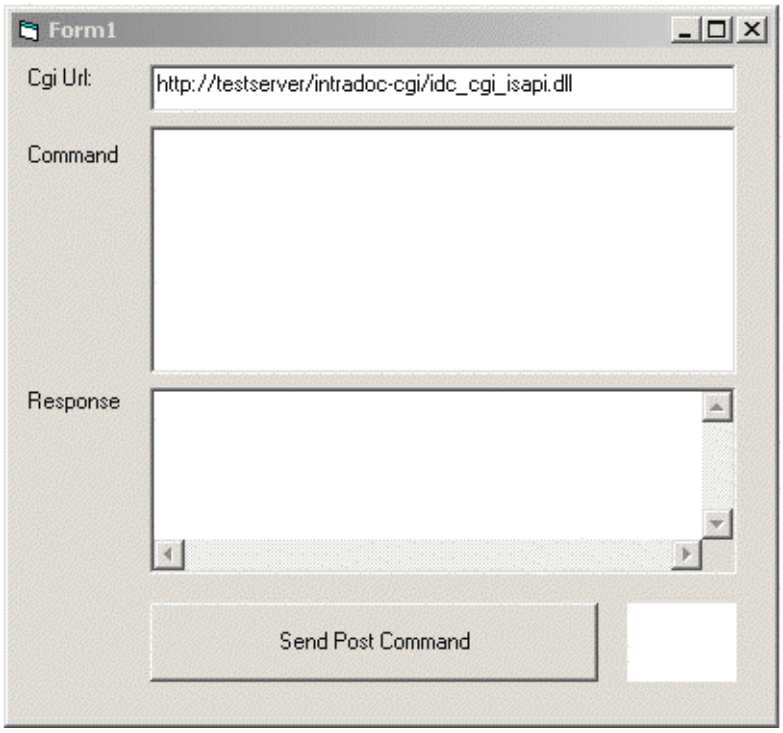
16. (Optional) Add appropriate descriptive labels, such as `Cgi Url`, `Command`, and `Response`.

Figure 8–9 Visual Interface with a Descriptive Label



17. Select **Run**, and then choose **Start** to test the visual interface.

Figure 8–10 Completed Visual Interface



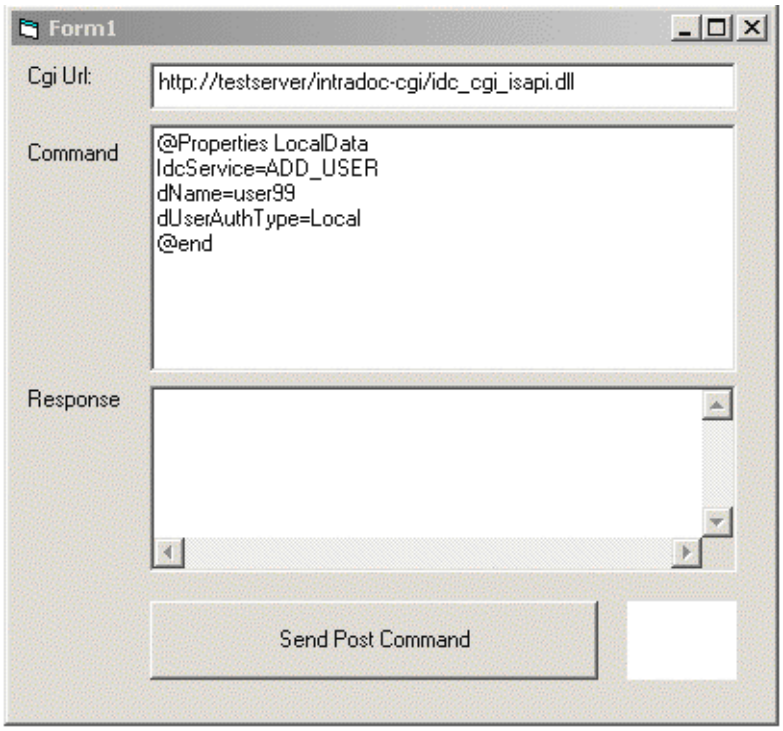
18. Enter a formatted command in the **Command** field.

    For example, this command adds a user:

```
@Properties LocalData
IdcService=ADD_USER
dName=user99
dUserAuthType=Local
@end
```

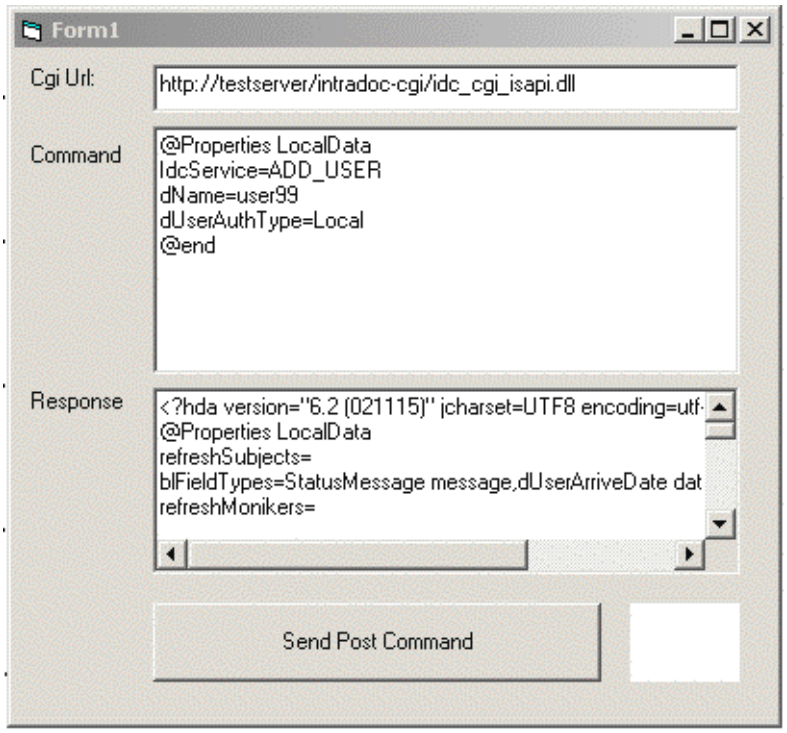
    For more information about the ADD_USER service, see the *Oracle Fusion Middleware Services References Guide*.

Figure 8–11 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 8–12 Visual Interface with Returned Results



Verify the Command

1. In a web browser, log in to Oracle Content Server as an administrator.
2. In the **Administration** tray, select **Admin Applets**.
3. Click **User Admin**. The applet launches and displays the added user (for example, `user99`).

8.6 IdcClient Events

Events are executed when the user or server performs an action. The following subsections describe the IdcClient OCX events:

- Section 8.6.1, "IntradocBeforeDownload"
- Section 8.6.2, "IntradocBrowserPost"
- Section 8.6.3, "IntradocBrowserStateChange"
- Section 8.6.4, "IntradocRequestProgress"
- Section 8.6.5, "IntradocServerResponse"

8.6.1 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

8.6.2 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

8.6.3 IntradocBrowserStateChange

Executes whenever the browser state changes.

Parameters

The event passes these parameters:

- ByVal `browserStateItem` As String
- ByVal `enabled` As Boolean

8.6.4 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

8.6.5 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

8.7 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.

8.7.1 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

8.7.2 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

8.7.3 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.

8.7.4 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 8.6, "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.

8.7.5 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 8.6, "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.

8.7.6 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 8.6, "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.

8.7.7 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.

8.7.8 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*.

8.7.9 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

8.7.10 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.

8.7.11 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

8.7.12 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 8.6, "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.

8.7.13 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 8.6, "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.

8.7.14 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.

8.7.15 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:

- Serialized HDA containing the requested information.
- The data that was passed in as parameters.

Parameters

- url: The complete URL path.

8.7.16 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.

8.7.17 Refresh Browser

Description

Refreshes the browser.

- This method refreshes the web browser and updates dynamic information.

Parameters

None

8.7.18 SendCommand

```
Function SendCommand(params As String) As String
```

Description

Issues a service request to Oracle Content Server.

- Given defined content item parameters, the function executes a service from Oracle 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.

8.7.19 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 8.6, "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.

8.7.20 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

8.7.21 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

8.7.22 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.*

8.7.23 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.

8.7.24 StartSearch

```
Sub StartSearch()
```

Description

Displays the query page in the browser control.

- Preforms browser manipulation.

Parameters

None

8.7.25 Stop

```
Sub Stop()
```

Description

Stops the browser.

- This method stops or cancels the loading of information in the browser.

Parameters

None

8.7.26 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 8.6, "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.*

8.7.27 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.

8.7.28 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:

- 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.

8.7.29 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 are:

- 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.

8.8 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

8.8.1 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 Content Server.

- Returns the value as a string.
- Takes no parameters.

8.8.2 HostCgiUrl

Provides the complete URL path of the host CGI bin.

- Returns the value as a string.
- Takes no parameters.

8.8.3 Password

Provides the assigned user password.

- Returns the value as a string.
- Takes no parameters.

8.8.4 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

8.8.5 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.

8.8.6 UserName

Provides the assigned user name.

Returns the value as a string.

Takes no parameters.

8.8.7 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.

8.9 ODMA Integration

The Open Document Management Application (ODMA) is a standard API used to interface between desktop applications and file management software. The ODMA integration for Oracle 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 Oracle 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.

8.9.1 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 Content Server, for example:

- Save a file and immediately check it in to Oracle Content Server.
- Save a file to check in later.
- Check out a file from Oracle Content Server.
- Update a file's metadata (content information).
- Save the file to your local file system and bypass the ODMA Client system.

8.9.2 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 Oracle 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 it into Oracle Content Server.
  - Select and open a file from the **Recent Files** list or from Oracle Content Server.
- Oracle Content Server **Interface with ODMA**: You can open and check out an ODMA file directly from the Oracle Content Server Content Information page. When you open a file from Oracle Content Server, it opens in its native application so you can edit it and quickly check the file back into Oracle 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.

9

Using Remote Intradoc Client (RIDC)

This chapter describes how to initialize and use Remote Intradoc Client (RIDC), which provides a thin communication API for communication with Oracle Content Server.

This chapter includes the following sections:

9.1 Introduction to RIDC

The RIDC communication API removes data abstractions to Oracle Content Server while still providing a wrapper to handle connection pooling, security, and protocol specifics. If you want to use a native Java API, then RIDC is recommended.

RIDC has these key features:

- Support is provided for Intradoc socket-based communication and the HTTP and JAX-WS protocols.
- Client configuration parameters include setting the socket time outs, connection pool size, and so forth.
- All calls to RIDC require some user identity for authentication. For Intradoc URLs, no credentials are required because the request is trusted between Oracle Content Server and the client. For HTTP URLs, the context requires credentials.
- To invoke a service, you can use the `ServiceRequest` object, which can be obtained from the client.

- The RIDC client pools connections, which requires that the caller of the code close resources when done with a response.
- Streams are sent to Oracle Content Server through the `TransferStream` interface.
- The RIDC objects follow the standard Java Collection paradigms, which makes them extremely easy to consume from a JSP/JSPX page.
- Binders can be reused among multiple requests.
- RIDC allows Secure Socket Layer (SSL) communication with Oracle Content Server.

9.1.1 RIDC Protocols

RIDC supports three protocols: Intradoc, HTTP, and WebServices/JAX-WS.

Intradoc: The Intradoc protocol communicates with Oracle Content Server over the Intradoc socket port (typically, 4444). This protocol does not perform password validation and so requires a trusted connection between the client and Oracle Content Server. Clients that use this protocol are expected to perform any required authentication. Intradoc communication can also be configured to run over SSL.

HTTP: RIDC communicates with the web server for Oracle Content Server using the Apache HttpClient package. Unlike Intradoc, this protocol requires authentication credentials for each request.

For more information, see the Jakarta Commons HttpClient documentation on the HttpClient Home page of the Apache HttpClient web site at

`http://hc.apache.org/httpclient-3.x`

JAX-WS: The JAX-WS protocol is supported only in Oracle UCM 11*g* with Oracle Content Server running in Oracle WebLogic Server. To provide JAX-WS support, several additional JAR archives are required. These JAR archives are provided with the ecm-client.zip distribution, which is available from the Oracle Technology Network (OTN).

These additional JAR archives are required for JAX-WS support:

- oracle.webservices.standalone.client.jar
- wseeclient.jar
- jps-az-api.jar

These JAR archives should be placed in the /src/lib/jaxws subdirectory of the RIDC installation directory.

9.1.2 SSL Communication

RIDC allows Secure Socket Layer (SSL) communication with Oracle Content Server using the Intradoc communication protocol.

Note: You must install and enable the SecurityProviders component in the Oracle Content Server instance that you want to access, and you must configure Oracle Content Server for SSL communication.

An example of using the IDC protocol over a secure socket (SSL) follows:

```
// build a secure IDC client as cast to specific type
IntradocClient idcClient = (IntradocClient
   manager.createClient("idcs://localhost:54444");

// 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
```

For more information, see Section 9.11, "Configuring SSL Communication with Oracle Content Server."

9.1.3 MBeans Implementation

RIDC provides an MBeans implementation allowing administrators to change properties of an RIDC connection at runtime using JMX and MBeans.

To register and enable MBeans, add the following to your code:

```
import oracle.stellent.ridc.convenience.adf.mbeans.IdcMBeanManager;
...
//connection name is the connection in the ADFContext you want to manage
IdcMBeanManager mbeanManager = IdcMBeanManager.getInstance(connectionName);
mbeanManager.register();
```

Once the application has started, edit the connection using a tool such as JConsole to connect to your application and change connection information while the application is running.

9.2 Initializing RIDC

To initialize RIDC, you will need the ECM Client libraries, which are shipped with the RIDC distribution, in your class path.

For the JAX-WS protocol, you also need to configure Oracle WebLogic Server security for the Oracle UCM web services. The security configuration includes these tasks:

- Setting up the policy for the login service
- Creating a new keystore file (or adding credentials to your existing keystore), which will be used by both the server and the client
- Setting up an Oracle wallet by adding the credentials

The client requires the following items:

- The JPS configuration file
- The keystore
- The Oracle wallet from the server

For information about configuring the server and client for web services, see the *Oracle Fusion Middleware Services Reference Guide for Oracle Universal Content Management*.

The following table shows the URL formats that are supported.

URL	Description
http://*host*/cs/idcplg	URL to the Oracle Content Server CGI path.
https://*host*/cs/idcplg	Uses SSL over HTTP; requires extra configuration to load the SSL certificates.
idc://*host*:4444	Uses the Intradoc port; requires only the hostname and the port number.
idcs://*host*:4444	Uses SSL over the Intradoc port; requires extra configuration to load the SSL certificates.
http://*wlsserver*:7044/idcnativews	Uses the JAX-WS protocol to connect to Oracle Content Server.

This example code initializes RIDC for an Intradoc connection:

```
// create the manager
IdcClientManager manager = new IdcClientManager();

// build a client that will communicate using the intradoc protocol
IdcClient idcClient = manager.createClient("idc://localhost:4444");
```

This example code initializes an HTTP connection (the only difference from an Intradoc connection is the URL):

```
// create the manager
IdcClientManager manager = new IdcClientManager();

// build a client that will communicate using the HTTP protocol
IdcClient idcClient = manager.createClient("http://localhost/idc/idcplg");
```

This example code initializes a JAX-WS client. These two web services are exposed by Oracle Content Server: the `login` service and the `request` service. You will need the web context root that these web services use. By default, this is `idcnativews`.

```
// create the manager
IdcClientManager manager = new IdcClientManager();

// build a client that will communicate using the JAXWS protocol
IdcClient idcClient = manager.createClient("http://wlsserver:7044/idcnativews");
```

9.3 Configuring Clients

Configuration of the clients can be done after they are created. Configuration parameters include setting the socket time outs, 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.

This example code sets the socket time out and wait time for Intradoc connections:

```
// build a client as cast to specific type
IntradocClient idcClient = (IntradocClient)manager.createClient
   ("http://host/cs/idcplg");

// get the config object and set properties
idcClient.getConfig ().setSocketTimeout (30000); // 30 seconds
idcClient.getConfig ().setConnectionSize (20);   // 20 connections
```

These JAX-WS specific configurations can be set after you have created the client:

```
// build a client as a cast for jaxws type
JaxWSClient jaxwsClient = (JaxWSClient) manager.createClient(
   ("http://wlsserver:7044/idcnativews");
JaxWSClientConfig jaxwsConfig = jaxwsClient.getConfig();
```

You can set the name of the Oracle Content Server instance that you want to connect to. By default, this is `/cs/`, which is the default web context for an Oracle UCM installation. If the server web context is different than the default, then you can set the the web context by editing a property. This example code sets your web context root:

```
// set the property
jaxwsConfig.setServerInstanceName("/mywebcontext/");
```

A JPS configuration file is required for most policies, such SAML or Message Token. This example code sets the location of the JPS configuration file:

```
jaxwsConfig.setJpsConfigFile("/my/path/to/the/jps-config.xml");
```

This example code sets the security policy:

```
jaxwsConfig.setClientSecurityPolicy
   ("policy:oracle/wss11_username_token_with_message_protection_client_policy");
```

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 URIs or URLs, you may need to modify the default values. This example code changes the `login` and `request` service URLs:

```
// login port wsdl url
jaxwsConfig.setLoginServiceWSDLUrl(new URL
   ("http://server:7044/webservices/loginPort?WSDL"));

//request port wsdl url
jaxwsConfig.setRequestServiceWSDLUrl(new URL
   ("http://server:7044/anotherservice/myrequestport?WSDL"));
```

The default streaming chunk size is set to `8192`. This example code changes the streaming chunk size:

```
jaxwsConfig.setStreamingChunkSize(8190);
```

9.4 Authenticating Users

All calls to RIDC require some user identity. Optionally, this identity can be accompanied by credentials as required by the protocol. The user identity is represented by the `IdcContext` object; once created, it can be reused for all subsequent calls. To create an identity, you pass in the user name and, optionally, some credentials:

```
//create a simple identity with no password (for idc:// urls)
IdcContext userContext = new IdcContext("sysadmin");

// create an identity with a password
IdcContext userPasswordContext = new IdcContext("sysadmin", "idc");
```

For Intradoc URLs, you do not need any credentials because the request is trusted between Oracle Content Server and the client.

For HTTP and JAX-WS URLs, the context needs credentials.

For HTTP URLs, the credentials can be a simple password or anything that the HttpClient package supports.

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.

9.5 Using Services

To invoke a service, use the `ServiceRequest` object, which you can obtain from the client. Creating a new request will also create a new binder and set the service name in the binder, along with any other default parameters. From that point, you can populate the binder as needed for the request.

This example code executes a service request and gets back a DataBinder object with the results:

```
// 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);

// get the binder
DataBinder serverBinder = response.getResponseAsBinder ();
```

The `ServiceResponse` object contains the response from Oracle Content Server. From the response, you can access the stream from Oracle Content Server directly, or you can parse it into a DataBinder object and query the results.

This example code takes a `ServiceResponse` and gets the search results, printing out the title and author values:

```
// 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"));
}
```

9.6 Handling Connections

The RIDC client pools connections, meaning that the caller of the code must close resources when it is done with a response. This is done by calling `getResponseAsBinder`, which closes resources automatically, or by calling `close` on the stream returned from a call to `getResponseStream`. If you do not want to examine the results, `close` must still be called, either by getting the stream and closing it directly or by calling `close` on the `ServiceResponse` object.

9.6.1 Closing Resources

The following examples show how to use the options for closing resources.

To close resources through getResponseAsBinder:

```
// execute the request
ServiceResponse response = idcClient.sendRequest (userContext, binder);

// get a binder closes the response automatically
response.getResponseAsBinder ();
```

To close resources by closing the stream:

```
// execute the request
ServiceResponse response = idcClient.sendRequest (userContext, binder);

// get the result stream and read it
InputStream stream = response.getResponseStream ();
int read = 0;
while ((read = stream.read ()) != -1)
{
}

//close the stream
stream.close ();
```

To close resources by calling the close method on the ServiceResponse object:

```
// execute the request
ServiceResponse response = idcClient.sendRequest (userContext, binder);

// close the response (which closes the stream directly)
response.close ();
```

9.6.2 Handling Connection Pooling

The `IdcClientConfig#getConnectionPool` property determines how RIDC will handle the pooling of connections. There are two options, `pool` and `simple`.

- `Pool` is the default, and it means to use an internal pool that allows a configurable number of active connections at a time (configurable via the `IdcClientConfig#getConnectionSize` property), with the default active size set to `20`.
- `Simple` does not enforce a connection maximum and rather lets every connection proceed without blocking.

You can register a different pool implementation through the `IdcClientManager#getConnectionPoolManager()#registerPool()` method, which maps a name to an implementation of the `ConnectionPool` interface. Then you can use the name in the `IdcClientConfig` object to select that pool for a particular client.

9.7 Sending and Receiving Streams

Streams are sent to Oracle Content Server through the `TransferStream` interface. This interface wraps the actual stream with metadata about the stream (length, name, and so on).

This example code performs a check-in to Oracle Content Server:

```
// 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);
```

You can receive a stream from Oracle Content Server through the `ServiceResponse` object; the response is not converted into a DataBinder object unless you specifically request it. If you just want the raw HDA data, you can get that directly, along with converting the response to a String or DataBinder object:

```
// 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 ();
```

9.8 Using RIDC Objects in JSP and JSPX Pages

The RIDC objects all follow the standard Java Collection paradigms, which makes them extremely easy to consume from a JSP or JSPX page. Assume the `ServerResponse` object (used in the previous example) was available in the `HttpServletRequest` object in an attribute called `idcResponse`. This example JSPX code will iterate over the response and create a small table of data:

```
<table>
  <tr>
    <th>Title</th>
    <th>Author</th>
    <th>Release Date</th>
  </tr>
<c:forEach var="row" items="${idcResponse.dataBinder.SearchResults.rows}">
  <tr>
    <td>${row.dDocTitle}</td>
    <td>${row.dDocAuthor}</td>
    <td>${row.dInDate}</td>
   </tr>
</c:forEach>
</table>
```

9.9 Reusing Binders for Multiple Requests

The binders can be reused among multiple requests. A binder from one request can be sent in to another request. For example, this example code pages the search results by reusing the same binder for multiple calls to Oracle Content Server:

```
// 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();
```

9.10 Providing User Security

Oracle UCM has several security models that are controlled by settings in Oracle 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 the Oracle 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 Oracle Content Server as that user to fetch that user's permissions. Once the user's permission controls are known, then they can be matched to the information in the document's metadata to resolve the access level for that user. The available access levels follow:

- READ
- READ/WRITE
- READ/WRITE/DELETE
- READ/WRITE/DELETE/ADMIN

It is preferable to reduce the number of calls to Oracle Content Server (using a cache) and to provide a default implementation for matching the user's permissions information with the document's permission information. One limitation is that Oracle Content Server controls which types of security are used in some server environment properties: `UseAccounts=true` and `UseCollaboration=true`.

The user security convenience is accessed through the `IUserSecurityCache` interface. Three concrete classes implement the optional Oracle Content Server security:

- The `UserSecurityGroupsCache` class simply keeps a cache of user permissions and will match documents considering only Security Group information.
- The `UserSGAccountsCache` class adds a resolver to also consider Account information if Oracle Content Server has the `UseAccounts=true` setting.
- The `UserSGAcctAclCache` class adds a resolver to also consider ACL permissions if `UseCollaboration=true`.

The `IAccessResolver` interface allows the addition of classes that can participate in the resolution of the access levels for a document.

This example code uses the three classes for implementing 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");

//Examples of the three concrete cache classes
IUserSecurityCache m_SGCache = new UserSecurityGroupsCache
   (m_client, 20, 1000);
IUserSecurityCache m_SGAcctCache = new UserSGAccountsCache
   (m_client, 20, 1000, 20000);
IUserSecurityCache m_SGAcctAclCache = new UserSGAcctAclCache
   (m_client, 20, 1000, 20000, m_superuser);

//Example test
testDocPermission () {
   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
      //Important: this makes a live call to Oracle Content Server
      //to get the user ID for "Acme1")
      //CacheId acme1 = m_SGAcctAclCache.getCacheIdForUser
      //   (new IdcContext("Acme1", "idc"));
      //You may want to include this:

   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_SGAcctAclCache.getAccessLevelForDocument (acme1, docInfo);
   }

//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()` processing:

- For the AccessResolverSecurityGroups class: `dSecurityGroup`
- For the AccessResolverAccounts class: `dDocAccount`
- For the AccessResolverSecurityGroups class: `xClbraUserList` and `xClbraAliasList`

The preceding `IAccessResolver` classes determine if they should participate based on cached information from Oracle Content Server. If they do participate, the access levels are put together with the AND operator.

9.11 Configuring SSL Communication with Oracle Content Server

RIDC allows Secure Socket Layer (SSL) communication with Oracle Content Server. This section provides basic information about SSL communication, including how to set up a sample implementation for testing purposes.

For SSL communication, you must install and enable the SecurityProviders component, configure Oracle Content Server 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 Oracle Content Server.

The sample implementation uses a JDK utility to create a self-signed key pair and certificates. Oracle does not provide signed certificates. For most implementations, you will want a certificate signed by a universally recognized Certificate Authority.

The following subsections describe how to configure SSL communication with Oracle Content Server:

- Section 9.11.1, "Installing and Enabling the SecurityProviders Component"
- Section 9.11.2, "Configuring an Incoming Provider for SSL Communication"
- Section 9.11.3, "Creating Self-Signed Key Pairs and Certificates"

9.11.1 Installing and Enabling the SecurityProviders Component

You must install and enable the SecurityProviders component in the Oracle Content Server instance you want to access. This component is installed and enabled by default in Oracle Content Server 11*g*R1.

In Oracle Content Server 10*g*R3, you need to install and enable the component manually. For information about installing and enabling components, see the Oracle UCM 10*g*R3 installation documentation.

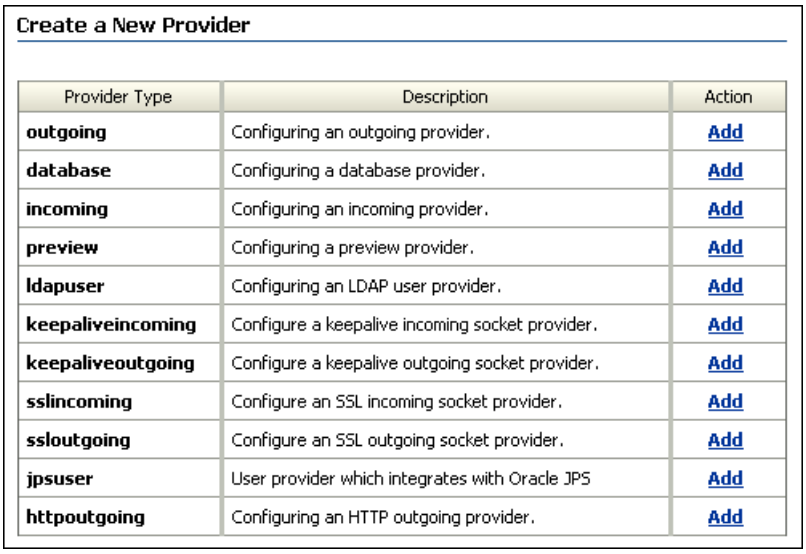
9.11.2 Configuring an Incoming Provider for SSL Communication

You can set up a new keepalive incoming socket provider or a new SSL incoming socket provider. The setup steps for both providers follow. Using keepalive improves the performance of a session and is recommended for most implementations.

To set up a new incoming provider:

1. Log in to Oracle Content Server as an administrator.
2. Click **Administration** and then **Providers**.

Figure 9–1 List of Providers in Oracle Content Server



Create a New Provider

Provider Type	Description	Action
outgoing	Configuring an outgoing provider.	Add
database	Configuring a database provider.	Add
incoming	Configuring an incoming provider.	Add
preview	Configuring a preview provider.	Add
ldapuser	Configuring an LDAP user provider.	Add
keepaliveincoming	Configure a keepalive incoming socket provider.	Add
keepaliveoutgoing	Configure a keepalive outgoing socket provider.	Add
sslincoming	Configure an SSL incoming socket provider.	Add
ssloutgoing	Configure an SSL outgoing socket provider.	Add
jpsuser	User provider which integrates with Oracle JPS	Add
httpoutgoing	Configuring an HTTP outgoing provider.	Add

3. Click **Add** for the `sslincoming` provider.

   The Add Incoming Provider page opens.
4. Enter a provider name and description.
5. Enter an open server port.
6. Enter configuration information for either a new SSL keepalive incoming socket provider or a new SSL incoming socket provider. The setup steps for both providers are listed in the following text. Using keepalive improves the performance of a session and is recommended for most implementations.

   SSL keepalive incoming socket provider

   - **Provider Class:** `idc.provider.ssl.SSLSocketIncomingProvider`
   - **Connection Class:** `idc.provider.KeepaliveSocketIncomingConnection`
   - **Server Thread Class:** `idc.server.KeepaliveIdcServerThread`

   SSL incoming socket provider

   - **Provider Class:** `idc.provider.ssl.SSLSocketIncomingProvider`
   - **Connection Class:** `intradoc.provider.SocketIncomingConnection`
   - **Server Thread Class:** `intradoc.server.IdcServerThread`
7. Click **Add**.

After you have completed setting up a new incoming provider, you must also specify truststore and keystore information.

9.11.3 Creating Self-Signed Key Pairs and Certificates

For most implementations, you will want a certificate signed by a universally recognized Certificate Authority. However, if you control both the client and server and want only to ensure that your transmissions are not intercepted, or if you are simply testing your implementation, you can create your own self-signed key pairs and certificates by using the JDK utility called keytool.

Key and Certificate Management Tool (keytool) is a key and certificate management utility that enables users to administer their own public and private key pairs and associated certificates for use in self-authentication. It is provided as part of the Sun JDK. Keytool is a command-line utility. The executable is located in the bin subdirectory.

9.11.3.1 Creating the Client and Server Keys

From a command-line prompt, navigate to the *JDK-Home*/bin subdirectory, and issue the `-genkey` command. This command generates a new key and takes several arguments. The arguments in the following table are used with this command.

Argument	Description
`-alias`	Alias of the key being created. This is the way a keystore knows which element in the file you are referring to when you perform operations on it.
`-keyalg`	Encryption algorithm to use for the key.
`-keystore`	Name of the binary output file for the keystore.
`-dname`	Distinguished name that identifies the key.
`-keypass`	Password for the key that is being generated.
`-storepass`	Password used to control access to the keystore.

Generate a separate key pair for both the client and the server. To do this, you will need to run the `-genkey` command twice, each time placing the key pair into a separate keystore.

You will need to specify the alias, the algorithm to use, the keystore name, the distinguished name, and passwords for the keys and the keystore. This example uses RSA as the algorithm and `idcidc` as the password for the key and the keystore.

Use these argument values for the client:

- `-alias SecureClient`
- `-keyalg RSA`
- `-keystore client_keystore`
- `-dname "cn=SecureClient"`
- `-keypass idcidc`
- `-storepass idcidc`

```
# keytool -genkey -alias SecureClient -keyalg RSA -keystore client_keystore -dname
"cn=SecureClient" -keypass idcidc -storepass idcidc
```

Use these argument values for the server:

- `-alias SecureServer`
- `-keyalg RSA`
- `-keystore server_keystore`
- `-dname "cn=SecureServer"`
- `-keypass idcidc`
- `-storepass idcidc`

```
# keytool -genkey -alias SecureClient -keyalg RSA -keystore client_keystore -dname
"cn=SecureClient" -keypass idcidc -storepass idcidc

# keytool -genkey -alias SecureServer -keyalg RSA -keystore server_keystore -dname
"cn=SecureServer" -keypass idcidc -storepass idcidc
```

Each of these commands will generate a key pair wrapped in a self-signed certificate and stored in a single-element certificate chain.

9.11.3.2 Self-Signing the Certificates

Keys are unusable unless they are signed. The keytool utility will self-sign them for you so that you can use the certificates for internal testing. However, these keys are not signed for general use.

From a command line prompt, issue the `-selfcert` command (this command self-signs your certificates and takes several arguments). Run the `-selfcert` command twice, once for the client and again for the server.

Use these argument values for the client:

- `-alias SecureClient`
- `-keystore client_keystore`
- `-keypass idcidc`
- `-storepass idcidc`

Use these argument values for the server:

- `-alias SecureServer`
- `-keystore server_keystore`
- `-keypass idcidc`
- `-storepass idcid`

Examples of `-selfcert` commands follow:

```
# keytool -selfcert -alias SecureClient -keystore client_keystore -keypass idcidc
-storepass idcidc

# keytool -selfcert -alias SecureServer -keystore server_keystore -keypass idcidc
-storepass idcidc
```

The certificate is now signed by its private and public key, resulting in a single-element certificate chain. This replaces the one that you generated previously.

9.11.3.3 Exporting the Certificates

After you have created the client and server keys and self-signed the certificates, you now have two key pairs (public and private keys) in two certificates locked in two keystores. Since each application will need to have the public key of the other to encrypt and decrypt data, you need to place a copy of each public key in the other application's keystore.

From a command-line prompt, issue the `-export` command (this command exports your certificates and takes several arguments). Run the `-export` command twice, once for the client and again for the server. Use the `-file` argument to redirect the output to a file instead of the console.

Use these argument values for the client:

- `-alias SecureClient`
- `-file client_cert`
- `-keystore client_keystore`
- `-storepass idcidc`

Use these argument values for the server:

- `-alias SecureServer`
- `-file server_cert`
- `-keystore server_keystore`
- `-storepass idcidc`

Examples of `-export` commands follow:

```
# keytool -export -alias SecureClient -file client_cert -keystore client_keystore
-storepass idcidc
```

(Certificate stored in the file client_cert)

```
# keytool -export -alias SecureServer -file server_cert -keystore server_keystore
-storepass idcidc
```

(Certificate stored in the file server_cert)

The certificate (containing the public key and signer information) has now been exported to a binary certificate file.

9.11.3.4 Importing the Certificates

The final step in setting up your self-signed certificates is to import the public certificates of each program into the keystore of the other. Keytool will present you with the details of the certificates you are requesting to be imported and provide a request confirmation.

From a command line prompt, issue the `-import` command (this command imports your certificates and takes several arguments). Run the `-import` command twice, once for the client and again for the server. Notice that the `-keystore` values are reversed.

Use these argument values for the client:

- `-alias SecureClient`
- `-file client_cert`
- `-keystore server_keystore`
- `-storepass idcidc`

Use these argument values for the server:

- `-alias SecureServer`
- `-file server_cert`
- `-keystore client_keystore`
- `-storepass idcidc`

Examples of `-import` commands follow:

```
# keytool -import -alias SecureClient -file client_cert -keystore server_keystore
-storepass idcidc

Owner: CN=SecureClient
Issuer: CN=SecureClient
Serial number: 3c42e605
Valid from: Mon Jan 14 08:07:01 CST 2002 until: Sun Apr 14 09:07:01 CDT 2002
Certificate fingerprints:
  MD5: 17:51:83:84:36:D2:23:A2:8D:91:B7:14:84:93:3C:FF
  SHA1: 61:8F:00:E6:E7:4B:64:53:B4:6B:95:F3:B7:DF:56:D3:4A:09:A8:FF
Trust this certificate? [no]: y
Certificate was added to keystore

# keytool -import -alias SecureServer -file server_cert -keystore client_keystore
-storepass idcidc

Owner: CN=SecureServer
Issuer: CN=SecureServer
Serial number: 3c42e61e
Valid from: Mon Jan 14 08:07:26 CST 2002 until: Sun Apr 14 09:07:26 CDT 2002
Certificate fingerprints:
  MD5: 43:2F:7D:B6:A7:D3:AE:A7:2E:21:7C:C4:52:49:42:B1
  SHA1: ED:B3:BB:62:2E:4F:D3:78:B9:62:3B:52:08:15:8E:B3:5A:31:23:6C
Trust this certificate? [no]: y
Certificate was added to keystore
```

The certificates of each program have now been imported into the keystore of the other.

9.12 Using Tables for Content Items, the Search Index, and the File Store

The following subsections describe how to search tables for information about content items:

- Section 9.12.1, "Finding Information for Each Content Item"
- Section 9.12.2, "Using a Search Index"
- Section 9.12.3, "Using the File Store Provider"

9.12.1 Finding Information for Each Content Item

Content managed by Oracle 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, with 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 for each revision
- One column for each metadata field
- Definition of 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

9.12.2 Using a Search Index

Oracle 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 `SDATA` section 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.

9.12.3 Using the File Store Provider

The File Store Provider can be used to distribute files managed by Oracle Content Server on the file system, a database, other devices, or any combination of these. The files are stored in SecureFiles in Oracle 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 field (SecureFiles in Oracle 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 Oracle 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.

10

Using Content Integration Suite (CIS)

This chapter describes how to use the Content Integration Suite (CIS), which offers access to Oracle Content Server by exposing its services and data in a unified object model. The Universal Content and Process Management (UCPM) API is modeled into a set of services APIs, which are API calls that communicate with the target server, and the returned value objects from the server.

This chapter includes the following sections:

10.1 CIS Architecture

CIS has a layered architecture that allows for its deployment in a number of different configurations. The architecture, at its core, is based on the standard J2EE Command Design Pattern. The layers on top of the commands provide the APIs that are exposed to the end user.

CIS uses the Universal Content and Process Management (UCPM) API, which uses the SCS API for communication to Oracle Content Server. The SCS API wraps communication from Oracle Content Server into an object model that allows access to the individual object metadata.

The UCPM API enables application developers to focus on presentation issues rather than being concerned with how to access Oracle Content Server services (IdcCommand services). It comprises a set of command objects which encapsulate distinct actions that are passed to the UCPM API and then mapped to Oracle Content Server. These commands include common content management functions such as search, check-out, and workflow approval. Each command is tied to one or more service calls. The UCPM API command objects have been developed in accordance with the J2EE Command Design Pattern.

This infrastructure is deployable in any J2EE-compliant application server or stand-alone JVM application. When deployed, the UCPM API leverages the features in the environment, whether this is a J2EE application server or non-J2EE server.

The UCPM API encapsulates Oracle Content Server business logic and validates the parameters of the incoming calls. It also handles communication with Oracle Content Server, encapsulates socket communication logic (opening, validating, and streaming bits through the socket), and provides a strongly typed API to the available services.

Internationalization and Character Encoding

Oracle recommends that encoding for CIS should be set to the same encoding as the Java Virtual Machine running Oracle Content Server. However, if CIS is communicating with multiple Oracle Content Server instances in different languages, then the ISCSContext.setEncoding method can be used to set the encoding to match that of the JVM running CIS.

Deprecated FixedAPI

The Fixed API available in CIS releases before for communication with the Image Server has been deprecated. Calling getFixedAPI() throws an error.

10.2 Access Through the UCPM API

The Universal Content and Process Management (UCPM) API offers access to Oracle Content Server instances by exposing their services and data in a unified object model. The UCPM API is modeled into a set of services APIs that communicate with the target server, which are API calls that communicate with the target server, and into ICISObject objects, which are the value objects returned from the server.

The UCPM API is available on the ICISApplication class via the get UCPMAPI() method. The getUCPMAPI() method returns a reference to the IUCPMAPI object, allowing access to all UCPM API objects. The IUCPMAPI public interface is the locator for the getActiveAPI object; **getActiveAPI()** returns a reference to the SCSActiveAPI object. The SCS API classes communicate with, and handle content stored on, Oracle Content Server.

10.3 UCPM API Methodology

The UCPM API is stateless; all method calls pass in the necessary state to the method. This means that you can share the reference to the CISApplication class across threads.

- **ISCSContext** for the SCS API. The ISCSContext interface is the context object used for communicating with Oracle Content Server.
- **ICISCommonContext** for calling some of the CIS APIs. The ICISCommonContext interface identifies which adapters to query and what user information to use.

The first parameter for all methods is an IContext bean. The IContext bean holds context information, such as user name and session ID, that is used in the underlying service APIs to identify the user invoking the given command.

The UCPM API is a service-oriented API that returns value objects, implemented as ICISObject objects (name changed from the 7.6 API). However, calling methods on the value objects themselves do not modify content on the server; one must call the UCPM API and pass in the value object as a parameter before the changes can be applied.

Example:

```
SCSActiveAPI activeAPI = m_cisApplication.getUCPMAPI ().getActiveAPI ();
ISCSDocumentID documentID = (ISCSDocumentID) m_cisApplication.getUCPMAPI ().
  createObject(ISCSDocumentID.class);
documentID.setDocumentID("10");
ISCSDocumentInformationResponse docResponse =
  activeAPI.getDocumentInformationAPI ().
  getDocumentInformationByID(m_context, documentID);
ISCSContent content = docResponse.getDocNode();

// call does not change object on server
  content.setTitle ("New Title");

// now item is updated on server after this call
  activeAPI.getDocumentUpdateAPI ().updateInfo (m_context, content);
```

10.4 CIS Initialization

Content Integration Suite (CIS) is initialized by accessing the CISApplicationFactory class, which resides in the com.stellent.cis.impl package, as the following subsections describe:

- Section 10.4.1, "Initialization"
- Section 10.4.2, "SCSInitializeServlet"

10.4.1 Initialization

CIS initialization should happen once per application. The CIS APIs are stateless and the initialized CISApplication instance can therefore be safely shared between threads.

To initialize CIS, you must define a number of properties. The cis.config.type value should be `server`, and the cis.config.server.type value should be `standalone`. The adapter configuration file contains configuration information in XML format for communicating with Oracle Content Server instances.

Initialization Examples

The following properties initialize the system and read the adapterconfig.xml file from the class path:

```
cis.config.type=server
cis.config.server.type=standalone
cis.config.server.adapterconfig=classpath:/adapterconfig.xml
```

Code example:

```
ICISApplication application;
  URL xmlRes = new File ("adapterconfig.xml").toURL()
  Properties properties = new Properties();
  properties.setProperty(ICISApplication.PROPERTY_CONFIG_TYPE, "server");
```

```
  properties.setProperty(ICISApplication.PROPERTY_CONFIG_SERVER_ADAPTER_CONFIG,
    xmlRes.toExternalForm());

properties.setProperty(ICISApplication.PROPERTY_CONFIG_SERVER_TYPE, "standalone");
application = CISApplicationFactory.initialize(properties);
```

Property Definitions

The properties are defined in the following table.

Property	Description
cis.config.type	Should be set to `server`.
cis.config.server.type	Should be set to `standalone`.
cis.config.server.adapterconfig	The URL pointing to the adapter configuration file. In addition to standard URLs, this can be in class path form (classpath:/) or file form (file:/)
cis.config.server.temporarydirectory	The location of the temporary directory used for file transfers, streaming, and file caching.

10.4.2 SCSInitializeServlet

The SCSInitializeServlet is a convenient way to initialize a CISApplication instance from within a web application. Any of the properties described can be used by the SCSInitializeServlet. The SCSInitializeServlet can be configured externally via a properties file. The `cis.initialization.file` property can be set with a path (either a web-relative path or a class path reference), to a property file containing the initialization properties. This allows you to easily externalize the initialization to a properties file.

By default, if SCSInitializeServlet finds no properties in the web.xml file, it will attempt to load a properties file from the WAR and then from the class path using the default value `/cis-initialization.properties`. Thus, if you place a file called cis-initialization.properties in your class path (that is, in the same directory as the adapterconfig.xml file), that file will be read during startup.

This properties file can hold all the standard initialization properties as defined in the CISApplication class. This allows you to move the configuration of how CIS initializes outside the scope of the EAR/WAR file.

Server Property Definitions

The server properties are defined in the following table. The defaults can be overridden by creating a file named cis-initialization.properties and saving the file to the server-ear directory of the unbundled CIS distribution file (this is the directory containing the adapterconfig.xml file).

Property	Description
cis.config.type	Must be set to server (default).
cis.config.server.adapterconfig	The URL pointing to the adapterconfig.xml file. In addition to standard URLs, this can also be in class path form (that is, classpath:/)
cis.config.server.type.options.ejb=true	Default is true (EJBs enabled).
cis.config.server.type.options.rmi=true	Default is true (MI enabled).

Initialization Process

At startup, the SCSInitializeServlet servlet begins the CIS server initialization process. It attempts to load various properties from the web.xml file and class path (that is, cis-initialization.properties). It then passes those properties to the static method CISApplicationFactory.initialize(…). This section describes the order of operation.

SCSIntitialize init(ServletConfig)

Called by the web application container during initialization of the web application.

1. Load properties from web.xml.
2. Load properties from classpath: /cis-initialization.properties.
3. Call CISApplicationFactory.initialize(properties).
4. Via the CISWebHelper class, it sets the CISApplication and the command application instance as an attribute on the servlet context.

CISApplicationFactory initialize(properties)

Called by the init(…) method of an object instance of SCSInitialize.
Calls CISApplicationFactory.initializeCisApplication(properties).

CISApplicationFactory initializeCisApplication(properties)

Determines if CIS should be started in client or server mode.
Calls CISApplicationFactory.initializeServer(properties).

CISApplicationFactory initializeServer(properties)

Called by CISApplicationFactory.initialize(properties).

1. Creates the adapter config URL (used to eventually load the adapterconfig).
2. Loads IsolatedJarClassloader.
3. Calls CISApplication.initialize(properties).

10.5 Integration in a Web Environment

This is what you would put in the web.xml file if you wanted the SCSInitializeServlet to start up and register the CIS application:

```
<servlet id="scsInitialize">
  <servlet-name>scsInitialize</servlet-name>
  <display-name>SCS Initialize Servlet</display-name>
  <servlet-class>com.stellent.cis.web.servlets.SCSInitializeServlet
    </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

If you add a new JSP page called search.jsp, it would look like this code:

```
<%-- JSTL tag library --%>
<%@ taglib uri="/WEB-INF/tlds/c.tld" prefix="c" %>

<%--  CIS Application object placed in servlet context by the SCSInitialize
servlet. Get the CIS Application and make a query --%>
```

```
<%
ICISApplication cisApplication =
  (ICISApplication) request.getSession().getServletContext().
    getAttribute ("CISApplication");
ISCSSearchAPI searchAPI =
  cisApplication.getUCPMAPI ().getActiveAPI ().getSearchAPI ();

// create a context
ISCSContext context =
  cisApplication.getUCPMAPI ().getActiveAPI ()._createSCSContext ();
  context.setUser ("sysadmin");

// execute the search
ISCSSearchResponse response =
  searchAPI.search (context, "dDocAuthor substring 'sysadmin'", 20);
%>
<!-- model the search results as desired -->
```

The SCSInitializeServlet places the initialized CISApplication class as an attribute in the javax.servlet.ServletContext with the name CISApplication. The CISApplication instance is available to the entire web application and is thread-safe; the one instance can be shared across the application.

10.6 Class Loading

The UCPM 8.0.0 API uses a custom class loader to isolate dependencies on specific libraries from any application that uses the CIS API. For more information, see the Java 2 Platform API specification on the ClassLoader page of Oracle Sun Technology Network at

http://download.oracle.com/javase/1.4.2/docs/api/java/lang/ClassLoader.html

The following subsections describe the custom class loader and how to use it:

- Section 10.6.1, "Custom Class Loader"
- Section 10.6.2, "Class Loader Usage"

10.6.1 Custom Class Loader

The implementation of this paradigm is found in the com.stellent.cis.impl.IsolatedJarClassLoader object. This custom class loader allows for a JAR to have a nested set of JAR files that serve as the library. This is the structure of the nested JAR files:

```
+ cis-application-8.0.0.jar
    +-- com/stellent/cis/...
    +-- META-INF
    +-- lib/
        +-- spring-1.1.5.jar
        +-- log4j-1.0.3.jar
        +-- ...
```

All libraries that live in the lib directory will be the class path for the CIS objects. The class loader will query this local directory first before loading files from the parent classloader.

However, all of the JAR files cannot be isolated as the consuming client needs to have access to some API classes so they can be imported into their application space. Therefore, only the interfaces are exposed in the application space, keeping the implementation and associated dependencies isolated. Because the class loader for the dependencies and implementation has access to the parent loader, it can access the same versions of the interfaces as the application that is using the APIs. This also implies that a client will be able to access only the interfaces of the UCPM API. Any attempt to create an implementation class using the new keyword will result in a `ClassNotFoundException` error.

10.6.2 Class Loader Usage

To use this class loader, use the new com.stellent.cis.impl.CISApplicationFactory object to initialize the system. This object will automatically detect and use the IsolatedJarClassLoader if required. This means that you can still deploy CIS in the original format, with all the class files and libraries at the application level, if you so desire.

In the current version, CIS only needs the cis-client-8.0.0.jar file in the class path; no other libraries are needed. Once cis-client-8.0.0.jar is in the application class path, you initialize CIS using the following code:

```
// the initialization properties (as defined in ICISApplication)
  Properties properties = new Properties ();
  ICISApplication cisApplication = CISApplicationFactory.initialize(properties);
```

Once you have a reference to com.stellent.cis.ICISApplication, you can interact with the APIs. The difference is that now you only have access to interface objects (everything in com.stellent.cis.client) and not the implementation objects.

If you implement custom commands, your public interfaces will also need to be in the class path, and the implementation classes packaged in the cis-client-8.0.0.jar.

10.7 Object Creation

The UCPM 8.0.0 API uses a customized classloader to hide library dependencies and implementation classes. Only the client interface classes are exposed to the user. However, this implies that you cannot use the new keyword to instantiate UCPM API objects. Therefore, in the UCPM API framework, use the generic `_create` methods available on the IUCPMAPI object to tell the system to instantiate an instance of the given object.

Because objects are mutable and only the interfaces are exposed, use the createObject method and set the properties that you need:

```
ISCSDocumentID docID =
  (ISCSDocumentID)ucpmAPI.createObject(ISCSDocumentID.class);
docID.setDocumentID("20");
```

10.8 Interaction with the UCPM API

With an initialized CISApplication instance, the getUCPMAPI () method (of the CISApplication object) returns a reference to the IUCPMAPI object, allowing access to all UCPM API objects. The IUCPMAPI interface is the locator for the various API objects in the UCPM API.

The IUCPMAPI object has methods to get references to the Active API; getActiveAPI () returns a reference to the SCSActiveAPI object for communicating with Oracle Content Server. This enables you to access the necessary APIs and begin making calls through the UCPM API to the target server.

Calling API Objects Using Newly Instantiated ICISObject Objects

Many UCPM API calls take in an ICISObject or an interface that inherits from ICISObject. For example, in the ISCSDocumentInformationAPI (SCS API) the getDocumentInformationByID () method takes as a parameter a ISCSDocumentID object.

The fully qualified method name is:

```
ICISApplication.getUCPMAPI ().getActiveAPI ().getDocumentInformationAPI ()
```

In such cases, to obtain a reference to a valid object to pass in as a parameter, you can either retrieve the object reference from another ICISObject or create a new instance of the ICISObject using the generic createObject method available in the UCPM API. In CIS 11*g*R1, a customized classloader is used to hide the library dependencies and implementation classes. Only the client interface classes are exposed to the user. However, this implies that you cannot use the new keyword to instantiate UCPM API objects. Therefore, in the CIS API framework, you use the generic create method available on the IUCPMAPI object to tell the system to instantiate an instance of the given object. The createObject method will let you create a new instance of any ICISObject.

For example, if you wanted to query for document information, but only had the document ID, you would do the following:

```
ISCSDocumentInformationAPI documentInfoAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI ().getDocumentInformationAPI ();

// create the document ID
ISCSDocumentID documentID =
  (ISCSDocumentID) m_cisApplication.getUCPMAPI ().
    createObject (ISCSDocumentID.class);
  documentID.setDocumentID("12345");

ISCSDocumentInformationResponse docResponse =
  documentInfoAPI.getDocumentInformationByID(m_context, documentID);
```

For any API that requires an ICISObject object, you can use the createObject method, which enables you to create a new instance of the API object. The createObject method is a client-side method; it does not make a call to the target server. It is to be treated as a constructor for the ICISObject implementations.

Building a More Complex Object

If you needed to build up a more complex object such as a new content item to be checked in to Oracle Content Server, you would need to create several objects and populate the data:

```
// Create an empty content object
ISCSContent content =
  (ISCSContent) m_cisApplication.getUCPMAPI ().createObject(ISCSContent.class);

// Create an empty content ID object, and then give it a content ID
ISCSContentID contentID = (ISCSContentID) m_cisApplication.getUCPMAPI ().
  createObject(ISCSContentID.class);
  contentID.setContentID("my_document");
```

```
// Set all of the properties of the content item required for check in
  content.setContentID(contentID);
  content.setAuthor (context.getUser ());
  content.setTitle ("Document Title");
  content.setSecurityGroup ("Public");
  content.setType ("ADACCT");
  content.setProperty ("xCustomProperty", "Value for custom property");
```

10.9 IContext Interface

The IContext interface is the generic context used for communication with the Command APIs. This interface handles contextual information to determine the current caller identity, the target adapter, and so on.

The context should be populated with the user name and adapter name. The adapter name is determined by the adapterconfig.xml file, and the user name can be any valid user ID for the target server.

IContext has the subinterfaces SCSContext and ISISContext, which extends the IContext interface. **ISCSContext** is the context object used for the SCS APIs and represents a user's operating context during communication with Oracle Content Server.

The context object can be created by using the `_create` method. Thus, ISCSContext can be created from SCSActiveAPI.

```
// create an ISCSContext
  ISCSContext context =
  m_cisApplication.getUCPMAPI ().getActiveAPI ()._createSCSContext ();
```

Once the context is created, it should be populated with a user name and the adapter name. This can be done by using the accessor methods on the IContext bean.

```
  context.setUser ("sysadmin");
  context.setAdapterName ("myadapter");
```

The CIS API will take either an ICISCommonContext object or an IContext object. ICISCommonContext is a special kind of context that is used as a container for ISCSContext and ISISContext. It is required in APIs that federate information between a number of different adapters; it identifies which adapters to query and what user information to use. In instances where the call only operates against one adapter at a time, a single IContext is required.

```
// create an ICommonContext
  ICISCommonContext m_commonContext =
  m_cisApplication.getUCPMAPI ().getCommonAPI ()._createCommonContext ();
```

Once the ICISCommonContext adapter is created, multiple adapters can be added to it. This is done using the ICISCommonContext.addContext() method. Any number of adapters can be added; all the adapters added to the ICISCommonContext are then used individually during a Common API call.

The same ISCSContext object can be used for multiple queries and across threads. In a web application context, the easiest method is to add the IContext object to the session and retrieve it from the session for each query.

An sample web application has been provided (located in SDK/Samples/WebSample) which has a login method that first validates the username against Oracle Content Server and, if successful, adds the IContext object to the HttpSession object. For more details, see the LoginActionHandler class in the src/com/stellent/sdk/web/sampleapp/handlers/login directory.

10.10 ICISObject Interface

ICISObject is the base interface for all objects with metadata in the UCPM API. Thus, all UCPM API objects are inherited from ICISObject. The ICISObject interface allows for the retrieval and setting of the object properties. The objects returned from calls to the UCPM API are value objects in the form of beans (reusable software components) that encapsulate data from the server call, not live objects. Updating or modifying the objects in any fashion will not affect server data; only by directly calling a method on a given UCPM API can the server data be modified.

The following subsections describe property accessors, object types, and collections:

- Section 10.10.1, "Property Accessors"
- Section 10.10.2, "Property Object Types"
- Section 10.10.3, "Property Collections"

10.10.1 Property Accessors

Most implementations of ICISObject have their own specific property accessor methods. However, all properties can be retrieved by calling the getProperty () method on the ICISObject. The ICISObject.getProperty () method will return an ICISProperty object. From this object you can get the property value or property information using these methods:

- **getValue()** returns the property value. If the property has a null value, calling getValue () will result in a null reference.
- **getDescriptor()** returns the property descriptor that describes the contents of the property value.

```
// use the response object from the previous example - retrieve the content object
  ISCSContent content = docResponse.getDocNode ();

// get the title property
  String title = content.getTitle ();

// get the title by using the ICISObject getProperty method
  title = content.getProperty ("title").getValue ().getStringValue ();
```

The ICISObject property methods may throw a PropertyRetrievalException if an error occurs during the lookup of a given property. Since the PropertyRetrievalException is a RuntimeException, it does not have to be caught directly in your code. Common cases for the exception to be thrown is when a property is asked for but does not exist or when the property value contains invalid data. For more information about the specific reason for the error, you can catch this exception and then query the exception class.

ISCSProperty also allows for setting property values back into the property object. To do this, you can use an appropriate set method or call setProperty () and pass in the bean property name (both are valid and both will set the property value on the target object).

```
// set the title - using the content object from the previous example
  content.setTitle ("My New Title");

// set using the setProperty method
  content.setProperty ("title", "My New Title");
```

10.10.2 Property Object Types

A property object type is determined by the return value of the property method on the ICISObject. When using the generic getProperty() method, the ISCSPropertyValue has methods to get both the value of the property and the value as a specific object type (for example, Boolean, float, or long).

The ISCSPropertyValue is retrieved through the getValue() method on the returned ISCSProperty object.

When you set a property through the generic setProperty() method, it is important that the property value passed into the method is of the correct type or can be converted to the appropriate type through simple BeanUtils property conversion.

Apache BeanUtil is a utility for populating bean properties from the org.apache.commons project.

In the ISCSContent object, the property readOnly is type Boolean. Therefore, in the following example, the first three methods will successfully set the property value and the last method will not:

```
// correct
  content.setReadOnly (true);
  content.setProperty ("readOnly", Boolean.TRUE);
  content.setProperty ("readOnly", "true");

// incorrect
  content.setProperty ("readOnly", "not a boolean");
```

Since the setProperty () method takes an object as the second parameter, the Boolean encapsulation must be used. Also, as mentioned, the method uses the BeanUtils property conversion and therefore the string `true` converts to the Boolean value TRUE. As shown in the preceding example, passing a property value that cannot be converted (for example, not a Boolean value) will result in an exception.

10.10.3 Property Collections

The available list of properties can be retrieved using the getProperties() method on the ICISObject interface. This will return all of the available properties for a given object.

```
// using the content item from the previous example
  Collection properties = content.getProperties ();

// iterate through the collection
for (Iterator it = properties.iterator (); it.hasNext (); ) {
  ISCSProperty property = (ISCSProperty)it.next ();
  String name = property.getDescriptor ().getName ();
  ICISPropertyValue value = property.getValue ();
  if (value != null) {
     System.out.println (name + " = " + value.getStringValue ());
  }
}
```

10.11 Adapter Configuration File

The adapter configuration file (adapterconfig.xml) contains XML-formatted configuration information for communicating with your Oracle Content Server instance. It specifies the CIS layer that servers use to open communications.

A single connection to a server is called an adapter. Any number of adapters can be configured in the adapterconfig.xml file. The adapterconfig.xml file is required to initialize the CISApplication instance.

The following subsections describe the elements in the adapter configuration file:

- Section 10.11.1, "The adapter Element"
- Section 10.11.2, "The config Element"

10.11.1 The adapter Element

Each adapter configuration is a separate element in the XML markup. The adapter element has four attributes as shown in the following table.

Adapter Attributes	Description
type	Should be `scs` for a connection to Oracle Content Server.
default	If true, then this is the default adapter for this type. Only one default adapter for a given type is allowed.
name	The adapter name.

A sample adapter element follows:

```
<adapter type="scs" default="true" name="myadapter">
```

10.11.2 The config Element

The config element includes a set of property elements that define the adapter-specific properties. These configuration elements are described in the following text.

SCS Adapter Configuration Elements

An SCS adapter communicates with Oracle Content Server. The configuration element for the SCS adapter has four general attributes, as shown in the following table.

Property Name	Description
port	The port for Oracle Content Server instance.
host	The host name or IP address of Oracle Content Server.
type	These values may be used: **socket**: Uses the Oracle Content Server socket communication layer. **mapped**: Uses shared directories to transfer the files for file upload and download. **web**: Uses HTTP requests to transfer files; requires an Oracle Content Server username and password for Basic HTTP Authentication (file download only).

A sample SCS configuration element follows:

```
<adapter name="myadapter" type="scs" default="true">
  <config>
    <property name="host">localhost</property>
    <property name="port">4444</property>
    <property name="type">socket</property>
    <property name="version">75</property>
  </config>
  <beans template=
  "classpath:/META-INF/resources/adapter/adapter-services-scs.jxml"/>
</adapter>
```

By default, the Oracle Content Server socket communication layer is used to stream files to and from Oracle Content Server. However, for high-volume check in or file retrieval, you can set the **mapped** or **web** optimized file transfer options.

A mapped transfer loads the files from a shared directory on Oracle Content Server; this results in much faster file transfers and does not tie up a socket that could be used for other requests. To use mapped transfer, you must define these properties.

Property Name	Description
contentServerMappedVault	The Oracle Content Server vault directory as seen from the application server.
appServerMappedVault	The application server vault directory as seen from Oracle Content Server.

A web transfer uses HTTP requests to the Oracle Content Server web server to download files. To use web transfer, you must define these properties.

Property Name	Description
contentServerAdminID	The Oracle Content Server administrator ID to use to authenticate against Oracle Content Server.
contentServerAdminPassword	The Oracle Content Server administrator password to use to authenticate against Oracle Content Server. This password is encrypted.

A sample SCS configuration element using web transfer follows:

```
<adapter type="scs" default="true" name="myadapter">
  <config>
    <property name="port">4444</property>
    <property name="host">localhost</property>
    <property name="type">web</property>
    <property name="contentServerAdminID">sysadmin</property>
    <property name="contentServerAdminPassword">idc</property>
  </config>
</adapter>
```

10.12 Access to the SCS API

The UCPM API is available on the CISApplication class via the getUCPMAPI () method. The getUCPMAPI () method returns a reference to the IUCPMAPI object, allowing access to all UCPM API objects. The IUCPMAPI public interface is the locator for the SCS, SIS, and CIS API objects. The SCS API is available via getActiveAPI (), which returns a reference to the SCSActiveAPI object.

The fully qualified method name follows:

```
CISApplication.getUCPMAPI ().getActiveAPI ()
```

The SCS API comprises the following APIs:

- **ISCSSearchAPI**: This is the command API implementation of the search commands.
- **ISCSFileAPI**: Deals with the retrieval of files, and the dynamic conversions of files, from Oracle Content Server.
- **ISCSWorkflowAPI**: Deals with the workflow commands such as approval and rejection, viewing a user's workflow queue, and interacting with the Oracle Content Server workflow engine.
- SCS Document APIs (**ISCSDocumentCheckinAPI** and **ISCSDocumentCheckoutAPI**), which deal with active content in Oracle Content Server, including checking in and out of content, content information, and deletion of content.
- Various APIs for the implementation of the administrative commands, component commands, and so on.

The ICommandFacade interface is the entry point into the command interface. It allows for interaction with the command layer, including command retrieval, registration, and execution. Commands are referenced by name, where a name can be any string. A name consisting of the dot character (".") will be treated in a hierarchy, where the first segment is the top-level category, and the next segment is the second-level category, and so on. Commands can either be retrieved by their full command name or by browsing all available commands.

The fully qualified class name is

```
com.stellent.command.ICommandFacade
```

The following example uses ISCSDocumentCheckinCommandAPI:

```
ISCSDocumentCheckinCommandAPI commandAPI =
  (ISCSDocumentCheckinCommandAPI)m_commandFacade.
    getCommandAPI ("document.checkin");
```

10.13 SCS API Objects

The SCS API is responsible for formulating requests to Oracle Content Server. SCS API calls translate into one or more IDC Service (Oracle Content Server service) calls.

The following subsections describe the SCS API objects:

- Section 10.13.1, "ISCSObject Interface"
- Section 10.13.3, "ISCSServerBinder Interface"
- Section 10.13.3, "ISCSServerBinder Interface"
- Section 10.13.4, "ISCSServerResponse Interface"
- Section 10.13.5, "ISCSRequestModifier Interface"

10.13.1 ISCSObject Interface

The ISCSObject interface is the base interface for all objects in the SCS API. It inherits from ICISObject and adds some specific functionality relative to Oracle Content Server objects. It allows access to the ISCSServerResponse object that created the object, and it also allows access to a collection of properties that have been modified since the object was initialized via the getModifiedProperties() method.

The ICISObject class name is new for UCPM 8.0.0 API.

ISCSObject objects have the concept of native property names. Specifically, properties that are available on ISCSObject are available by two different names: the Java property name and the Oracle Content Server native name. For example, to get the title of a ISCSContent item, the following three methods are equal:

```
String title = content.getTitle ();
title = content.getProperty ("title").getValue ().getStringValue ();
title = content.getProperty ("dDocTitle").getValue ().getStringValue ();
```

Oracle Content Server supports a metadata model that can be extended. ISCSObject objects can have more properties than those exposed via the getProperty () methods. The ISCSObject implementations expose the most common properties, but other properties, such as the extended metadata, are only available via the getProperty () method. Also, the getProperties() method will list all the properties on the object, including properties without a corresponding **getter** or **setter** method.

```
for (Iterator it = content.getProperties ().iterator (); it.hasNext (); ) {
  ISCSProperty property = (ISCSProperty)it.next ();
  ISCSPropertyDescriptor descriptor = property.getActiveDescriptor ();
  if (descriptor.isBeanProperty ()) {
    System.out.println ("Property is available via get or set: " +
                        property.getDescriptor().getName ());
  } else {
    System.out.println ("Property is a hidden or extended property: " +
                        property.getDescriptor().getName ());
  }
    System.out.println ("Native property name: " + descriptor.getNativeName ());
}
```

The properties returned from ISCSObject implement the ISCSProperty interface, which adds the getActiveDescriptor() method. The ISCSPropertyDescriptor adds the `beanProperty` and `nativeName` properties to the available properties on an item.

The beanProperty property determines if the current property object has an available **getter** or **setter** method; if the property is false, this property object is available only via the getProperty() method.

The nativeName property returns the Oracle Content Server property name for the given property.

Date Objects

Date fields in the SCS API are handled as Java Date objects in Coordinated Universal Time (UTC time). All dates passed into the various properties of ISCSObject must be in UTC time. All date objects returned from the SCS API are in UTC time and need to be converted into the appropriate time zone for the particular application.

```
Date releaseDate = content.getReleaseDate ();

// convert from UTC time to Pacific Time Zone
  Calendar calendar = Calendar.getInstance ();
  calendar.setTime (releaseDate);
  calendar.setTimeZone (TimeZone.getTimeZone ("America/Los_Angeles"));
// use calendar to display date...
```

10.13.2 ICISTransferStream Interface

File streams to and from Oracle Content Server have changed. In an effort to keep with the interface-only approach to the CIS APIs, all streams are sent through the ICISTransferStream interface. This interface represents the actual physical stream object and additionally some metadata, including filename, content length and mime-type. Because ICISTransferStream is an interface, it cannot extend InputStream directly. Therefore, when using this object, you must first obtain the stream via a call to ICISTransferStream.getStream() and then manipulate the stream appropriately.

Some commands that in previous versions would return an InputStream now return ICISTransferStream objects. For example, if calling getFile() in the FileAPI, to access the stream from Oracle Content Server, your code would look like:

```
ICISTransferStream transferStream = fileAPI.getFile (context, documentID);
InputStream inputStream = transferStream.getInputStream();
```

The implementation of ICISTransferStream contains all the necessary plumbing to transfer the stream to and from the command client to command server. Since InputStream objects are not directly serializable, it does some extra work to put streams into places where the server can access them. All of the logic is hidden from the user of the API.

The stream instance can be obtained from the root IUCPMAPI interface using the method createTransferStream. That returns an empty instance of the stream container, which you can then use the accessors methods to set the stream properties. The following example creates a transfer stream and points it at a local file:

```
// create the stream implementation
ICISTransferStream transferStream = ucpmAPI.createTransferStream ();

// point it at a file
transferStream.setFile (new File ("mytestdoc.doc"));
```

If you had a stream in memory already rather than a file handle, and you wanted to check in the content in that stream into Oracle Content Server, you would need to specify all of the attributes for the stream such as a filename, content type, and the length of the stream.

```
ICISTransferStream transferStream = ucpmAPI.createTransferStream ();
  transferStream.setFileName ("sample.txt");
  transferStream.setInputStream (inputStream);
  transferStream.setContentType ("text/plain");
  transferStream.setContentLength (length);
  checkinAPI.checkinFileStream (context, content, transferStream);
```

10.13.3 ISCSServerBinder Interface

The CIS 8.0.0 API provides a new object, ISCSServerBinder, which is the root object used for all communication to and from Oracle Content Server. The ISCSServerBinder object encapsulates a message both to and from Oracle Content Server. It is a collection of properties, result sets, files, option lists and other specific type of information needed by Oracle Content Server.

All API calls into Oracle Content Server will create an instance of the ISCSServerBinder. Each API call, for example ISCSSearchAPI.search(), will use the supplied ISCSObject parameters to populate a binder and possibly add in other particular information. Likewise, all responses from Oracle Content Server that are not streams are ISCSServerResponse objects which extend ISCSServerBinder.

As all ISCSObjects are collections of arbitrary metadata, the ISCSServerBinder is a collection of a number of objects metadata contained within one object. A particular ISCSServerBinder might contain the metadata for an Oracle Content Server query (see ISCSSearchQuery), metadata concerning a piece of content, and a list of objects dealing with user data. As each ISCSObject contains the particular metadata for its object, the ISCSServerBinder is responsible for the metadata of many objects.

Properties

As ISCSServerBinder extends the core ISCSObject interface, it has the ability to get and set arbitrary properties via the getProperty and setProperty methods. These arbitrary properties are usually where arguments for a particular Oracle Content Server instance are placed. They can be added directly, via the setProperty method as shown in the following code example.

```
// create an empty binder
ISCSServerBinder binder =
  (ISCSServerBinder)getUCPMAPI ().createObject (ISCSServerBinder.class);

// set some properties
  binder.setProperty ("dDocTitle", "test");
  binder.setProperty ("dSecurityGroup", "Public");
```

Alternatively, properties can be set using the mergeObject functionality available on the ISCSObject interface. The following example shows creating another object, setting some properties on that object, and then using merge to put those properties into the server binder.

```
// create an empty content item
ISCSContent content = (ISCSContent)getUCPMAPI ().createObject (ISCSContent.class);

// set some properties
content.setTitle ("test");
content.setSecurityGroup ("Public");

// merge into binder; this copies all the properties from content into the binder
binder.mergeObject (content);
```

The preceding two examples are identical: they both result in setting the content item title (dDocTitle) and security group (dSecurityGroup) in the ISCSServerBinder object. However, the second method is an abstraction from the specifics of naming. The ISCSContent object handles the mapping of standard Java properties into Oracle Content Server metadata.

Result Sets

A result set represents a collection of rows returned from an Oracle Content Server query. This is exposed in ISCSServerBinder through the getResultSet and setResultSet methods. A result set in the SCS API is then exposed as a homogeneous list, containing a type of object the represents a single row of the result set. Many items returned from Oracle Content Server queries come back as result sets. As all the result sets are lists of ISCSObject objects, the items from the result sets can be used in other calls. For example, look at the following code snippet where a search is executed and the first item then has its contents updated:

```
// create an simple query
ISCSSearchQuery query =
  (ISCSSearchQuery)getUCPMAPI ().createObject (ISCSSearchQuery.class):
  query.setQueryText ("dDocName substring 'test' ");

// execute a search
ISCSSearchResponse response =
  getUCPMAPI ().getActiveAPI ().getSearchAPI ().search (context, query);

// search results come back as a result set of ISCSSearchResult items
ISCSSearchResult result = (ISCSSearchResult)response.getResults ().get (0);

// change the title and check it in
result.setTitle ("new title");
getUCPMAPI ().getActiveAPI ().getDocumentUpdateAPI ().
  updateInfo (context, result);
```

File Objects

The ISCSServerBinder allows files to be sent to Oracle Content Server through the addFile method. This method takes an ICISTransferStream. The resulting file is sent to Oracle Content Server, along with the binder, during the request. Adding a file is similar to adding a property:

```
// create an empty stream
ICISTransferStream stream = getUCPMAPI ().createTransferStream ();

// point the stream at a local file
stream.setFile (new File ("testfile.txt"));

// add the stream to the binder
serverBinder.addStream ("myFile", stream);
```

When the preceding binder is sent to Oracle Content Server, the stream will be transferred along with the binder. Inside Oracle Content Server, the stream will be available under the "myFile" key which was specified when adding the stream to the binder.

Object Copying and Casting

Each ISCSObject in the SCS API has an object that holds the data in a low-level format compatible with Oracle Content Server. This object, referred to as a data object, is used by all ISCSObject implementations. This implies that any ISCSObject can be mutated to any other type of ISCSObject. There are two methods that expose this functionality: the castObject and copyObject methods available on the ISCSObject interface.

Both methods take in a single parameter: a class type representing the type of object that should be created. A call to **castObject** will result in the creation of a new object that points at the same backing data of the object it was invoked against. This implies that changes to the original object will be reflected in the object returned from the

castObject call as well. A call to **copyObject** will result in a copy of the backing data being made, allowing the newly created object to act independently from the original object.

For example, imagine a custom Oracle Content Server service called "MY_DOC_INFO" that is similar to the standard "DOC_INFO" but does some extra business logic processing. However, the returned binder from the "MY_DOC_INFO" call is very close to the "DOC_INFO" call. Because there is no explicit API call in the SCS API to call this "MY_DOC_INFO" service, the generic executeIDCService call has to be used. But the castObject method can be used to change the return type into something more user friendly:

```
// build the custom call
ISCSServerBinder binder =
  (ISCSServerBinder)getUCPMAPI ().createObject (ISCSServerBinder.class);
  binder.setService ("MY_DOC_INFO");

// create a document ID and add it to the binder
ISCSDocumentID documentID =
  (ISCSDocumentID)getUCPMAPI ().createObject (ISCSDocumentID.class);
  documentID.setDocumentID ("12345");
  binder.mergeObject (documentID);

// execute the call
ISCSServerResponse response =
  (ISCSServerResponse)getUCPMAPI ().getAdministrativeAPI ().
    executeIDCService (context, binder);

// use the cast to change it to a ISCSDocumentInformationResponse
ISCSDocumentInformationResponse infoResponse =
  (ISCSDocumentInformationResponse)response.
    castObject (ISCSDocumentInformationResponse.class);

// use the info response as usual
System.out.println ("Title: " + infoResponse.getDocNode ().getTitle ());
```

As mentioned in the preceding text, the castObject call links the two objects by sharing the same backing data. In the preceding example, any changes made to the response object would be reflected in the infoResponse object as well:

```
// set a property on the response
response.setProperty ("customProperty", "customValue");

// value is then available in the infoResponse object
String value = infoResponse.getPropertyAsString ("customProperty");
```

If copyObject was called instead, the response and infoResponse would be independent of each other. The castObject creates a smaller memory footprint than copyObject, since the result from a castObject call does not create a new backing data object.

10.13.4 ISCSServerResponse Interface

The result of a call to the SCS API is usually an ISCSServerResponse object. ISCSServerReponse is the base interface for all the response objects. It encapsulates the response from Oracle Content Server for the last request. Most methods have specific implementations of this interface, which provide properties that are specific to those responses. For the specific response objects and the properties available for each response object, see the Javadocs.

In the current release, some calls that previously returned references to an InputStream now return a ICISTransferStream object instead. For information about how to use the new transfer stream interface, see Section 10.13.5, "ISCSRequestModifier Interface."

10.13.5 ISCSRequestModifier Interface

All requests to Oracle Content Server through the SCS API result in the creation of an ISCSServerBinder object. In certain situations, it becomes necessary to change the binder contents for a given API call to match the requirements of a specific Oracle Content Server instance. The ISCSRequestModifier interface is designed for this very purpose: to augment a call to Oracle Content Server with custom modifications.

All APIs in the SCS API have a corresponding method that takes as the first parameter an ISCSRequestModifier object. Look at the API for ISCSSearchAPI:

```
/**
   * Command the implements searching against Oracle Content Server.
  * @param SCSContext the context object representing the current user
  * @param searchQuery the Oracle Content Server query object
  */

public com.stellent.cis.client.api.scs.search.
  ISCSSearchResponse search (com.stellent.cis.client.api.scs.context.
    ISCSContext SCSContext,
    com.stellent.cis.client.api.scs.search.ISCSSearchQuery searchQuery)
  throws com.stellent.cis.client.command.CommandException;

/**
  * Command the implements searching against Oracle Content Server.
  * @param requestModifier modify the request
  * @param SCSContext the context object representing the current user
  * @param searchQuery the Oracle Content Server query object
  * @see com.stellent.cis.server.api.scs.commands.search.SearchCommand
  */

public com.stellent.cis.client.api.scs.search.
  ISCSSearchResponse search (com.stellent.cis.client.api.scs.
    ISCSRequestModifier requestModifier, com.stellent.cis.client.api.scs.context.
    ISCSContext SCSContext, com.stellent.cis.client.api.scs.search.I
    SCSSearchQuery searchQuery)
    throws com.stellent.cis.client.command.CommandException;
```

The second API takes all the parameters of the first, with the additional ISCSRequestModifier argument. The standard Search API, and all its logic, can be used and custom logic added. For example, imagine a custom component installed on Oracle Content Server that will do some extra processing if it find the `myCustomProperty` property set during a search query. To do this with CIS, the ISCSRequestModifier can be used to change the binder, as follows:

```
// build a search query
ISCSSearchQuery query =
  (ISCSSearchQuery)getUCPMAPI ().createObject (ISCSSearchQuery.class);
  query.setQueryText ("dDocName substring `test`");

// build a request modifier
ISCSRequestModifier modifier =
  (ISCSRequestModifier)getUCPMAPI ().createObject (ISCSRequestModifier.class);

// access the binder off the modifier and add in the custom data
modifer.getServerBinder().setProperty ("myCustomProperty", "customValue");
```

```
// execute the search as normal
getUCPMAPI ().getActiveAPI ().getSearchAPI ().search (modifier, context, query);
```

Now the modified binder will be used during the search call. The custom property value will get sent along with the standard search call. This same method can be used to set any properties, add files, set result sets or even override which service call is being made on Oracle Content Server.

10.14 SCS API Servlets

The SCS API requires that a number of servlets be available to the system while operating in a J2EE/web environment and running in server mode.

The following subsections describe the SCS API servlets and how they work:

- Section 10.14.1, "Servlet Descriptions"
- Section 10.14.2, "SCS Servlet Parameters"
- Section 10.14.3, "Servlet Security"
- Section 10.14.4, "Servlets and API Interaction"

10.14.1 Servlet Descriptions

This table lists the servlet names and the appropriate configuration information needed in the web.xml file for a given web application.

Fully Qualified Name	Mapping	Description
SCSFileDownloadServlet com.stellent.web.servlets.SCSFileDownloadServlet	/getfile	Allows clients to retrieve files from Oracle Content Server.
SCSCommandClientServlet com.stellent.web.servlets.SCSCommandClientServlet	/scscommandclient	Publishes the CIS server configuration information for CIS clients.
SCSFileTransferServlet com.stellent.web.servlets.SCSFileTransferServlet	/scsfiletransfer	Enables APIs in the UCPM API to transfer files to a CIS client.
SCSInitialize com.stellent.web.servlets.SCSInitialize	N/A	Initializes the CIS Application instance. Should be set as a LoadOnStartup servlet.
SCSDynamicConverterServlet com.stellent.web.servlets.SCSDynamicConverterServlet	/getdynamicconversion/*	Executes a dynamic conversion and streams the result to the client.
SCSDynamicURLServlet com.stellent.web.servlets.SCSDynamicURLServlet	/scsdynamic/*	Retrieves dynamic files from Oracle Content Server; used when rewriting the dynamically converted document URLs.

10.14.2 SCS Servlet Parameters

This section provides a description of the parameters for these servlets:

- SCSFileDownloadServlet
- SCSDynamicConverterServlet
- SCSDynamicURLServlet

This section does not specify any of the available security parameters, which are described in Section 10.14.3, "Servlet Security." All calls made to Oracle Content Server use the identity as specified in the servlet security section.

10.14.2.1 SCSFileDownloadServlet

Property	Required	Description
adapterName	true	The adapter name to query for the document.
dDocName	n/a	The content ID of the document to retrieve.
rendition	false	The content rendition; valid only when specifying the dDocName.
revisionSelection	false	The revisionSelection to use when selecting content; valid only when specifying the dDocName.
forceStream	false	If true, the contents are streamed from Oracle Content Server via the GET_FILE call regardless of optimized file transfer settings for the adapter; defaults to false.

10.14.2.2 SCSDynamicConverterServlet

Property	Required	Description
adapterName	true	The adapter name to query for the document.
contentID	Either contentID or documentID is required.	The content ID (dDocName) of the document to retrieve.
documentID	n/a	The document ID (dID) of the document to retrieve.
rendition	false	The rendition of the document to retrieve; valid only if contentID is specified.
revisionSelectionMethod	false	The revisionSelectionMethod to use to select the document; valid only if a `contentID` value is specified.
viewFormat	false	The view format of the conversion (that is, Native or WebViewable).
useAlternate	false	If true, use the alternate file for conversion; default is false.

10.14.2.3 SCSDynamicURLServlet

Property	Required	Description
adapterName	true	The adapter name to query for the document; not passed in as a parameter, but, rather, specified as the last segment on the URL: /cis-server/scsdynamic/*adaptername*?...
fileUrl	true	The relative path to the Oracle Content Server file to retrieve.

10.14.3 Servlet Security

All servlets, except for SISFileDownloadServlet and SCSInitializeServlet, make UCPM API calls and therefore must have a user context. By default, they will use the HttpServletRequest.getUserPrincipal() method to determine the user ID and pass that ID via the ISCSContext object to the UCPM API call. This behavior can be overridden by specifying a couple of initialization parameters to the servlet:

- **principalLookupAllowed**: If set to TRUE, the servlet will look for a user ID in the configured scope. The default scope is `session`.
- **principalLookupScope**: The scope of the lookup. Valid if principalLookupAllowed is TRUE. The defined scope will be used to call the getAttribute() method to discover the name of the current user; can be either `request`, `session`, or `application`. The default is `session`.
- **principalLookupName**: The name of the scoped parameter that holds the user ID. Valid if principalLookupAllowed is TRUE. The default is `principal`.
- **getUserPrincipalEnabled**: If set to FALSE, no call will be made to the HttpServletRequest.getUserPrincipal() method to determine the user ID. The default is TRUE.
- **principal**: The default user ID if no user ID can be determined. The default is `guest`.

To determine the current user ID, the servlets will first check the status of the principalLookupAllowed flag. If TRUE, it looks up the name of the user by determining the scope as set by the parameter principalLookupScope. With the current scope, the getAttribute() method is called, using principalLookupName as the parameter. If it is unable to locate a principal, it then checks the status of the getUserPrincipalEnabled flag. If that flag is TRUE, it calls the HttpServletRequest.getUserPrincipal() method. If that returns null, it uses the default principal to execute the request.

Without any changes to the servlet, the default behavior is to check the HttpServletRequest.getUserPrincipal() method and then use the default, if necessary. The other checks on the request, session, and application are done only if specified in the **init-param** of the servlet definition in the web.xml file.

10.14.4 Servlets and API Interaction

The ISCSFileAPI.getDynamicConversion() method performs a dynamic conversion of the given document (assuming the Dynamic Converter component is installed on Oracle Content Server). The getDynamicConversion() call will also rewrite the returned URLs, so that they point back to the CIS servlets (as opposed to pointing

directly to Oracle Content Server) and display properly in the web/Portal environment when they are rendered.

The rewritten URLs point back to SCSDynamicURLServlet, which then retrieves the item from Oracle Content Server, through the SCS API, and streams it back to the client. The servlet determines the user ID for the context by the method described in Section 10.14.3, "Servlet Security."

Since the servlet determines the user ID, the user who executed the getDynamicConversion() call might not have the same user ID as the user clicking a link on the rendered HTML. This would be the case if the HttpServletRequest.getUserPrincipal() user ID does not match the ISCSContext user ID

In that event, the SCSDynamicURLServlet can be directed to look for a user parameter on the session by customizing the servlet with the methods described in Section 10.14.3, "Servlet Security." Alternatively, SCSDynamicURLServlet can call the getDynamicConversion() and pass in an ISCSConvertedUrlInfo object that allows a user to optionally add parameters to the URL, which can then be used by your application to identify the context.

For example, if your application stored the current User ID in a session attribute named `stellentPrincipal`, you would modify the web.xml for the SCSDynamicURLServlet (and other servlets, as necessary) as follows:

```
<servlet>
  <servlet-name>scsdynamic</servlet-name>
  <servlet-class>com.stellent.web.servlets.SCSDynamicURLServlet</servlet-class>
  <init-param>
    <param-name>sessionPrincipalAllowed</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>sessionPrincipalName</param-name>
    <param-value>stellentPrincipal</param-value>
  </init-param>
</servlet>
```

10.15 SCS APIs

The SCS Search, SCS File, SCS Document, and SCS Workflow APIs are discussed and sample code provided. These APIs perform task such as searching, checking in and out of content, and workflow approval and rejection.

It is assumed that you have initialized a CISApplication instance (referred to as m_cisApplication) and created a context object (referred to as m_context). Additional samples can be found in the SDK/Samples/CodeSamples directory.

The following subsections describe the SCS APIs:

- Section 10.15.1, "SCS Search API"
- Section 10.15.2, "SCS File API"
- Section 10.15.3, "SCS Document APIs"
- Section 10.15.4, "SCS Workflow API"

10.15.1 SCS Search API

The ISCSSearchAPI is the command API implementation of the search commands. You can use ISCSSearchAPI to search Oracle Content Server with the following code:

```
// get a handle to the SCS Search API
ISCSSearchAPI searchAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI ().getSearchAPI ();
ISCSSearchResponse searchResponse =
  searchAPI.search (m_context, "dDocTitle substring 'HR'",  25);

// iterate all results
for (Iterator it = searchResponse.getResults ().iterator (); it.hasNext (); ) {
  ISCSSearchResult searchResult = (ISCSSearchResult)it.next ();

// print out the title and author
System.out.println ("Found result: " + searchResult.getTitle () + " by " +
  searchResult.getAuthor ());
}
```

10.15.2 SCS File API

The ISCSFileAPI deals with the retrieval of files, and the dynamic conversions of files, from Oracle Content Server. A file can be retrieved simply by passing in the ID for the content. Alternatively, different versions of the file can be retrieved by using the optional ISCSFileInfo object to obtain references to the `Web` and `Alternate` versions of the file.

```
// get the SCS File API
ISCSFileAPI fileAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI ().getFileAPI ();

ICISTransferStream transferStream =
  fileAPI.getFile (m_context, content.getDocumentID ());

InputStream stream = transferStream.getInputStream();
// do something with the stream...
```

You can also use the _createFileInfo() method to get an ISCSFileInfo object. This object has several properties, which enable you to further select which rendition of a file to retrieve. The following sample uses the **fileinfo** object to get the `web-viewable` rendition of a file. A similar process can be used to get the `Alternate` rendition.

```
// get the web-viewable version of the file
ISCSFileInfo fileInfo =
  (ISCSFileInfo) m_cisApplication.getUCPMAPI ().createObject(ISCSFileInfo.class);
  fileInfo.setRendition ("Web");

// get the file
ICISTransferStream transferStream =
  fileAPI.getFile (m_context, content.getDocumentID (), fileInfo);
  InputStream stream = transferStream.getInputStream();
// do something with the stream...
```

The SCS File API can be used to generate HTML renditions of the content via the Dynamic Converter component of Oracle Content Server (you must have the Dynamic Converter component installed).

In a similar fashion to the getFile() calls, you can either call getDynamicConversion() with an ID to retrieve the HTML conversion, or you can use the ISCSFileInfo and ISCSConvertedFileInfo objects to pass information into the API to process conversion rules and apply explicit templates.

```
ICISTransferStream transferStream =
  fileAPI.getDynamicConversion (m_context, content.getDocumentID ());
// process the stream...
```

The following sample combines the preceding features in one method that dynamically converts the alternate rendition of a given content object by using a custom conversion template.

```
// create the converted file bean and set the properties
ISCSConvertedFileInfo convertedInfo = fileAPI.__createConvertedFileInfo ();
convertedInfo.setConversionLayout ("custom_layout");
convertedInfo.setRendition ("Alternate");

// execute the dynamic conversion
ICISTransferStream transferStream =
  fileAPI.getDynamicConversion (m_context, content.getDocumentID (),
    convertedInfo);
// do something with the stream...
```

10.15.3 SCS Document APIs

The SCS Document APIs deal with content in Oracle Content Server, including the checking in and out of content, content information, and the deletion of content.

The following subsections describe these APIs:

- Section 10.15.3.1, "ISCSDocumentCheckinAPI"
- Section 10.15.3.2, "ISCSDocumentCheckoutAPI"

10.15.3.1 ISCSDocumentCheckinAPI

This API deals with the check-in of all content to Oracle Content Server. For a simple check-in of a file from disk, the following code will work:

```
// get the checkin api
ISCSDocumentCheckinAPI checkinAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI ().getDocumentCheckinAPI ();

// create an empty content object with the specified content ID
ISCSContent content =
(ISCSContent) m_cisApplication.getUCPMAPI ().createObject(ISCSContent.class);

ISCSContentID contentID =
(ISCSContentID) m_cisApplication.getUCPMAPI ().createObject(ISCSContentID.class);
  contentID.setContentID("my_test_file");
  content.setContentID(contentID);
  content.setAuthor (m_context.getUser ());
  content.setTitle ("Custom Title");
  content.setSecurityGroup ("Public");
  content.setType ("ADACCT");

// get the file stream
File myFile = new File ("c:/test/testcheckin.txt");
ICISTransferStream transferStream =
  m_cisApplication.getUCPMAPI ().createTransferStream();
  transferStream.setFile(myFile);
```

```
// execute the checkin
checkinAPI.checkinFileStream (m_context, content, transferStream);
```

In many deployments of Oracle Content Server, some required extended properties need to be set for a new piece of content. These properties can be set on the content object through the setProperty() call available to all ICISObject objects. For example, some custom properties can be set as follows:

```
// set an extended property
  content.setProperty ("xCustomProperty", "Custom Value");
```

You can use the setProperty() method to set all the properties as opposed to calling the **setter** methods. You can use either the JavaBean name (for example, *title*) or the native Oracle Content Server property name that the JavaBean property corresponds to (that is, *dDocTitle*). In the next sample, the title property will be set in three ways, all equivalent:

```
// set through a standard property setter
  content.setTitle ("My Title");

// set a standard property using the JavaBean property name
  content.setProperty ("title", "My Title");

// set a property using the native Oracle Content Server property name
  content.setProperty ("dDocTitle", "My Title");
```

10.15.3.2 ISCSDocumentCheckoutAPI

This API deals with checking out content from Oracle Content Server. Content items are identified by their ID.

```
// get the checkout api
ISCSDocumentCheckoutAPI checkoutAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI ().getDocumentCheckoutAPI ();

// checkout the file
checkoutAPI.checkout (m_context, content.getDocumentID ());
```

10.15.4 SCS Workflow API

The ISCSWorkflowAPI deals with the workflow commands such as approval and rejection, viewing a user's workflow queue, and interacting with the Oracle Content Server workflow engine. The following sample code shows an example of querying the workflow engine for the workflows currently active in the system:

```
// get the workflow API
ISCSWorkflowAPI workflowAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI().getWorkflowAPI ();
ISCSWorkflowResponse workflowResponse =
  workflowAPI.getActiveWorkflows (m_context);

// iterate through the workflows
for (Iterator it = workflowResponse.getActiveWorkflows().iterator();
  it.hasNext (); ) {
    ISCSWorkflow workflow = (ISCSWorkflow)it.next ();
    String name = workflow.getName ();
    String status = workflow.getWorkflowStatus ();
    System.out.println ("SCS workflow: " + name + "; status = " + status);
}
```

The most common interaction with workflows is to reject them or approve them and advance them to the next step in the workflow. The following code illustrates how to get a user's personal workflow queue and approve all workflows pending:

```
// get the workflow API
ISCSWorkflowAPI workflowAPI =
  m_cisApplication.getUCPMAPI ().getActiveAPI().getWorkflowAPI ();

// get the workflow queue
ISCSWorkflowQueueResponse queueResponse =
  workflowAPI.getWorkflowQueueForUser (m_context);
for (Iterator it = queueResponse.getWorkflowInQueue().iterator(); it.hasNext();) {
ISCSWorkflowQueueItem queueItem =
  (ISCSWorkflowQueueItem)it.next();

// approve the workflow
workflowAPI.approveWorkflow(m_context, queueItem.getDocumentID ());
}
```

11

Using the Java Content Repository Adapter

This chapter describes how to use the Oracle Content Server Java Content Repository (JCR) adapter.

This chapter includes the following sections:

11.1 Introduction to Using 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 Oracle 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 Oracle Content Server through the standards-based JCR specification.

11.1.1 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.

11.1.2 Oracle Content Server JCR Adapter Data Model

This is the data model for the Oracle 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>
```

11.2 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.

The following subsections describe how to install or deploy these APIs and runtime libraries:

- Section 11.2.1, "Installing ADF Runtime Libraries"
- Section 11.2.2, "Deploying Remote Intradoc Client (RIDC)"
- Section 11.2.3, "Deploying the JCR API"
- Section 11.2.4, "Installing the JCR Integration Libraries"
- Section 11.2.5, "Installing the XML Integration Files"

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).

11.2.1 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
- adfloggінghandler.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.)

11.2.2 Deploying Remote Intradoc Client (RIDC)

Remote Intradoc Client must be deployed on your application. RIDC provides a thin communication API for communication with Oracle Content Server. This API removes data abstractions to the Oracle 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 9, "Using Remote Intradoc Client (RIDC)."

11.2.3 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 web site at `http://www.apache.org/`

The JCR API is also part of the JSR-170 specifications download from the Java Community Process web site at

`http://www.jcp.org/`

11.2.4 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.

11.2.5 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.

11.3 Deploying the JCR Adapter

The JCR adapter must be deployed on your application to enable communication with an Oracle Content Server instance. 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.

11.4 Configuring Communication with Oracle Content Server

You must supply several configuration values to enable communication between the JCR adapter and Oracle Content Server. The following subsections describe these configuration values:

- Section 11.4.1, "Supplying a Communication Method"
- Section 11.4.2, "Configuring Socket Communication (Listener Port)"
- Section 11.4.3, "Configuring Secure Socket Communication (SSL)"
- Section 11.4.4, "Configuring Web Communication (Web Server Filter)"
- Section 11.4.5, "Configuring the User Agent"
- Section 11.4.6, "Supplying Cache Settings"

11.4.1 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 Oracle 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 Oracle 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.

11.4.2 Configuring Socket Communication (Listener Port)

You must supply values for these configuration settings if Oracle Content Integration Suite (CIS) is connecting through the Oracle Content Server listener port (socket communication) or if secure socket communication (SSL) is used as the communication protocol:

- SERVER_HOST_CONFIG: The hostname of the machine on which Oracle Content Server is running. The default value is `localhost`.

  `oracle.stellent.jcr.configuration.server.host`

- SERVER_PORT_CONFIG: The port on which Oracle Content Server is listening. The default value is `16200`.

  `oracle.stellent.jcr.configuration.server.port`

11.4.3 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 11.4.2, "Configuring Socket Communication (Listener Port)."

11.4.4 Configuring Web Communication (Web Server Filter)

You need to supply a value for one of these configuration settings if CIS is connecting through the web server filter (web communication):

- `SERVER_WEB_CONTEXT_ROOT_CONFIG`: The web server context root for the Oracle Content Server instance, 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 Oracle 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

11.4.5 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

11.4.6 Supplying Cache Settings

You can optionally supply values for these cache settings:

- VCR_CACHE_INVALIDATION_INTERVAL: Polling interval used by the Oracle UCM 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

11.5 Using Tables for Content Items, the Search Index, and the File Store

The following subsections describe how to search tables for information about content items:

- Section 11.5.1, "Finding Information for Each Content Item"
- Section 11.5.2, "Using a Search Index"
- Section 11.5.3, "Using the File Store Provider"

11.5.1 Finding Information for Each Content Item

Content managed by Oracle 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

11.5.2 Using a Search Index

Oracle 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.

11.5.3 Using the File Store Provider

The File Store Provider can be used to distribute files managed by Oracle Content Server on the file system, a database, other devices, or any combination of these. The files are stored in SecureFiles in Oracle 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 field (SecureFiles in Oracle 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 Oracle 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.

12

Using Oracle UCM Web Services

This chapter describes how to use Oracle Universal Content Management (Oracle UCM) web services with Oracle WebLogic Server web services to manage Oracle Content Server.

This chapter includes the following sections:

12.1 Overview of Oracle UCM Web Services

Web services reside as a layer on top of existing software systems such as application servers, .NET servers, Oracle WebLogic Server, and Oracle Content Server. Web services can be used as a bridge to dissimilar operating systems or programming languages. Web services are adapted to the Internet as the model for communication and rely on the HyperText Transfer Protocol (HTTP) as the default network protocol. Thus, using web services, you can build applications using a combination of components.

Oracle UCM web services work with Oracle WebLogic Server web services to perform management functions for Oracle Content Server installed on Oracle WebLogic Server. Oracle WebLogic Server web services provide SOAP capabilities, and Oracle UCM web services include several built-in SOAP requests. Oracle UCM web services are automatically installed with an Oracle UCM instance, but they require additional configuration to set up security.

Core enabling technologies for Oracle UCM web services include:

- SOAP (Simple Object Access Protocol) is a lightweight XML-based messaging protocol used to encode the information in web service request and response messages before sending them over a network. SOAP requests are sent by the Oracle UCM web services to the 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 Oracle UCM 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* web 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 Oracle UCM web services, policies are used for applying WS-Security to web services. The two supported policies are username-token security and Security Assertion Markup Language (SAML) security.

  Historically, Oracle used Oracle Web Services Manager (OWSM) 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 OWSM and WS-SecurityPolicy policies can work with each other.

  Note: Use OWSM policies over Oracle WebLogic Server web services whenever possible. You cannot mix your use of OWSM and Oracle WebLogic Server web services policies in the same web service.

  Oracle UCM web services (/idcws as context root) are SOAP based, while Oracle UCM native web services (/idcnativews as context root) are JAX_WS based. Both kinds of web services can be assigned OWSM policies through the Oracle WebLogic Server Administration Console.

  The generic Oracle UCM web services are JAX-WS based and can be assigned OWSM policies and managed by OWSM. The native Oracle UCM web Services are SOAP based and can only support WS-Policy policies managed through the Oracle WebLogic Server Administration Console.

  For more information about OWSM, see the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

  A subset of WebLogic web service policies interoperate with OWSM policies. For more information, see "Interoperability with WebLogic Web Service Policies" in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

  Web Services Security Policy (WS-SecurityPolicy) is a set of security policy assertions for use with the WS-Policy framework. For more information, see *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*.

12.2 Oracle UCM Web Services

Oracle UCM 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 an Oracle Content Server instance through any regular web services client.
- `/idcnativews/` - The Remote IDC client (RIDC) uses the native web services. It is recommended that you **do not** develop custom client against these services.

The following table describes the Oracle UCM web service in the `/idcws/` context root.

Oracle UCM 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 Oracle Content Server when you set IsSoap=1. Details of the format can be found in the published WSDL at idcws/GenericSoapPort?WSDL. You can apply WS-Security to GenericSoapService through WS-Policy. Oracle Content Server supports Oracle Web Services Manager (OWSM) policies for Security Assertion Markup Language (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.

The following table describes the Oracle UCM web services in the idcnativews/ context root.

Oracle UCM Web Services	Descriptions
IdcWebRequestService	This is the general Oracle UCM service. Essentially it is a normal socket request to Oracle Content Server, wrapped in a SOAP request. Requests are sent to Oracle 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 for future calls to IdcWebRequestService. If you want to use WS-Security with IdcWebRequestService, then apply it here. Oracle Content Server supports Oracle Web Services Manager (OWSM) policies for Security Assertion Markup Language (SAML) and username-token.

12.3 Installation and Configuration

The Oracle UCM web services are installed and ready to use by default with the Oracle UCM EAR. However, unless you configure WS-Security on any of the Oracle UCM web services, all connections to Oracle Content Server will use the "anonymous" user. Additional configuration is required to enable authentication.

12.4 Security

The following subsections describe how to configure security for Oracle UCM web services.

12.4.1 Configuring WS-Security through WS-Policy

Web service security (WS-Security) is set through the use of web service policies (WS-Policy). Security policies can be set to web services in order to define their security protocol. In particular, the Oracle UCM web services support OWSM policies.

Two general classes of policies are supported: username-token, and SAML. The following is a list of supported OWSM policies:

- oracle/wss11_saml_token_with_message_protection_service_policy
- oracle/wss11_username_token_with_message_protection_service_policy

To set WS-Policy

1. Access the Oracle WebLogic Server Administration Console.
2. Select **Deployments** from the side panel, then expand either the Oracle UCM native web services or the Oracle UCM generic web services.
3. Select **IdcWebLogicService** or **GenericSoapService**, then click the **Configuration** tab, and then click the **WS-Policy** tab.
4. Click the main service. From here you can choose which OWSM policies to add.
5. When you have finished adding OWSM policies, you must update the Oracle UCM native web services or the Oracle UCM generic web services.

12.4.2 Configuring SAML Support

To provide SAML support so that the client can be the identity provider (that is, assert credentials) then additional steps must be taken to configure a keystore, configure a JPS provider to use the keystore, create a client credential store (CSF), and configure a Java client to use the keystore and CSF.

12.4.2.1 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 used in the following example are the defaults for the domain's config/fmwconfig/jps-config.xml file (explained in the next section):

```
$ keytool -genkey -alias orakey -keyalg RSA -keystore default-keystore.jks
-keypass welcome -storepass welcome
```

Any relevant data can be entered in the `keytool` command, but the specifics do not matter except for the passwords for the keystore and the certificate, which the client uses.

12.4.2.2 Configuring Server JPS to Use the Keystore

Configuring the keystore on the Oracle WebLogic Server domain involves editing the $domain/config/fmwconfig/jps-config.xml file.

A provider must be defined in <serviceProviders>. A provider should be defined by default.

```
<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>
```

When you have verified the provider, or created or modified a provider, a keystore instance must be defined in <serviceInstances>. A keystore instance should be defined by default.

```
<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>
```

The location of the keystore instance must be set to the same location as when you created the keystore.

Additionally, the keystore must be added to <jpsContexts>. This setting should be in the jps-config.xml file by default.

```
<jpsContext name="default">
    <serviceInstanceRef ref="credstore"/>
    <serviceInstanceRef ref="keystore"/>
    <serviceInstanceRef ref="policystore.xml"/>
    <serviceInstanceRef ref="audit"/>
    <serviceInstanceRef ref="idstore.ldap"/>
</jpsContext>
```

12.4.2.3 Creating a Client CSF

On the client, there must be a credential store to store the keys to unlock the keystore. A Credential Store Framework (CSF) can be made in a variety of ways, but one way is to use the Oracle WebLogic Server Scripting Tool (WLST). You must use the `wlst` command from the EM interface.

To use WLST to create a credential, you must be connected to the Oracle WebLogic Server domain. Note that the resulting wallet can be used only on the client.

```
$ ./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")
```

The preceding example creates a CSF wallet at $domain/config/fmwconfig/cwallet.sso that must be given to the client. You need to change the values from the example to match the alias and passwords from the keystore you created.

12.4.2.4 Configuring a Java Client to Use the Keystore and CSF

In order to configure a Java client to use the keystore and CSF, there are two requirements:

- The Java client must have a copy of both the keystore and the CSF wallet.
- There must be a client version of the jps-config.xml file. This file must contain entries for locating the keystore as well as the CSF wallet. To configure security, the Java system property "oracle.security.jps.config" must point towards the jps-config.xml file. This can be set during execution in the client.

  ```
  System.setProperty("oracle.security.jps.config", "jps-config.xml");
  ```

The following example shows a jps-config.xml file for clients based on the configuration provided in previous examples.

```
<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>
```

13

Customizing DesktopTag

This chapter describes how to customize the DesktopTag component of Oracle Content Server to specify properties for checked out versions of Microsoft Word, Excel, and PowerPoint files.

This chapter includes the following sections:

13.1 About the DesktopTag Component

DesktopTag is an Oracle 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 Oracle 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 13.5, "Configuring the DesktopTag Component."

The custom properties provide information about where a content item resides in Oracle 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 Oracle Content Server after check-out; for example, in an external workflow (that is, one that is not managed by Oracle 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.

13.2 System Requirements

The DesktopTag component is included with Oracle Content Server 11*g*R1. It must be enabled on Oracle 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 Oracle 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 the *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*.

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

13.3 DesktopTag Component Operation

The DesktopTag component modifies the check-out (*file get*) and check-in operations for Oracle Content Server.

13.3.1 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.

13.3.2 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 result set 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.

13.4 Using the DesktopTag Component

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.

Figure 13–1 shows an example of a Word 2003 document without custom properties added by DesktopTag, and Figure 13–2 shows a number of custom properties added.

The properties that are added to the Microsoft Office files depend on the settings in the DesktopTag configuration file (see Section 13.5, "Configuring the DesktopTag Component"). In Figure 13–2, 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 Oracle Content Server again.

Figure 13–1 Word 2003 Document Without Custom Properties Added by DesktopTag

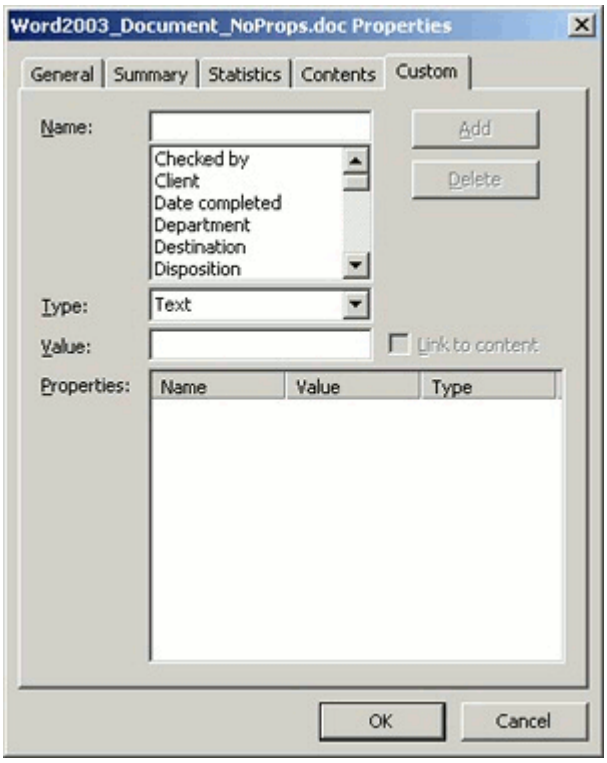
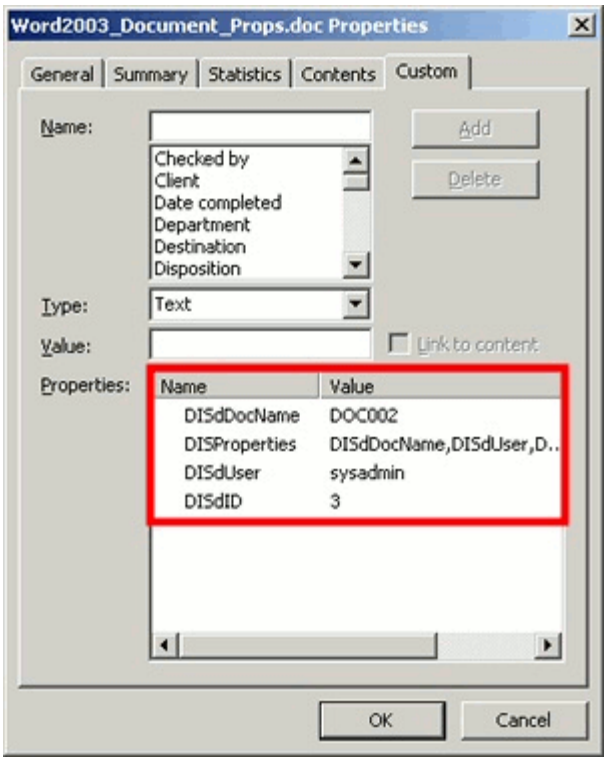


Figure 13–2 Word 2003 Document with Custom Properties Added by DesktopTag

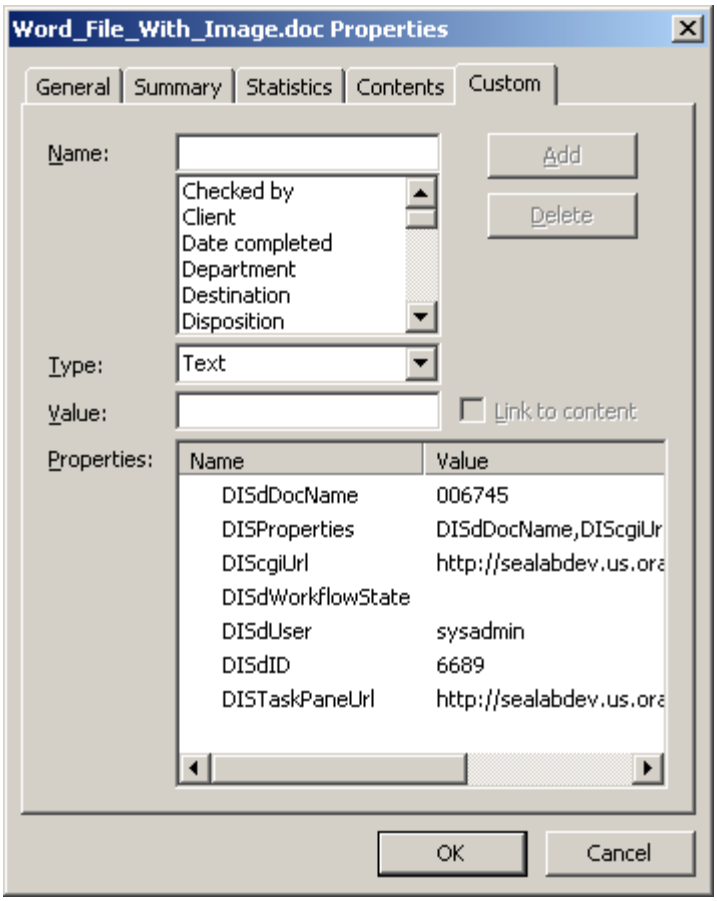


13.4.1 Viewing Custom Properties

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 13–3 Custom Document Properties (Microsoft Word 2003)



13.4.2 Checking in Documents from Outside Oracle Content Server

These custom document properties allow Desktop Integration Suite to keep track of where a managed file resides in an Oracle Content Server instance. This, in turn, enables users to check a Microsoft Office document back in to Oracle Content Server even from outside a content management integration context. 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 UCM menu or ribbon in Word. Desktop Integration Suite checks the custom properties embedded in the Word document to find out where to upload the file to.

13.5 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 Oracle 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`

13.5.1 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.

13.5.2 DesktopTagPrefix Property

The value of the `DesktopTagPrefix` property is the prefix added to the names of all standard Oracle Content Server metadata fields in the list of standard DesktopTag fields (see Section 13.5.3, "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`

13.5.3 DesktopTagFields Property

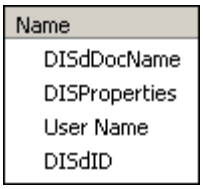
The value of the `DesktopTagFields` property is a comma-separated list of all standard Oracle Content Server metadata fields that are added to Microsoft Office files as custom properties. You should use the server-internal field names (for example, `dDocName` for the content ID). See the *Oracle Fusion Middleware Idoc Script Reference Guide* for the internal field names of the standard metadata fields.

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 13–4 shows the result of the preceding `DesktopTagFields` definition (assuming the default `DIS` prefix is used).

Figure 13–4 Example of Property Names



Note: The `DISProperties` custom property is always added. Its value is a list of all properties added by DesktopTag.

13.5.4 DesktopTagPrefixCustom Property

The value of the `DesktopTagPrefixCustom` property is the prefix added to the names of all custom Oracle Content Server metadata fields in the list of custom DesktopTag fields (see Section 13.5.4, "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`

13.5.5 DesktopTagFieldsCustom Property

The value of the `DesktopTagFieldsCustom` property is a comma-separated list of all custom Oracle 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 13.5.3, "DesktopTagFields Property").

Example: `DesktopTagFieldsCustom=xComments(Extra Info),xArchiveStatus`

Note: The standard and custom Oracle 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.

13.5.6 DesktopTagPrefixExtended Property

The value of the `DesktopTagPrefixExtended` property is the prefix added to the names of all custom Oracle Content Server metadata fields in the list of extended DesktopTag fields (see Section 13.5.7, "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`

13.5.7 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 Oracle 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`.

13.5.8 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 Oracle 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`.

13.5.9 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 Oracle Content Server log files (accessible from the server's administration pages).

Figure 13–5 DesktopTag Event in Oracle Content Server Log File

13.5.10 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`.

A

Using WSDL Generator and SOAP

This chapter describes using WSDL Generator and SOAP to manage Oracle Content Server. This feature is supported in Oracle Universal Content Management (Oracle UCM) 11g for backward compatibility.

This appendix includes the following sections:

A.1 Overview

Web Services Definition Language (WSDL) files and SOAP (Simple Object Access Protocol) can be used to manage Oracle Content Server. SOAP is a lightweight XML-based messaging protocol used to encode the information in web service request and response messages before sending them over a network. The WSDL Generator component, which is installed (enabled) by default with Oracle Content Server, allows for creating WSDLs for the services of Oracle Content Server. Users can then take the WSDLs and plug them into APIs to create web services that can be used with Oracle Content Server.

Some SOAP functionality has been built into the core Oracle Content Server. The WSDL Generator component is not essential to use SOAP; administrators can still write or call Oracle Content Server service calls in SOAP if needed. The WSDL Generator provides flexibility in altering existing client applications.

Oracle UCM has a WSDL 1.1 implementation that exposes the Oracle UCM IDCService (Internet Distributed Content Service), which in turn extends all of the capabilities of the Unversal Content Management (UCM) system. Using the IDCService, content can be checked out and checked in, workflows can be created, run and approved, content can be made available for publishing, and content can be searched by category (metadata), content (full-text), or a combination of both.

You can use WSDL files to map to Oracle UCM and the SOAP to access the content and content management functions within Oracle UCM and to deploy your content management capabilities as a web service.

A.2 Using Web Services

This section provides an overview of web services, general information about WSDL files and the SOAP protocol, and installation information.

This section covers the following topics:

- Section A.2.1, "Web Services Framework"
- Section A.2.2, "Implementation Architecture"
- Section A.2.3, "Implementation on .NET"
- Section A.2.4, "The SOAP Protocol"

A.2.1 Web Services Framework

Web services reside as a layer on top of existing software systems such as application servers, .NET servers, and Oracle Content Server. Web services can be used as a bridge to dissimilar operating systems or programming languages. Web services are adapted to the Internet as the model for communication and rely on the HyperText Transfer Protocol (HTTP) as the default network protocol. Thus, using web services, you can build applications using a combination of components.

The core enabling technologies for web services are XML Data, WSDL Interface, SOAP Communication, and UDDI Registry. This section provides a general overview of the technologies in these sections:

- Section A.2.1.1, "XML Data"
- Section A.2.1.2, "WSDL Interface"
- Section A.2.1.3, "SOAP Communication"
- Section A.2.1.4, "UDDI Registry"
- Section A.2.1.5, "DIME: Message Format"
- Section A.2.1.6, "How the Enabling Technologies Work Together"

A.2.1.1 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 differing 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. It is XML that provides the capability to model data and define the structure specific to the programming language (for example, Java, C#, or Visual Basic), the database management system, or 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.

A.2.1.2 WSDL Interface

The Web Services Description Language (WSDL) provides the interface that is exposed to web services. It is the WSDL layer that 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.

A.2.1.3 SOAP Communication

The Simple Object Access Protocol (SOAP) provides communication for web services interfaces to communicate to each other over a network. SOAP is an XML-based communication protocol used to access web services. Web services receive requests and return responses using SOAP packets encapsulated within an XML document.

A.2.1.4 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 registration and lookup of web services.

Oracle Content Server currently does not publish to any public or private UDDI sources. However this does not prevent users from integrating Oracle UCM with other applications using web services.

A.2.1.5 DIME: Message Format

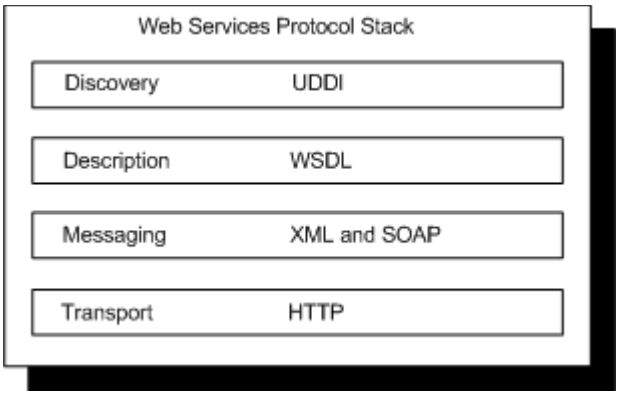
DIME is a lightweight, binary message format that can be used to encapsulate one or more application-defined groups of arbitrary type and size into a single message construct. This format can be used when uploading or downloading content. The payloads consist of the SOAP message and one or more groups of file content.

A.2.1.6 How the Enabling Technologies Work Together

The XML, WSDL, SOAP, and UDDI technologies work together as layers on the web services protocol stack. The web services protocol stack consists of these layers.

- The service *transport* layer between applications (HTTP).
- The *messaging* layer that provides a common communication method (XML and SOAP).
- The service *description* layer that describes the public interface to a specific web service (WSDL).
- The service *discovery* layer that provides registry and repository services for storing and retrieving web services interfaces (UDDI).

Figure A–1 Web Service Protocol Stack



Note: While several protocols are available as a transport layer (for example, 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.

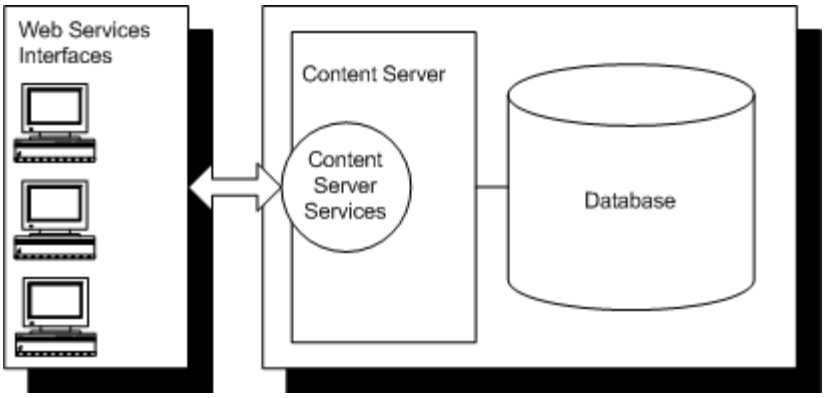
To help grasp the connection between these technologies, consider this simple analogy: Think of HTTP as the telephone wire (transport between applications) and UDDI as a telephone book (where a developer can browse a UDDI registry to locate a registered service). SOAP could be described as the voices 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 as it includes information such as the available functions and data types).

A.2.2 Implementation Architecture

Web services are not executable, but rather exchange data within the development environment. Thus, 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.

The following scenario provides a web services implementation architecture for the Oracle Content Server application. The primary value of this architecture remains in the features and functions of Oracle Content Server. Web services access Oracle Content Server through the WSDL Generator component and use the exposed Oracle Content Server services to execute actions and provide data transaction between the user employing web services and Oracle Content Server.

Figure A–2 Web Service Implementation Architecture



A.2.3 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 a 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 and consists of a common language run-time, unified class libraries, and includes the ASP .NET server.
- The Visual Studio .NET product provides the 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 easily create and deploy the program as a web service.

A.2.4 The SOAP Protocol

Employing a SOAP integration provides a standardized interface for executing Oracle UCM services using the Java API (IdcCommand) and provides XML and non-XML content managed by Oracle Content Server.

Because SOAP uses the Hypertext Transfer Protocol (HTTP) for data transmission, it can be invoked across the Web and enable content to be accessible over a network in a platform-independent and language-neutral way.

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

Using SOAP to access content management capabilities as a web service enables real-time programmatic interaction between applications and enables the integration of business processes and facilitates information exchange.

Web services are modular components that are contained in an XML wrapper and defined by the Web Services Description Language (WSDL) specifications. The Universal Description Discovery and Integration (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 the SOAP packet is sending a Remote Procedure Call (RPC). The default is to evaluate SOAP messages as document-style SOAP messages, rather than Remote Procedure Call (RPC) style SOAP messages. Using the SOAP Toolkit client with a .NET developed web service returns an error reading 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.

A.3 SOAP Clients

You can use SOAP to access the content and content management functions within Oracle Content Server and to deploy your content management capabilities as a web service.

Note: If you are developing SOAP client implementations, make sure that *chunking* is disabled in your client API code.

A.3.1 Using the Java SOAP Client

These are the command line parameters used as arguments for the Java programs.

Parameters	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 Oracle Content Server.
`-p <primary file>`	The filename to upload as the primary file.
`-a <alternate file>`	The filename to upload as the alternate file (optional).
`-l <log file>`	The filename containing the request and response data (optional).

A.4 SOAP Service Calls

This section discusses executing various Oracle Content Server IdcCommand services using the SOAP interface. Oracle Content Server IdcCommand services can be executed using the SOAP interface. The user must have appropriate permissions to execute the commands. Some commands require administrative access, other commands may require only write permission.

The WSDL Generator component is installed (enabled) by default with Oracle Content Server, and it must remain enabled to call services. See the *Oracle Fusion Middleware Services Reference Guide* for a list of available services and the required parameters.

This section covers these topics:

- Section A.4.1, "SOAP Packet Format"
- Section A.4.2, "Special Characters"

A.4.1 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.

A.4.1.1 HTTP Headers

This entry is required in a SOAP request HTTP header:

```
Content-Type: text/xml; charset="utf-8"
```

This SOAPAction header is suggested, but not required:

```
SOAPAction: "http://www.oracle.com/IdcService"
```

A.4.1.2 Namespaces

Within the body of a SOAP message the SOAP message XML namespaces are used to qualify element and attribute names within the parts of the document. Element names can be global (referenced throughout the SOAP message) or local. The local element names are provided by namespaces and are used in the particular part of the message where they are located. Thus, SOAP messages use namespaces to qualify element names in the separate parts of a message. Application specific namespaces are used to qualify application specific element names. Namespaces also identify the envelope version and encoding style.

Oracle Content Server defines a namespace called `idc` that explains the schema and allowable tags for the SOAP content.

A.4.1.3 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. Thus, 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 may be required for a service request or may be returned in the response:

- Service Node
- Document Node
- User Node
- Optionlist Node
- Option Subnode
- Resultset Subnode
- Row Subnode
- Field Subnode

Note: On requests, Oracle 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.

A.4.1.3.1 Service Node This is the main node of the IDC namespace.

- This node must exist to process a request.
- The required attribute *IdcService* defines the service you are requesting.
- It is not required that the subnodes of <service> carry the namespace in their tags. For example, <document> can be used rather than <idc:document>. However if you do define the namespace identifier in the child nodes, it must match the one specified in the service tag.

For example:

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="PING_
SERVER">
</idc:service>
```

A.4.1.3.2 Document Node This 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 Oracle Content Server instance. For example, *dID*, *dDocTitle*, and *dDocType* are common attributes.

- Custom content item information such as *xSpec* is valid if defined as metadata.
- All known document fields can be used as attributes.

In the following document node example, the CHECKOUT_BY_NAME service is used:

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="CHECKOUT_
BY_NAME">
<idc:document dDocName="soap_sample">
</idc:document>
</idc:service>
```

A.4.1.3.3 User Node This is the node to contain all user information:

- Attributes that are valid for users are defined by your specific Oracle Content Server instance. For example, *dName*, *dFullName*, and *dEmail* are common attributes.
- Custom user information is valid if defined as metadata.
- All known user fields may be used as attributes.

For example:

```
<idc:service xmlns:idc="http://www.oracle.com/IdcService/" IdcService="GET_USER_
INFO">
<idc:user dUser="sysadmin">
</idc:user>
</idc:service>
```

A.4.1.3.4 Optionlist Node This is the node to contain any option lists. The *name* attribute specifies the name of the option list. Each option subnode contains a value in the optionlist node.

For example:

```
<idc:optionlist name="Users_UserLocaleList">
<idc:option>
English-US
</idc:option>
</idc:optionlist>
```

A.4.1.3.5 Option Subnode This subnode is specified within the <optionlist> node. The *option* attribute specifies the name of the option for the option list.

For example:

```
<idc:optionlist name="dDocType">
<idc:option>ADACCT</idc:option>
<idc:option>ADHR</idc:option>
<idc:option>ADSALES</idc:option>
</idc:optionlist>
```

A.4.1.3.6 Resultset Subnode This subnode can be specified within a <document> or <user> node.

- This subnode contains result set information in a request or response.
- The *name* attribute specifies the name of the result set.

For example:

```
<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>
```

A.4.1.3.7 Row Subnode This subnode is specified within a <resultset> subnode.

- This subnode can appear multiple times within <resultset> and specifies each row in the result set.
- Attributes that are valid are defined by your specific Oracle Content Server instance. These are the same fields that can appear as attributes in a document or user node.

For example:

```
<idc:resultset name="UserAttribInfo">
<idc:row dUserName="jsmith" AttributeInfo="role,contributor,15">
</idc:row>
</idc:resultset>
```

A.4.1.3.8 Field Subnode This subnode can be specified within a <document>, <user>, or <row> node. The *name* attribute specifies the name of the field.

It often represents data such as *refreshSubjects* or *dSubscriptionID*.

For example:

```
<idc:field name="dSubscriptionID">
stellent
</idc:field>
```

- It may represent document or user metadata that is user configurable or custom metadata such as *xComments*.
- It is used to pass search result values such as *QueryText* and *OriginalQueryText*. For example:

```
<idc:field name="QueryText">
dDocType <Substring> "ADSALES&"
</idc:field>
```

A.4.2 Special Characters

When passing special characters, such as a left angle bracket (<) or right angle bracket (>), to Oracle UCM, you must use the XML-encoding format.

Standard Format	XML-Encoding
<	&lt;
>	&gt;
"	" ` (use back quote if using universal query syntax)
&	&amp;
\	&#39;

Note: Some search result values, such as the *QueryText* and *OriginalQueryText* values, are URL-encoded in the response.

This example passes a string submitted for a Oracle Content Server content item query (using universal query syntax) as both a standard formatted string and an 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>
```

A.5 Using Active Server Pages

You can execute Oracle 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.

This section covers these topics:

- Section A.5.1, "Sample SOAP Request"
- Section A.5.2, "Sample Active Server Page"

  For information about formatting XML-encoded strings, see Section A.4.2, "Special Characters."

A.5.1 Sample SOAP Request

This section provides an example Active Server Page that calls a service from Oracle Content Server. A description of the service is provided including the required and optional parameters. This section also provides an XML-formatted version of the embedded SOAP request.

For more information about service calls, including required and optional parameters, see Section A.8, "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>
```

A.5.2 Sample Active Server Page

The embedded SOAP request forms the basis of the Active Server Page. The following sample executes the GET_SEARCH_RESULTS.

For more information about service calls and examples of SOAP response/request messages, see Section A.8, "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
'----------------------------
%>
```

A.6 Using WSDL Files

This section provides an overview of the Oracle Content Server WSDL files. In addition, information about generating WSDL files for interfacing with Oracle UCM services is provided.

This section covers these topics:

- Section A.6.1, "Understanding WSDL Files"
- Section A.6.2, "Sample WSDL File"
- Section A.6.3, "Generating WSDL Files"
- Section A.6.4, "Generating Proxy Class from WSDL Files"

A.6.1 Understanding WSDL Files

WSDL files provide the ability to pass data that can be understood by Oracle Content Server services, which enables access to the content and content management functions within Oracle UCM. 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 A.6.2, "Sample WSDL File," for additional information.

A.6.1.1 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 code fragments in this section are from the DocInfo.wsdl file provided with the WSDL Generator component. For a complete WSDL file, see Section A.6.2, "Sample WSDL File."

A.6.1.1.1 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>
```

A.6.1.1.2 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>
```

A.6.1.1.3 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.

A.6.1.1.4 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>
```

A.6.1.1.5 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 an Oracle Content Server browser window to view the underlying SOAP code for that page.

A.6.2 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 Oracle 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>
```

A.6.3 Generating WSDL Files

When the WSDL Generator component is installed and enabled during Oracle Content Server 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 A.7, "Creating a Custom WSDL Using Administration Pages."

A.6.4 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 ErrorMessage.

Note: In addition to the WSDL files provided with the WSDL Generator component, you can generate WSDL files for any Oracle UCM service. For more information, see Section A.6.3, "Generating WSDL Files."

A.7 Creating a Custom WSDL Using Administration Pages

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 Oracle UCM service. See Section A.6.3, "Generating WSDL Files," for additional information.

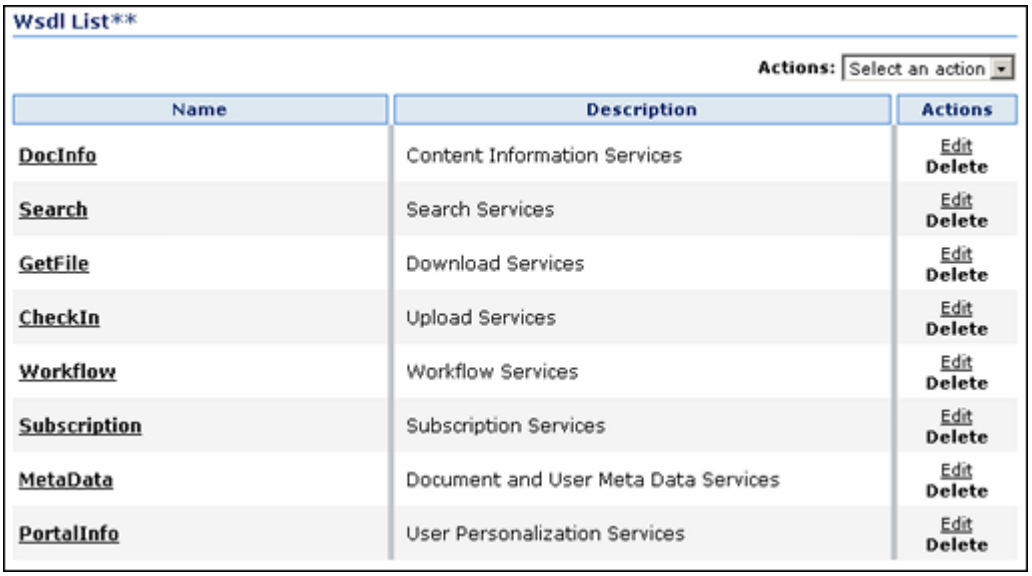
See the *Oracle Fusion Middleware Services Reference Guide* for a list of available services and the required parameters.

This section provides step-by-step instructions on creating and editing a custom WSDL using the Soap Custom WSDL administration pages.

1. In a web browser, log in to Oracle Content Server as an administrator.
2. In the **Administration** tray, choose **Soap Wsdls**.

   The Wsdl List page is displayed.

Figure A–3 Wsdl List Page

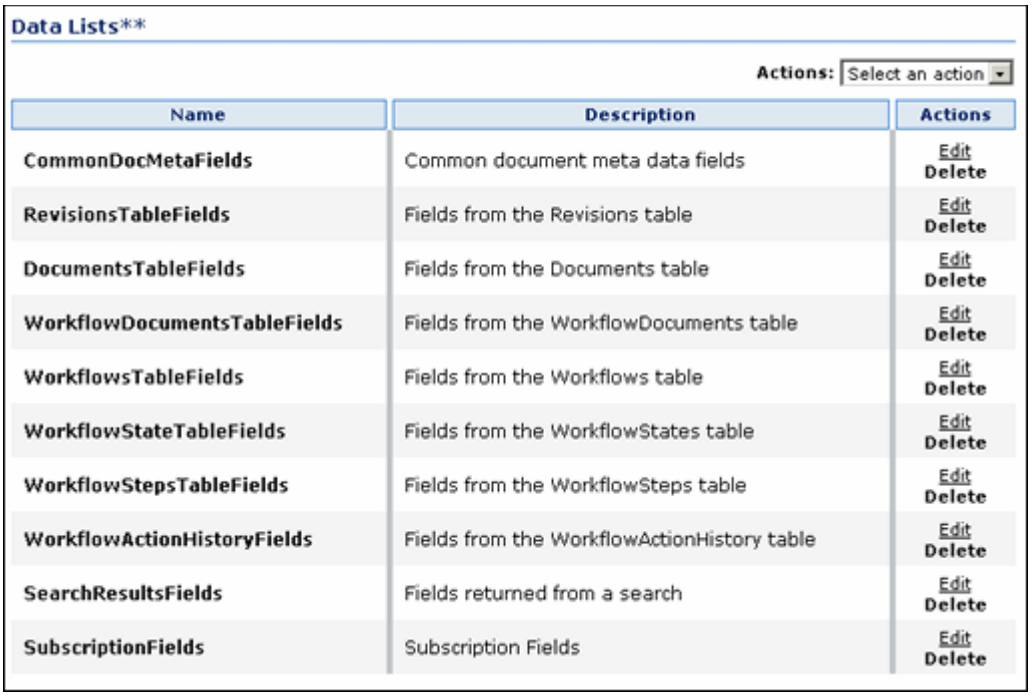


3. Click **Data Lists** from the Action menu.

   The Data Lists page is displayed.

   Data Lists are global lists of data that can be used with complex types, service parameters, or other DataLists. When a DataList is specified as a parameter or a subtype of a complex type, all the subtypes of the DataList will appear as data types. DataLists are defined once but can be referenced multiple times with different WSDLs and services. All the DataLists have a prefix of "d:" in the data type list.

Figure A–4 DataLists Page



Note: System-specific WSDLs cannot be deleted. You can, however, edit the WSDL and enable or disable the complex type elements for that WSDL.

4. Select **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**

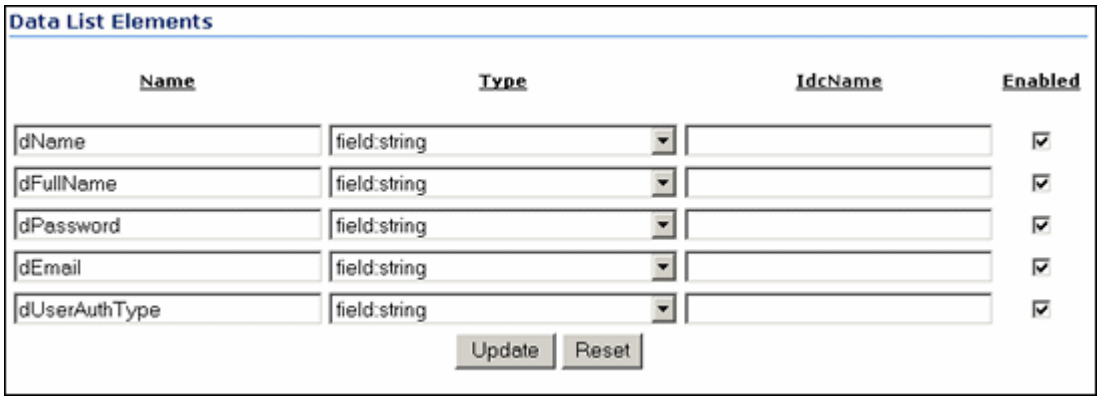
6. Click **Add**.

   The Data List Information / Data List Elements page is displayed.

7. Enter the following Data Elements information, selecting the Type from the menu.

Name	Type	Idc Name
dName	field:string	
dFullName	field:string	
dPassword	field:string	
dEmail	field:string	
dUserAuthType	field:string	

Figure A–5 DataList Elements



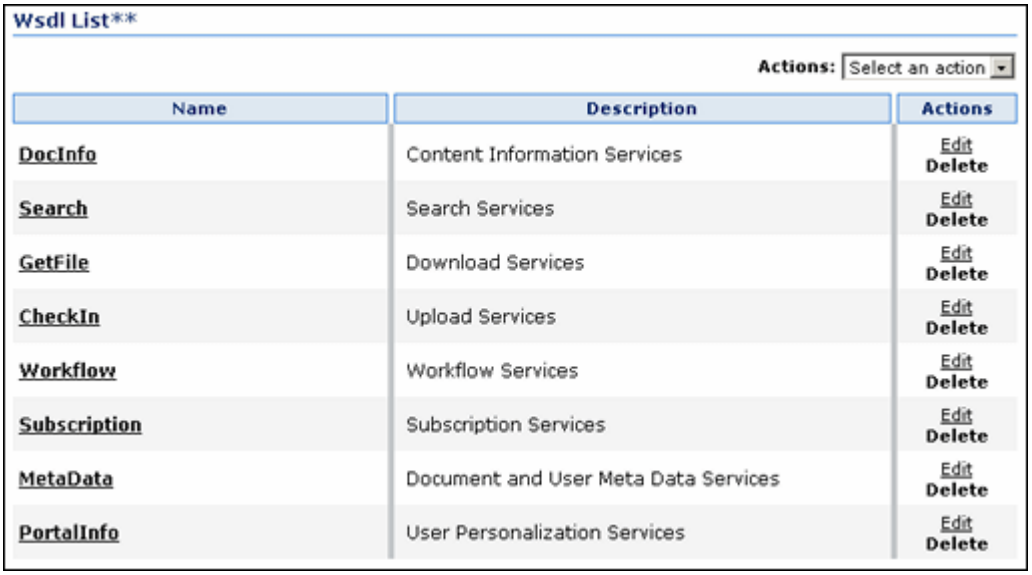
8. Click **Update**.

   You are returned to the updated Data Lists page. Note that `UserMetaFields` now appears at the bottom of the list.

9. Select **Wsdls List** from the Actions menu.

   The Wsdl List page is displayed.

Figure A–6 Wsdl List Page Redisplayed



10. Select **Add Wsdl** from the Actions menu.

    The Add Wsdl page is displayed.

11. Enter the following information:

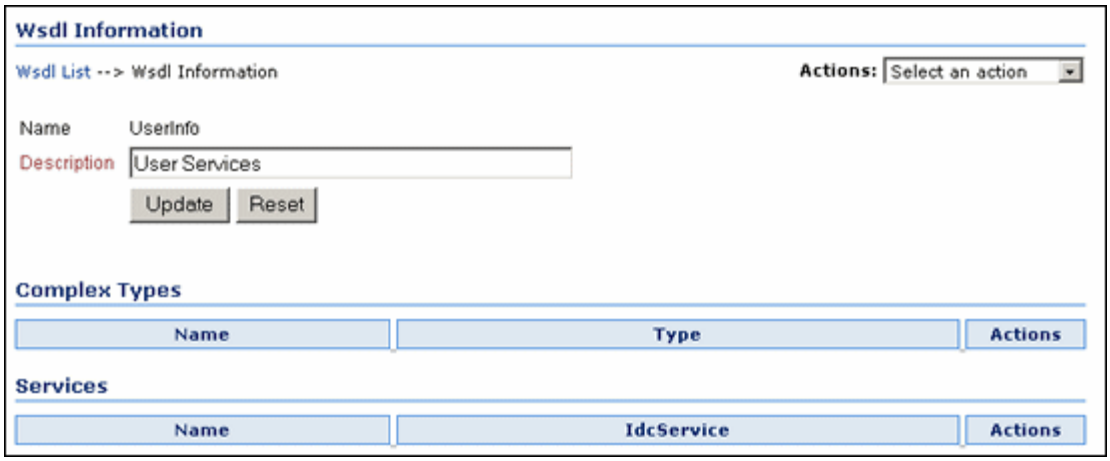
    Name: **UserInfo**

    Description: **User Services**

12. Click **Add**.

    The Wsdl Information page is displayed.

Figure A–7 Wsdl Information Page



13. Select **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:

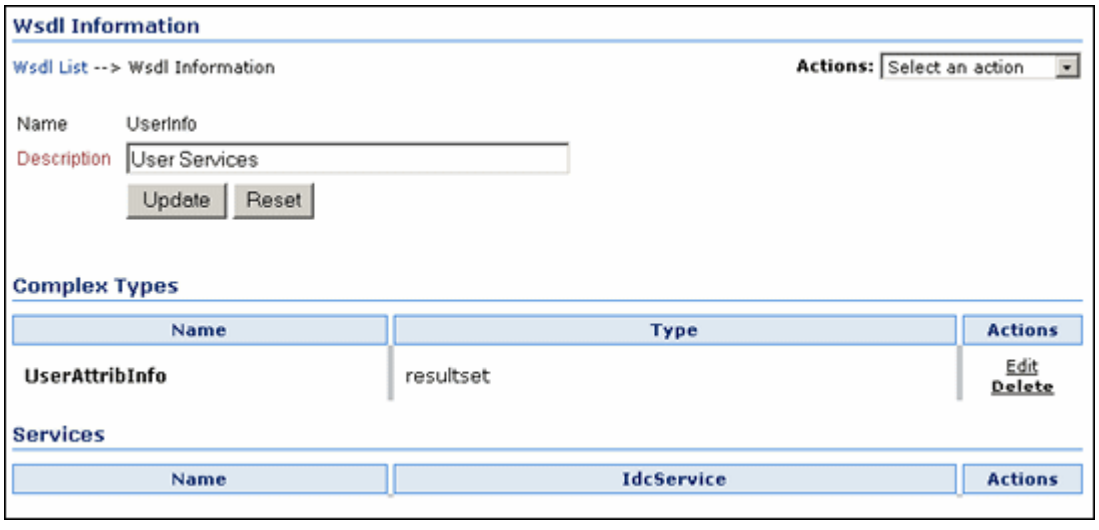
    Name: **UserAttribInfo**

    Type: select **resultset** from the menu

15. Click **Add**.

    The Wsdl Information page is displayed.

Figure A–8 Wsdl Information Page Redisplayed



16. Select **Edit** from the UserAttribInfo line.

    The Complex Type Information/Complex Type Elements page is displayed.

17. Enter the following Complex Type Elements, selecting the Type 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. Select **Add Service** from the Actions menu.

    The Add Service page is displayed.

20. Enter the following information:

    Name: **AddUser**

    IdcService: **ADD_USER**

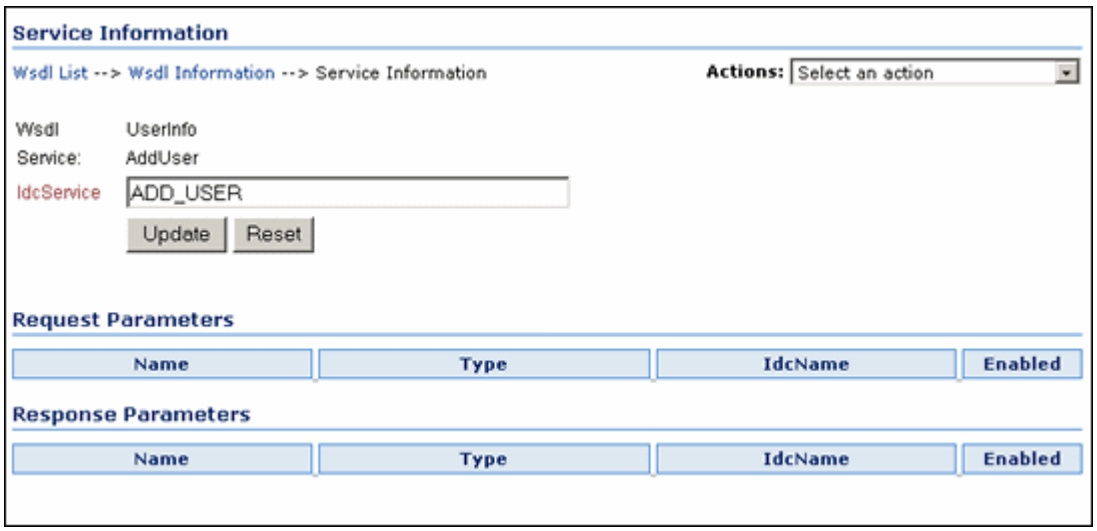
21. Click **Add**.

    The Wsdl Information page is displayed.

22. Select **Edit** from the AddUser Service line.

    The Service Information page is displayed.

Figure A–9 Service Information Page



Note: When you create a WSDL, you create services that correspond to the IdcServices feature of Oracle 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. Select **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. Select **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 A–1).

Example A–1 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 a .wsdl file extension rather than the default .xml file extension.

A.8 Sample Service Calls with SOAP Response/Request

This section provides sample service calls with SOAP response/request and presents information about executing Oracle Content Server services in a SOAP request. See the *Oracle Fusion Middleware Services Reference Guide* for a list of available services and the required parameters.

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 A.8.1, "Ping the Server,".
ADD_USER	This service adds a new user to the system. See "Add a New User" on page A-29.
EDIT_USER	This service edits an existing user. See "Edit Existing User" on page A-32.
GET_USER_INFO	This service retrieves the user list. See "Get User Information" on page A-35.
DELETE_USER	This service deletes an existing user. See "Delete User" on page A-36.
CHECKIN_UNIVERSAL	This service performs an Oracle Content Server controlled check in. See "Check in Content Item" on page A-38.
CHECKOUT_BY_NAME	This service marks the latest revision of the specified content item as locked. See "Check out Content Item" on page A-42.
UNDO_CHECKOUT_BY_NAME	This service reverses a content item checkout using the Content ID. See "Undo Content Item Checkout" on page A-44.
DOC_INFO	This service retrieves content item revision information. See "Get Content Item Information" on page A-46.
GET_FILE	This service retrieves a copy of a content item without performing a check out. See "Get File" on page A-48.

IdcCommand	Description
GET_SEARCH_RESULTS	This service retrieves the search results for the passed query text. See "Get Search Results" on page A-51.
GET_TABLE	This service exports the specified table in Oracle Content Server database. See "Get Table Data" on page A-54.
GET_CRITERIA_WORKFLOWS_FOR_GROUP	This service returns criteria workflow information. See "Get Criteria Workflow Information" on page A-56.

A.8.1 Ping the Server

The PING_SERVER service evaluates whether a connection to the server exists.

- This service returns status information for Oracle 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 the Oracle Content Server instance and that you are logged in as a user authorized to execute commands.

A.8.1.1 Required Parameters

These parameters must be specified.

Parameter	Description
IdcService	Must be set to PING_SERVER.

A.8.1.2 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
```

A.8.1.3 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>
```

A.8.2 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.

A.8.2.1 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.

A.8.2.2 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.

A.8.2.3 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 pre-defined. 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 users rights to the named account, the information is in this form:

```
account,books,1
```

A.8.2.4 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>
```

A.8.2.5 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>
```

A.8.3 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.

A.8.3.1 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.

A.8.3.2 Optional Parameters

These optional parameters may 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.

A.8.3.3 Optional Attribute Information

A result set containing the user's attribute information and referencing 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 pre-defined. 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 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 users rights to the named account, the information is in this form:

  ```
  account,books,1
  ```

A.8.3.4 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>
```

A.8.3.5 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>
```

A.8.4 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.*

A.8.4.1 Required Parameters

These parameters must be specified.

Parameter	Description
dUser	The defined user.
IdcService	Must be set to GET_USER_INFO.

A.8.4.2 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>
```

A.8.4.3 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>
```

A.8.5 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.

A.8.5.1 Required Parameters

These parameters must be specified.

Parameter	Description
dName	The unique name.
IdcService	Must be set to DELETE_USER.

A.8.5.2 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>
```

A.8.5.3 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>
```

A.8.6 Check in Content Item

The CHECKIN_UNIVERSAL service performs an Oracle Content Server controlled check-in.

- 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 an Oracle Content Server controlled check-in. 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 an Oracle Content Server controlled check-in. 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.

A.8.6.1 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, the content 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.

A.8.6.2 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.

A.8.6.3 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, the content 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

A.8.6.4 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>
```

A.8.6.5 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>
```

A.8.7 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.

A.8.7.1 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.

A.8.7.2 Optional Parameters

This optional parameter may be specified.

Parameter	Description
dDocTitle	The content item title.

A.8.7.3 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>
```

A.8.7.4 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>
```

A.8.8 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}''.`

A.8.8.1 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.

A.8.8.2 Optional Parameters

This optional parameter may be specified.

Parameter	Description
dDocTitle	The content item title.

A.8.8.3 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>
```

A.8.8.4 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>
```

A.8.9 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.

A.8.9.1 Required Parameters

These parameters must be specified.

Parameter	Description
dID	The generated content item revision ID.
IdcService	Must be set to DOC_INFO.

A.8.9.2 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>
```

A.8.9.3 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>
```

A.8.10 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 filename 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 filename 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 Oracle Content Server are based on the assumption that it is present, as are other features, such as forms.

A.8.10.1 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 may 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*.

Parameter	Description
RevisionSelection Method	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 may be *Specific*, *Latest*, or *LatestReleased*. ■ If the value is *Specific*, the dDocName is ignored, and dID is required and 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`.

A.8.10.2 Optional Parameters

These 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.

A.8.10.3 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>
```

A.8.10.4 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--
```

A.8.11 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, on 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 as both a standard formatted string and XML-encoded format:

  – Parameter with standard formatted string.

    ```
    QueryText=dDocType <Substring> "ADSALES"
    ```

  – Parameter with XML-encoded string

    ```
    <idc:field name="QueryText">
    dDocType <Substring> `ADSALES`
    </idc:field>
    ```

    For more information about formatting XML-encoded strings, see Section A.4.2, "Special Characters."

- If this service is unable to execute, this message is displayed to the user: *Unable to retrieve search results.*

A.8.11.1 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.

A.8.11.2 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.

A.8.11.3 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>
```

A.8.11.4 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>
```

A.8.12 Get Table Data

The GET_TABLE service exports the specified table in the Oracle Content Server database.

- Exports the specified table by creating a result set and adding it to the serialized *hda*. If the table is not found, the service will fail. It is up to the calling program receiving the serialized *hda* to store this result set for later usage.
- 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.

A.8.12.1 Required Parameters

These parameters must be specified.

Parameter	Description
IdcService	Must be set to GET_TABLE.
tableName	The name of table to export.

A.8.12.2 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>
```

A.8.12.3 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>
```

A.8.13 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 result sets 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.

A.8.13.1 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.

A.8.13.2 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>
```

A.8.13.3 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>
```

Index

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