# Oracle® Fusion Middleware

Developer's Guide for Content Server

11*g* Release 1 (11.1.1)

**E10807-01**

May 2010

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

E10807-01

Primary Author: Karen Johnson

Contributing Author: Sandra Christiansen, Jean Wilson

Contributor: Eva Cordes, Daniel Lew, Rick Petty, Peter Walters, Sam White

# Contents

# Preface

While Content Server 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 content server instance.

## Audience

This guide is intended for developers and administrators who want to customize Content Server 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 Modifying Your Content Server" provides an introduction to the methods and tools you can use to customize Content Server.
- Chapter 2, "Content Server Architecture" describes the architecture of Content Server and how that affects the customization you can make.
- Chapter 3, "Working with Components" describes how to use components to modify or add functionality to Content Server.
- Chapter 4, "Changing the Look and Navigation of the Content Server Interface" defines the items you can adjust to change the look and navigation of the Content Server interface.
- Chapter 5, "Modifying System Functionality" describes how you can change the functionality of Content Server with system settings, components, and configuration variables.
- Chapter 6, "Integration Methods" provides information about integrating the Content Server with enterprise applications such as application servers, catalog solutions, and enterprise portals.
- Chapter 7, "Using Oracle UCM Web Services" discusses using Web Services and SOAP (Simple Object Access Protocol) to manage 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 Content Server 11*g* Release 1 (11.1.1) documentation set:

- *Oracle Fusion Middleware System Administrator's Guide for 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. |

# What's New

This section introduces the new and changed features of Oracle Universal Content Management (UCM) Content Server developer information 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 Content Server release 10g documents:
  - *Getting Started with the Stellent SDK*
  - *Modifying the Content Server Interface*
  - *Working with Components*
  - *Idc Command Reference Guide*
  - *Dynamic Server Pages Guide*
  - *Using WSDL Generator and SOAP*
- **Web services:** Oracle UCM uses Oracle WebLogic Server Web Services. See Chapter 7, "Using Oracle UCM Web Services".
- **ComponentTool:** The ComponentTool utility has been added to provide a command-line tool for installing, enabling, and disabling components. See Chapter 3, "Working with Components".
- **Content Server deployment:** Content Server is deployed on an Oracle WebLogic Server in the Oracle Enterprise Content Management Suite, which means changes in configuring and administering Oracle UCM. For more information, see *Oracle Fusion Middleware System Administrator's Guide for 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 (including Oracle Enterprise Content Management Suite and Oracle WebLogic Server), and the deployed Oracle UCM directories and files. The directory structure for an Oracle UCM 11*g* instance is different from a 10*g* Release instance.The following terms and pathnames are important to understanding and working with the Oracle UCM structure:
  - *IdcHomeDir*: This variable is used to refer to the directory in *ECM_ORACLE_HOME* where the Oracle UCM (`ucm`) server media is located. The server

media can run Content Server, Inbound Refinery, or Universal Records Management.

  - *DomainHome*: This variable is used to refer 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.
  - *short-product-id*: This variable is used to refer to the type of Oracle UCM server deployed on an Oracle WebLogic Server. Possible values include:
    * `cs` (Content Server)
    * `ibr` (Inbound Refinery)
    * `urm` (Universal Records Management)
  - *IntradocDir:* This variable is used to refer to the root directory for configuration and data files specific to a Content Server instance deployed on an Oracle UCM domain on an Oracle WebLogic Server. This Idoc Script variable is configured for one type of Content Server instance: Content Server, or Inbound Refinery, or Universal Records Management. This directory can be located elsewhere, but the default location is *DomainHome*/ucm/*short-product-id*/.

- **SOAP**: SOAP is provided with Oracle WebLogic Server, not in Oracle UCM.
- **WSDL Generator:** The WSDL Generator is not supported.
- **Web Form Editor:** The Web Form Editor user interface and FCKEditor are not supported.

1

# Introduction to Modifying Your Content Server

This chapter provides an overview of Content Server customization and describes the tools you need and the resources that are available. It includes the following sections:

- "Customization Types" on page 1-1
- "Customization Planning" on page 1-1
- "Recommended Skills and Tools" on page 1-2
- "Troubleshooting" on page 1-3

## 1.1 Customization Types

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

- **Altering the look and feel of the product:** You can customize the look-and-feel of the Content Server interface to meet your organization's specifications. Interface modifications can be as simple as replacing the icons that appear on the standard Content Server Web pages or as complex as a complete redesign of the interface.
- **Modifying the functionality of the product:** By changing how the product functions, you can tailor Content Server to the way your business or organization functions. For example, you can change the default date and time stamp, or change aspects of check-in behavior.
- **Integrating the product into your environment**: You can use shell scripts, SOAP, J2EE, JSP, and clusters to more fully integrate Content Server into your site's current environment.

## 1.2 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 the 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 the 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 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 Content Server 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

Content Server 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 skills you may need to modify the Content Server:

- **Content Server Architecture—**You should thoroughly understand how the Content Server works and how components and dynamic server pages function before you begin customizing your system.
- **HTML/CSS—**You'll need a good understanding of HTML and cascading style sheets (CSS) to make changes to the Content Server Web page templates. The templates are not complex in their use of HTML, but they make constant use of HTML tables and frequent use of forms. The *std_page.idoc* and *std_css.idoc* files include cascading style sheets to control the look-and-feel of the default templates, including fonts and layout.
- **Idoc Script—**Idoc Script is the custom server-side scripting language for Content Server. Almost every Content Server Web page includes some Idoc Script, which provides the methods for processing various page elements.

- **JavaScript—**The internal content of most Content Server pages do not use JavaScript, but the Search, Checkin, 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—**The 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—**The Content Server is implemented with Java classes. You should have a thorough understanding of Java and the Content Server Java class files before attempting to make any changes to the underlying functionality. However, the product can be customized extensively without having to work 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 your content management system with other systems.

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

- **Text Editor—**Most product customizing can be done with a normal text editor such as Microsoft WordPad or vi.
- **HTML Editor—**If you prefer to use a graphical HTML editor to work with HTML pages, use caution—such programs often change the source HTML, and may cause Idoc Script tags to be converted into a string of characters that are no longer be recognized by the Content Server. If you use a graphical editor, make sure you edit in a non-graphical mode.
- **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 Content Server pages as they are used. This section discusses three broad types of troubleshooting aids:

- "Viewing Server Errors" on page 1-4
- "Viewing Page Data" on page 1-4
- "Monitoring Resource Loading" on page 1-4

### 1.4.1 Viewing Server Errors

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

- If you run the Content Server from a command prompt, you can view the error in the console window.
- If you can log in to the Content Server and display the Admin Server page, you can view the Content Server log by selecting the Content Server and then clicking the View Server Output link.
- You can view the 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 a Content Server Web page.

- In a Web browser, add the following code in the Address box to the end of the page's URL:

  ```
  &IsJava=1
  ```

- On a template page or in an include, use the following code:

  ```
  <$IsJava=1$>
  ```

The *IsDebugTrace* setting displays a tree structure view of all includes being called on a Content Server Web page. The debug trace appears at the bottom of the Web page.

- In a Web browser, add the following code in the Address box to the end of the page's URL:

  ```
  &IsDebugTrace=1
  ```

- On a template page or in an include, use the following code:

  ```
  <$IsDebugTrace=1$>
  ```

- 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 `IsDebugTrace` 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 the 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 is an example of 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 Content Server Architecture

To create a customization efficiently and effectively, you should have an understanding of how Content Server works. This chapter describes the architecture of Content Server in the context of what you need to know before beginning a customization project. It includes the following sections:

- "Content Server Directories and Files" on page 2-1
- "Resources" on page 2-5
- "Content Server Behavior" on page 2-6

## 2.1 Content Server Directories and Files

When you create custom components or dynamic server pages, you work primarily with files in certain directories. This section includes the following topics:

- "Terminology for UCM Directories" on page 2-1
- "bin Directory" on page 2-2
- "config Directory" on page 2-3
- "components Directory" on page 2-4
- "resources/core Directory" on page 2-4
- "weblayout Directory" on page 2-5

**Caution:** Modifying the default variables in these files can cause the content server to malfunction. See the *Oracle Fusion Middleware Idoc Script Reference Guide* for more information about configuration variables.

### 2.1.1 Terminology for UCM Directories

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

- *IdcHomeDir*: The variable for the directory in $ORACLE_HOME where the Oracle UCM (`ucm`) server media is located. The server media can run Content Server, Inbound Refinery, or Universal Records Management.
- *DomainHome*: The variable for the user-specified directory where the Oracle UCM server is deployed to run inside the Oracle WebLogic Server application server. The *DomainHome*/ucm/*short-product-id*/bin directory contains the intradoc.cfg file and executables.

- *short-product-id*: An abbreviated name for the type of Oracle UCM server deployed on an Oracle WebLogic Server, used as the context root (default HttpRelativeWebRoot configuration value). Possible values are:
  - `cs`—Content Server
  - `ibr`—Inbound Refinery
  - `urm`—Records Manager
- *IntradocDir*: The variable for the root directory for configuration and data files specific to a Content Server instance. The default location for this directory is *DomainHome*/ucm/*short-product-id*.

### 2.1.2 bin Directory

The bin directory is the root directory for content server startup files. It contains the intradoc.cfg file and the executable files that run the content server services, applets, and utilities. It is located at *DomainHome*/ucm/*short-product-id*/bin, where *short-product-id* specifies whether it is for Content Server, Inbound Refinery, or Records Manager.

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

**Note:** If the content server is set up as an automatic service and you attempt to start a content server service (IdcServer or IdcServerNT) from the command line, you 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 the content server, including directory, Internet, and refinery settings. Several of these variables can be set using the content server's System Properties utility.

The following is a typical intradoc.cfg file:

```
<?cfg jcharset="Cp1252"?>
#Content Server Directory Variables
IntradocDir=C:/oracle/idcm1/
WebBrowserPath=C:/Program Files/Internet Explorer/iexplore.exe
```

```
CLASSPATH=$COMPUTEDCLASSPATH;$SHAREDDIR/classes/jtds.jar

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

### 2.1.3 config Directory

The config directory stores global content server configuration information. 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 content server system. Several of these variables can be set using the content server's System Properties utility or by modifying the variables on the Admin Server General Configuration page.

The following is a typical config.cfg file:

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

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

#Internet Variables
HttpServerAddress=jwilsonnote
MailServer=mail.company.com
SysAdminAddress=sysadmin@company.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 components Directory

The *IntradocDir*/data/components directory contains the files that the content server uses to configure system components:

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

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 resources/core Directory

The *IdcHomeDir*/resources directory contains two directories: admin and core. The resources/core directory contains files that the content server uses to assemble Web pages:

| **Element** | **Description** |
|---|---|
| config | Holds base config files for the 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 the content server. |
| reports | Holds templates for content server reports. |

| Element | Description |
|---|---|
| resources | Holds resource definitions (queries, page resources, services, and other resource data) for the content server. |
| tables | Holds IdocScript resource table definitions. |
| templates | Holds templates for all content server pages (except reports). |

The *IdcHomeDir*/resources/admin directory contains the following resource types:

| Element | Description |
|---|---|
| idoc | Holds IdocScript dynamichtml definitions. |
| tables | Holds IdocScript resource table definitions. |
| templates | Holds templates for all content server pages (except reports). |

### 2.1.6 weblayout Directory

The *DomainHome*/ucm/*short-product-id*/weblayout directory contains the files that are available to the Web server for display on the various pages of the content server 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 the content server. They can be pieces of HTML code, dynamic page elements, queries that gather data from the database, services that perform content server actions, or special code to conditionally format information.

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

Resources fall into  distinct categories, although the first four 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 content server Web page. | *IdcHomeDir*/resources/core/idoc /std_page.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 the content server. | *IdcHomeDir*/resources/core/datas toredesign/columnIndexdList.hda |

| Resource Type | Description | Example of Standard Resource |
|---|---|---|
| Static Table (HTML format) | Provides static (seldom changed) content in table format to the 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 the content server. | *IdcHomeDir*/resources/core/tables/std_services.htm |
| Template | Defines templates, which contain the code that the content server uses to assemble a particular Web page. | *IdcHomeDir*/resources/core/templates/checkin_new.htm |
| Environment | Defines configuration settings for the content server. | *IntradocDir*/config/config.cfg |

## 2.3 Content Server Behavior

This section describes how the content server behaves in the following situations:

- "Startup Behavior" on page 2-6
- "Resource Caching" on page 2-8
- "Content Server Requests" on page 2-9
- "Page Assembly" on page 2-11
- "Database Interaction" on page 2-11
- "Resolving Localized Strings" on page 2-12

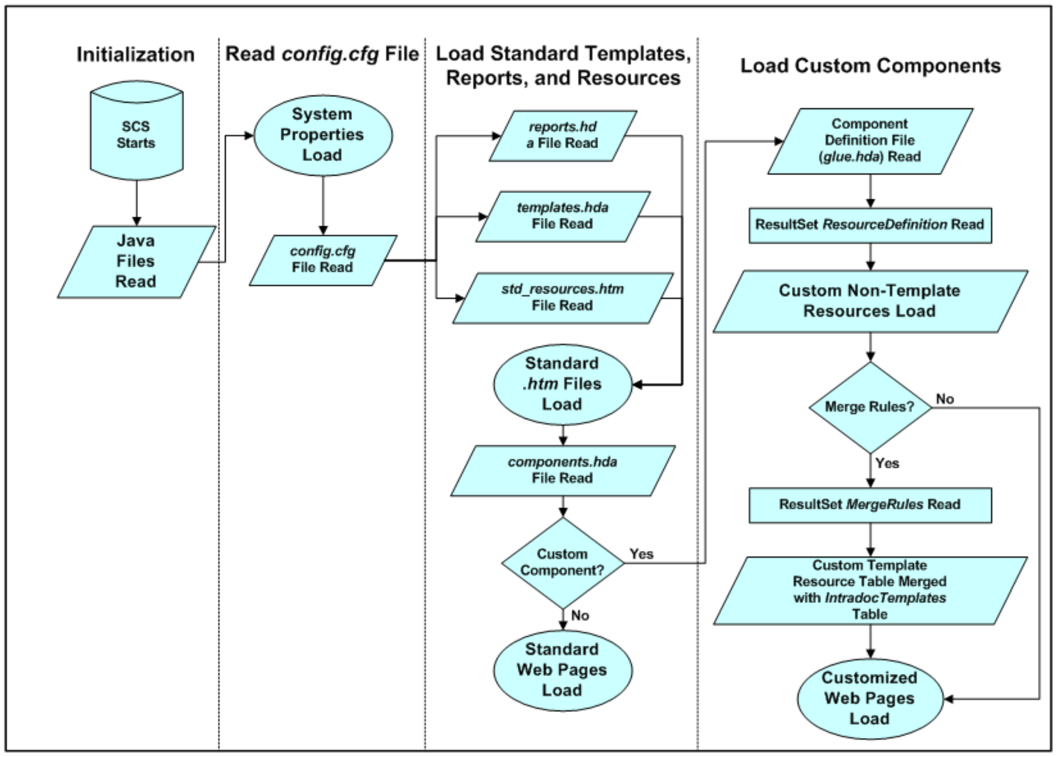
### 2.3.1 Startup Behavior

The following steps occur during 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).

These steps are discussed in more detail on the following pages.

***Figure 2–1 Content server startup behavior***


Initialization
SCS Starts
Java Files Read
Read config.cfg File
System Properties Load
config.cfg File Read
Load Standard Templates, Reports, and Resources
reports.hd a File Read
templates.hda File Read
std_resources.htm File Read
Standard .htm Files Load
components.hda File Read
Custom Component?
Yes
No
Standard Web Pages Load
Load Custom Components
Component Definition File (glue.hda) Read
ResultSet ResourceDefinition Read
Custom Non-Template Resources Load
Merge Rules?
No
Yes
ResultSet MergeRules Read
Custom Template Resource Table Merged with IntradocTemplates Table
Customized Web Pages Load

1. **Internal Initialization Occurs:** When the content server initializes internally, the Java class files from the 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, the 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 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 UNIX).
   - By editing the configuration files directly.
   - By using a custom component.

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

3. **Standard Resources, Templates, and Reports Load**: For the Content Server to function properly, many standard resources, templates, and reports must be loaded. After the configuration settings have been loaded, the Content Server reads the entries in the *IdcHomeDir*/resources/core/templates/templates.hda file and the *IdcHomeDir*/resources/core/reports/reports.hda file.

4. **Custom Components Load**: The content server loads custom components in the order specified in the *IntradocDir*/custom/components.hda file. As each custom component is loaded, any resources defined for that component override any resources with the same name. For example, if two components are enabled that both use a resource called *my_resource.htm*, the resource definition for the component that is loaded last takes precedence.

#### 2.3.1.1 Effects of Configuration Loading

It is important to understand the effect of the load order of the different configuration files.

The *IntradocDir*/config/config.cfg file is loaded first, and the *IntradocDir*/data/components/*component_name*/config.cfg file is loaded last.

Therefore, if a variable is set in more than file, the last variable loaded takes precedence. 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 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 don't include the one to change, make the change directly in one of the configuration files.

### 2.3.2 Resource Caching

- When the 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 the content server before the changes go into effect. This is because such changes could cause the content server to malfunction if they were implemented immediately. You do not need to restart the content server when changing strings,

however, you must republish the ww_strings.js files by selecting **Publish dynamic files** from the Admin Actions page. For more information see Chapter 3, "Working with Components".

### 2.3.3 Content Server Requests

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

- Retrieves pages—See "Page Retrieval" on page 2-10.
- Runs a content server service—See "Content Server Services" on page 2-10.
- Runs a search engine service—See "Search Services" on page 2-10.

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

When a content server Web page is requested, all of the necessary information can be sent to the content server through the URL. The following is a typical content server URL; in this case, it is the URL for the Home page:

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

- `http://cs.company.com` and `instancename` is the Web address of the content server instance.
- `IdcService=GET_DOC_PAGE` tells the content server to execute the GET_DOC_ PAGE service.
- `Action=GetTemplatePage` tells the content server to return the results using a specified template page.
- `Page=HOME_PAGE` tells the content server which template page to use.
- The question mark (**?**) indicates the end of the Web server path and the beginning of content server instructions.
- Ampersands (**&**) are used as separators between 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. For example, the following variables can be used for customization:
  - &StdPageWidth=1000
  - &dDocAuthor:isHidden
  - &dDocType=HRForm

Necessary information can also be sent to the content server through form fields on the page. The following is a typical Content Server form:

```
<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 the Content Server, one of the following page types is returned:

- **static page:** The content of a static Web page is pre-formatted, and does not change from one request to the next. In some 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 Content Server services and templates to determine the content and formatting. For example, each user's portal design page is generated using a content server service called GET_PORTAL_PAGE and a template called PNE_PORTAL_DESIGN_PAGE.

#### 2.3.3.2 Content Server Services

When a Web browser requests a dynamic page from the Content Server, the browser is actually placing a request for a 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 a Content Server function, and sends the specific request to the Content Server.
3. When the 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 *admin* permission.
   - 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 Content Server service to the originating Web browser client.

#### 2.3.3.3 Search Services

A search request is a special kind of Content Server service. When the Content Server receives a search request, it sends the request on to the search engine using a search engine API. This allows different search engines to be used with the 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 a Content Server function, and sends the specific request to the Content Server.
3. The Content Server passes the request to the search engine.
4. The search engine returns the search results to the Content Server.
5. Based on the user login and security permissions, the 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

The 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 /templates or /reports directory.
- The template file references *resources*, which are located in .htm and .idoc files in 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 Content Server resources are cached in memory at startup, the Content Server has a definition for the standard pieces that appear on the page. The 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 the Content Server. When one of these pages is requested by a Web browser, the Content Server recognizes the file extension as a dynamic server page, which allows special processing to occur. 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 the Content Server.

### 2.3.5 Database Interaction

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

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

See *"Modifying System Functionality"* on page 5-1 for information about using the upper_clmns_map.htm file.

### 2.3.6 Resolving Localized Strings

Localized strings are the means by which the user interface and error messages are presented in the language specified by the user's locale. The 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 Components

This chapter describes the structure of components and how to work with components, which are programs used to modify Content Server functionality. It covers these topics:

## 3.1 Components Overview

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

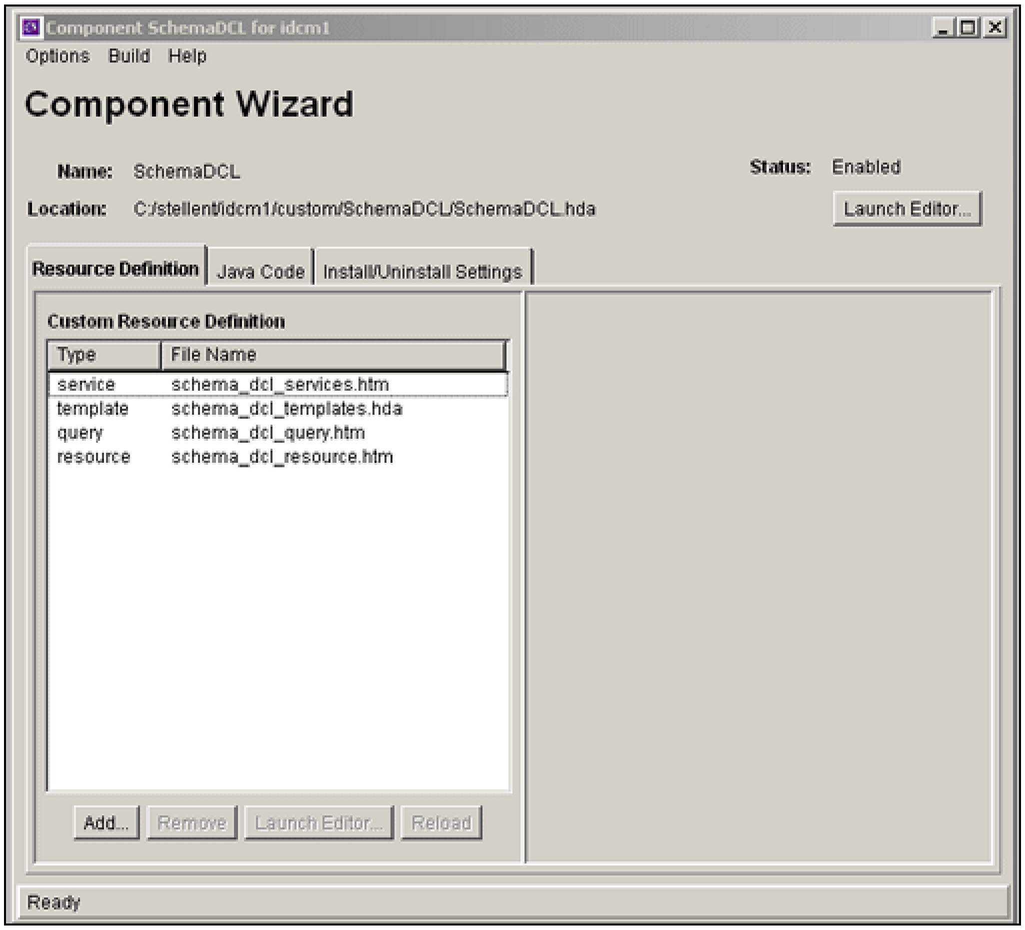
This section provides an overview of component management and the files and directory structure associated with components. It covers these topics:

### 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 Content Server instances.

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

*Figure 3–1 Component Wizard interface*



Component Wizard

**Name:** SchemaDCL

**Status:** Enabled

**Location:** C:/stellent/idcm1/custom/SchemaDCL/SchemaDCL.hda

**Custom Resource Definition**

| Type | File Name |
|---|---|
| service | schema_dcl_services.htm |
| template | schema_dcl_templates.hda |
| query | schema_dcl_query.htm |
| resource | schema_dcl_resource.htm |

Ready

**To access the Component Wizard**

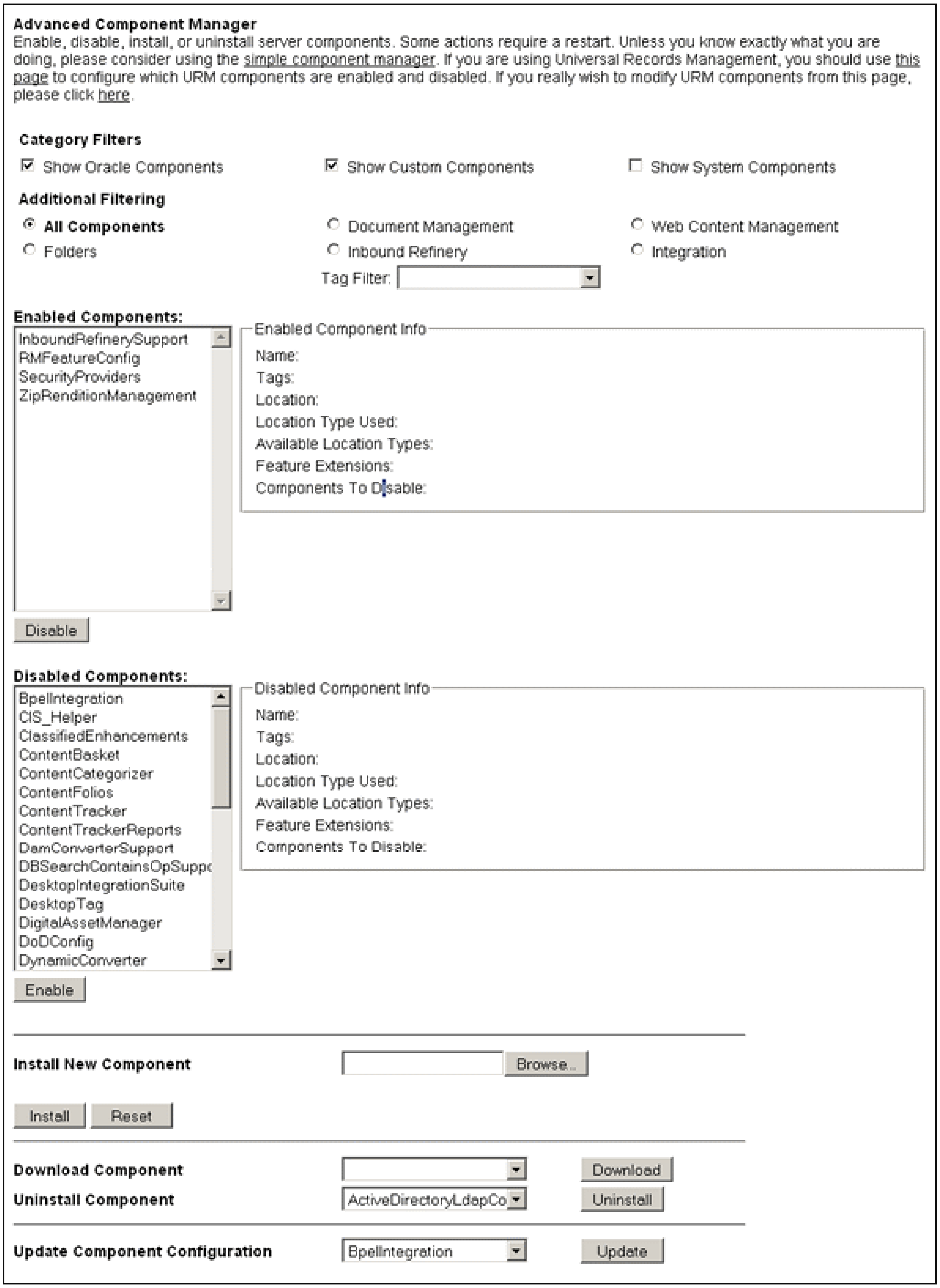
- (Windows) From the **Start** menu click the instance name, then **Utilities**, then **Component Wizard**. The Component Wizard main page is displayed.
- (UNIX) Run ComponentWizard, stored in *DomainHome*/ucm/*short-product-id*/bin. The Component Wizard main page is displayed.

## 3.1.2 Advanced Component Manager

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

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

***Figure 3–2 Advanced Component Manager page***



**Advanced Component Manager**
Enable, disable, install, or uninstall server components. Some actions require a restart. Unless you know exactly what you are doing, please consider using the simple component manager. If you are using Universal Records Management, you should use this page to configure which URM components are enabled and disabled. If you really wish to modify URM components from this page, please click here.

**Category Filters**

[✓] Show Oracle Components  [✓] Show Custom Components  [ ] Show System Components

**Additional Filtering**

[● All Components]  ○ Document Management  ○ Web Content Management
○ Folders  ○ Inbound Refinery  ○ Integration

Tag Filter:

**Enabled Components:**

InboundRefinerySupport
RMFeatureConfig
SecurityProviders
ZipRenditionManagement

Enabled Component Info
Name:
Tags:
Location:
Location Type Used:
Available Location Types:
Feature Extensions:
Components To Disable:

Disable

**Disabled Components:**

BpelIntegration
CIS_Helper
ClassifiedEnhancements
ContentBasket
ContentCategorizer
ContentFolios
ContentTracker
ContentTrackerReports
DamConverterSupport
DBSearchContainsOpSuppo
DesktopIntegrationSuite
DesktopTag
DigitalAssetManager
DoDConfig
DynamicConverter

Disabled Component Info
Name:
Tags:
Location:
Location Type Used:
Available Location Types:
Feature Extensions:
Components To Disable:

Enable

**Install New Component**  Browse...

Install  Reset

**Download Component**  Download
**Uninstall Component**  ActiveDirectoryLdapCo  Uninstall

**Update Component Configuration**  BpelIntegration  Update

To use the Advanced Component Manager, click **Administration** on the portal navigation tray for your Content Server, then click **Admin Server**. On the Admin Server page, the **Component Manager** page is displayed. In the first paragraph of the Component Manager page, which displays lists of enabled and disabled server components, click **advanced component manager**. You can select individual components in the tables to view details about each component and select categories of components to view. You can enable and disable components on this page, plus install and uninstall custom components.

### 3.1.3 ComponentTool

ComponentTool is a command-line utility for installing, enabling, and disabling components in 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 the Content Server what components are enabled and where to find the definition file for each component.
- The component definition (or *glue*) file, which tells the Content Server where to find the resources for the custom component.
- Different custom resource files, which define your customization to standard Content Server resources.
- Template files, which define custom template pages.
- Other files which contain customization to Content Server graphics, Java code, help files, and so forth.

These files are all discussed in more detail in "About Directories and Files" on page 3-5.

Any type of file can be included in a component, but the following file formats are used most often:

- HDA
- HTM
- CFG
- Java CLASS

If you build or unpackage components in the Component Wizard, or upload and download components in the Component Manager, you work with the following files:

- A compressed zip file used to deploy a component on other Content Servers.
- A manifest.hda file that tells the 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:

- **Manual editing:** In the *IntradocDir*/data/components/ directory, open the idc*short-product-id*_components.hda file in a text editor and add or delete the two-line entry for the component. Alternately, you can change only the `Enabled` or `Disabled` setting for the component and save the file.
- **ComponentTool**: Run the *DomainHome*/ucm/*short-product-id*/bin/ComponentTool to enable or disable a component. For example, `ComponentTool -enable` *`component_name`*
- **Component Wizard:** Select **Enable** or **Disable** from the **Options** menu. For details, see the *Oracle Fusion Middleware System Administrator's Guide for Content Server*.
- **Component Manager:** Select the check box next to a component name to enable a server component specified on the Component Manager screen. Clear the check box next to a component name to disable a server component on the Component Manager screen. For details, see the *Oracle Fusion Middleware System Administrator's Guide for Content Server*.
- **Advanced Component Manager:** Select a component name and click **Disable** to disable the component on the Advanced Component Manager screen. Select a component name and click **Enable** to enable the component on the Advanced Component Manager screen.

## 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 topics:

- "HDA Files" on page 3-5
- "Custom Resource Files" on page 3-9
- "Data Binder" on page 3-10
- "Manifest File" on page 3-11
- "Other Files" on page 3-12
- "Typical Directory Structure" on page 3-12

### 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 the 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 the 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 version="" jcharset=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 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 the 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.

The Properties Section and ResultSet Section are the two section types that are relevant to component development. These 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. ResultSets often represent 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 type is rarely used. The recommended practice is to define global environment variables in a configuration file in an Environment resource.

  The following is an example of a Properties section in an HDA file.

  ```
  @Properties LocalData
  PageLastChanged=952094472723
  LocationInfo=Directory,Public,
  IsJava=1
  refreshSubMonikers=
  PageUrl=/intradoc/groups/public/pages/index.htm
  LastChanged=-1
  TemplatePage=DIRECTORY_PAGE
  IdcService=PAGE_HANDLER
  LinkSelectedIndex=0
  PageName=index
  HeaderText=This is a sample page.  The Page Name must remain index.  The Page
  Properties for this index page should be customized.
  ```

```
PageFunction=SavePage
dSecurityGroup=Public
restrictByGroup=1
PageType=Directory
PageTitle=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 results 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 |

The Content Server uses some predefined ResultSets, so the following names should not be used for custom component tables:

| ResultSet Name | Location | Purpose |
|---|---|---|
| Components | *IntradocDir*/data/components/idc*custon-name_*components.hda | Defines the name and location of any custom components you have created. You must specify the custom name. |
| IntradocReports | *IdcHomeDir*/resources/core/reports/reports.hda | Specifies the default report templates for the Content Server. |
| IntradocTemplates | *IdcHomeDir*/resources/core/templates/templates.hda | Specifies all of the default templates for the 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 the Content Server. |

#### 3.2.1.2 The idc_components.hda File

The idc_components.hda file is a text file that tells the 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 11gR1, 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 idcshort-product-name_components.hda file but does find the legacy files, then it will migrate to the new format.

The following is an example of an idccs_components.hda file, listing 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 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 |

These files are all discussed in detail in "Resources Detail" on page 3-20.

In addition, a template.htm page is used by the Content Server to assemble web pages. See "Templates" on page 3-35 for details about the template.hdm file.

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

The Content Server caches data (such as variable values and lookup keys) internally in a DataBinder. All data in the DataBinder 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 DataBinder is evaluated in the following default order:

1. LocalData
2. Active or Non-active ResultsSets
3. Environment

This precedence can be changed using Idoc Script functions. See the *Oracle Fusion Middleware Idoc Script Reference Guide* for details.

### 3.2.3.1 LocalData

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

The LocalData information is maintained only during the lifetime of the Content Server request and response. Unlike information about the server environment, which rarely changes, the LocalData information for each request is dynamic.

From the point of view of an HTTP request, the initial LocalData is collected from the REQUEST_METHOD, CONTENT_LENGTH, and QUERY_STRING HTTP environment variables. As the service request is processed, the LocalData values can be added and changed.

### 3.2.3.2 Active or Non-active ResultsSets

Each `@ResultSet` section of an HDA file maps to a named result in the DataBinder. A ResultSet becomes active when the ResultSet is looped on during page assembly. The 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 non-active ResultSet values 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 the Content Server. This file tells the 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 | Java Server Pages | *DomainHome*/ucm/*short-product-id*/weblayout/jsp |

- 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 the Content Server which resource files are included in the component.
  - For other entry types, the location can be a path (to specify all files in a particular subdirectory) or a path and 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 only be installed on Content Servers with the same installation directory path.

The following is an example of a manifest.hda file.

```
@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 the Content Server uses for functionality or to generate 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 the Content Server. For example, the following types of files are often referenced in custom resources:

- **Graphics:** Replace the icons, backgrounds, and logos that constitute the standard Content Server interface.
- **Help:** With the assistance of Consulting Services, help files can be customized for your content management system.
- **Classes:** Java code can change or extend the functionality of the Content Server. Java class files must be packaged into directories before placing them in the *DomainHome*/ucm/*short-product-id*/classes/ directory.

#### 3.2.5.2 Component Zip File

A component zip file contains all files that define a Content Server component. It can be unpackaged to deploy the component on other Content Servers.

#### 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 component directory within the data/components directory. These files hold the preference data as follows:

- config.cfg: Contains the parameters that can be reconfigured after installation.
- install.cfg: 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, /Resources). The *component_name*.hda file (the definition file) is stored in the /*component_name* directory.

## 3.3 Development Recommendations

This section provides some guidelines to assist you in developing custom components. It covers these topics:

- "Creating a Component" on page 3-13
- "Working with Component Files" on page 3-13
- "Using a Development Instance" on page 3-14
- "Component File Organization" on page 3-14
- "Naming Conventions" on page 3-15

See the *Oracle Fusion Middleware System Administrator's Guide for Content Server* or online help for detailed instructions on creating or modifying components.

### 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 the Content Server to apply the component.
4. Create resources and other files to define your customization. A good approach is to copy, rename, and modify standard Content Server files to create your custom resource files.
5. Test and revise your customization as necessary. You may need to restart the Content Server to apply your changes.
6. If you want to package the component for later use or for deployment on other Content Servers, build the component and create a component zip file.

### 3.3.2 Working with Component Files

There are two ways to work with component files:

- **Component Wizard:** The Component Wizard is a Content Server utility that helps you create and edit component files. You can also use the Component Wizard to package, unpackage, enable, and disable components. For more information on using this utility, see the *Oracle Fusion Middleware System Administrator's Guide for 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 instructions on using the Component Wizard to create components, see the *Oracle Fusion Middleware System Administrator's Guide for Content Server*.

### 3.3.3 Using a Development Instance

Whenever you are customizing the 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 the Content Server after enabling or disabling a component.

If you are having problems with your Content Server after you have installed a custom component, disable the component and restart the Content Server. If this fixes the problem, you probably need to troubleshoot your component. If the problem is not fixed, you may need to remove the component completely using the Component Wizard to see if there is a problem with the component or with the Content Server.

### 3.3.4 Component File Organization

To keep your custom components organized, follow these file structure guidelines. See "Typical Directory Structure" on page 3-12 for more information.

**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- 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.
- The 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 the Content Server, but all component files must be accessible to the Content Server.
- Images and other objects that are referenced by Content Server Web pages must reside somewhere in the *DomainHome*/ucm/*short-product-id*/weblayout/ directory (so 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 the 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 in the Content Server, it overrides any resources with the same file names, so you should use duplicate file names only if you want certain components to take precedence.
- If you are copying a standard Content Server file, a common practice is to place the prefix `custom_` in front of the original file name. This ensures that you do not overwrite any default templates, and your customization is easy to identify.
- HTM file types should have a `.htm` extension, and HDA file types should have a `.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`.
- The Content Server is case sensitive even if your file system is not. For example, when a file is named `My_Template`, the 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 the 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. See "Resolving Localized Strings" on page 2-12.

## 3.4 Component File Detail

This section discusses the HDA file type and the component definition (glue) file in more detail. The following topics are discussed:

- "The idc_components.hda File" on page 3-15
- "Component Definition (Glue) File" on page 3-16

The information in this chapter 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 the Content Server which components are enabled and where to find the component definition (glue) file for each component. With 11*g* Release 1 (11.1.1) this file has three forms, one for each of the UCM products: idccs_components.hda (for Content Server), idcibr_components.hda (for Inbound Refinery), and idcurm_components.hda (for Universal Records Management). The file is always stored in the *IntradocDir*/data/components/ directory.

The 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. See "Enabling and Disabling Components" on page 3-4 for more information.

In the following example of an idccs_components.hda file, two components called "My Component" 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
```

#### 3.4.1.1 Components ResultSet

The order that components are listed in the `Components` ResultSet determines the order that components are loaded when you start the Content Server. If a component listed later in the ResultSet has a resource with the same name as an earlier component, the resource in the later component takes precedence.

A `Components` ResultSet has two columns:

- The `name` column provides a descriptive name for each component, which is used in the Component Wizard, Component Manager, and Content Server error messages.
- The `location` column defines the location of the definition file for each component. The location can be an absolute path or can be a path relative to the Content Server install 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 includes a ResourceDefinition ResultSet, and may contain a MergeRules ResultSet, a Filters ResultSet or a ClassAliases ResultSet or both.

The following example shows a typical component definition file.

```
@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, classpath, library path, features, and load order of custom resources. 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
```

**3.4.2.1.1 ResourceDefinition ResultSetColumns** 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.

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
```

**3.4.2.2.1 MergeRules 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 must have the same number of columns and the same column names as the toTable.
- The toTable column specifies the name of the existing table into which the new data is merged. Usually, the toTable value is one of the standard Content Server tables, such as IntradocTemplates or QueryTable.
- The column column is the name of the table column that the Content Server uses to compare and update data.
  - The Content Server compares the values of the specified column in the fromTable and toTable. For each fromTable value that is identical to a value currently in the toTable, the row in the toTable is replaced by the row in the fromTable. For each fromTable value that is not identical to a

value currently in the `toTable`, a new row is added to the `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 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 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 the Content Server. Resources can be snippets of HTML code, dynamic page elements, queries that gather data from the database, services that perform Content Server actions, or special code to conditionally format information.

The custom resource files for a component are typically located in the *DomainHome*/ucm/*short-product-id*/custom/*component_name*/ directory. If your component has more than a few resources, it is easier to maintain the files if you place them in sub-directories (such as /resources or /templates) within the component directory.

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

- **Manual editing:** Open the resource file in a text editor and edit the code manually. *This is not recommended.*

- **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.

See the *Oracle Fusion Middleware System Administrator's Guide for Content Server* or online help for more information.

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

The following sections discuss these resource categories:

- "HTML Include" on page 3-21
- "String" on page 3-22
- "Dynamic Tables" on page 3-25
- "Static Tables" on page 3-26
- "Query" on page 3-26
- "Service" on page 3-28
- "Templates" on page 3-35
- "Environment" on page 3-41

## 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 the 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 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. You can also specify multiple `super` tags to call an include that was loaded earlier than the last version.

---

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

---

***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 resource, select the Includes tab in the right pane.
4. Modify the includes in the Custom HTML Includes list.
   - To edit an existing include, select the include and click **Edit**. Modify the code and click **OK**.
   - To add an include to the resource file, click **Add**.
   - To remove an include, select the include and click **Delete**. Click **Yes** to confirm.

### 3.5.2 String

A `string` resource defines locale-sensitive text strings that are used in error messages and on Content Server Web pages and applets. Strings are resolved by the 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:

```
<$lc("wwStringID")$>
```

**Note:** On Content Server Web pages, you should use only the strings in the ww_strings.htm file.

Standard English strings are defined in the *IdcHomeDir*/resources/core/lang/ directory. Strings for other supported languages are provided by the Localization component.

HTML includes, strings, and static tables can be present in the same HTM file. A `string` resource does not require merge rules.

You must use HTML escape encoding to include the following special characters in a `string` value:

| **Escape Sequence** | **Character** |
|---|---|
| &at; | @ |
| \&lf; | line feed (ASCII 10) |
| \&cr; | carriage return (ASCII 13) |
| \&tab; | tab (ASCII 9) |
| \&eatws; | Eats white space until the next non-white space character. |
| \&lt; | < (less than) |
| \&gt; | > (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 are all single-byte or all multibyte. 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 Content Server reads them correctly.

#### 3.5.2.1 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 that contains the locale information. The following table describes the syntax for parameterized strings:

| **Syntax** | **Meaning** | **Examples** |
|---|---|---|
| {{} | Opening curly brace. (Note that only the opening curly brace must be expressed as a literal.) | {{}Text in braces} |
| {n} | Substitute the *n*th argument. | Content ID {1} not found |
| {ni} | Substitute the *n*th argument, formatted as an integer. | dID {1i} does not exist |
| {nx} | Substitute the *n*th argument, formatted as an integer in hexadecimal. | |
| {nd} | Substitute the *n*th argument, formatted as a date. | The release date is {1d} |
| {nD} | Substitute the *n*th argument, formatted as a date. The argument should be ODBC-formatted. | The release date is {1D} |
| {nt} | Substitute the *n*th argument, formatted as a date and time. | The release date is {1t} |
| {ne} | Substitute the *n*th argument, formatted as elapsed time. | |
| {nT} | Substitute the *n*th argument, formatted as a date and time. The argument should be ODBC-formatted. | The release date is {1T} |
| {nfm} | Substitute the *n*th argument, formatted as a float with m decimal places. | The distance is {1f3} miles. |
| {nk} | Substitute a localized string using the *n*th argument as the string ID. | Unable to find {1k} revision of {2} |
| {nm} | Localize the *n*th argument as if it were a string-stack message. (For example, the argument could include concatenated text strings and localized string IDs.) | Indexing internal error: {1m} |
| {nl} | Substitute the *n*th argument as a list. The argument must be a list with commas (,) and carets (^) as the separators. | Add-ons: {1l} |
| {nK} | Takes a list of localization key names, separated by commas, and localizes each key into a list. | Unsupported byte feature(s): {1K} |
| {nM} | Takes a list of message strings and localizes each message into a list. | {1q} component, version {2q}, provides older versions of features than are currently enabled. {3M} |
| {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 |

| Syntax | Meaning | Examples |
|---|---|---|
| {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*.<br>▪ If the value of the *n*th argument is 1, substitute *text2*.<br>The (n?) function can be extended with as many substitution variables as required. The last variable in the list always corresponds to a value of 1. | There {1?are:is} currently {1} active search{1?es}. |
| {n?text1:text2:text3} | ▪ If the value of the *n*th argument is not 1 or 2, substitute *text1*.<br>▪ If the value of the *n*th argument is 2, substitute *text2*.<br>▪ If the value of the *n*th argument is 1, substitute *text3*.<br>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. |

#### 3.5.2.2 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 resource, select the Strings tab in the right pane.
4. Modify the strings in the Custom Strings list.
   - To edit an existing string, select the string and click **Edit**. Modify the string text and click **OK**.
   - To add a string to the resource file, click **Add**.
   - To remove a string, select the string and click **Delete**. Click **Yes** to confirm.

### 3.5.3 Dynamic Tables

Dynamic table resources are defined in the HDA file format. See "Elements in HDA Files" on page 3-6 for more information and an example of an HDA ResultSet table. 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.

#### 3.5.3.1 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 in the right pane of the Resource Definition tab.

### 3.5.4 Static Tables

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

#### 3.5.4.1 Editing a Static Table Resource

Use the following procedure to edit an existing static 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 in the Resource Tables list.

### 3.5.5 Query

A query resource defines SQL queries, which are used to manage information in the Content Server database. Queries are used with service scripts to perform tasks such as adding to, deleting, and retrieving data from the database.

The standard Content Server queries are defined in the *QueryTable* table in the *IdcHomeDir*/resources/core/tables/query.htm file. You 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 |

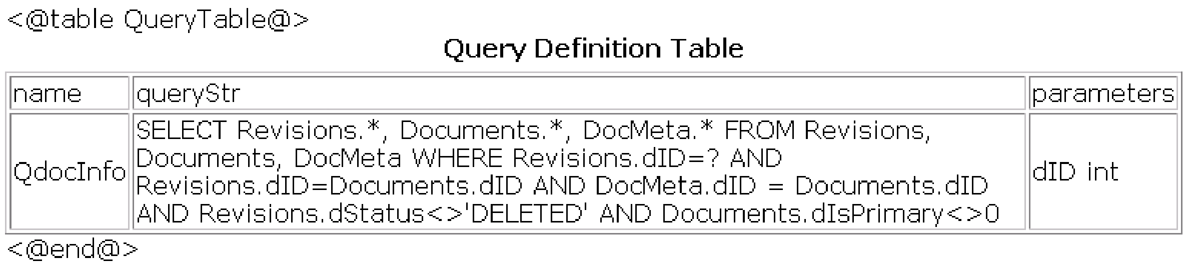
| First Character | Query Type |
|---|---|
| U | Update |

- 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.

The following example shows the standard *QdocInfo* query as defined in the <*instance_dir*>/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 for the primary revision from the *Revisions*, *Documents*, and *DocMeta* database tables that matches the dID, if the revision does not have "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>
```

#### 3.5.5.1 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 and click **Edit**. Modify the query expression or parameters or both, and click **OK**.
   - To remove a query, select the query and click **Delete**. Click **Yes** to confirm.

### 3.5.6 Service

A service resource defines a function or procedure that is performed by the Content Server. A service call can be performed from either the client or server side, so services can be performed on behalf of the Web browser client or within the system itself. For example:

- **Client-side request:** When you click a "Search" link on a Content Server Web page, the standard search page is delivered to your Web browser by the GET_DOC_PAGE service using the following URL segment:

  ```
  IdcService=GET_DOC_PAGE&Action=GetTemplatePage&Page=STANDARD_QUERY_PAGE
  ```

- **Server-side request:** You can use the START_SEARCH_INDEX service to update or rebuild the search index automatically in a background thread.

Services are the only way a client can communicate with the server or access the database. Any program or HTML page can use services to request information from the Content Server or perform a specified function.

**Important:** This section provides an overview of custom service resources. See the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management* for comprehensive information on Content Server services.

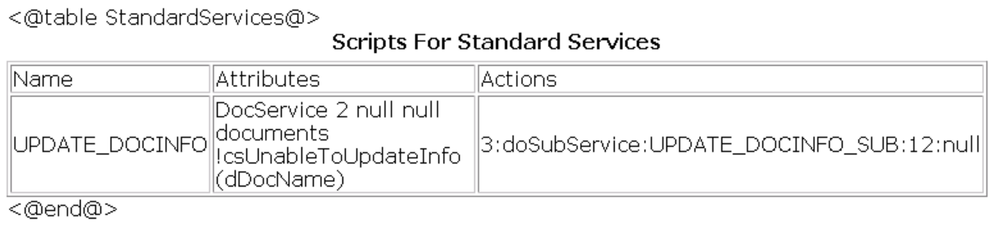
The standard Content Server services are defined in the *StandardServices* table in the *IdcHomeDir*/resources/core/tables/std_services.htm file. You can also find special-purpose services in the *workflow.htm* file in the *IdcHomeDir*/resources/core/tables/ directory.

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:<br>READ_PRIVILEGE = 1<br>WRITE_PRIVILEGE = 2<br>DELETE_PRIVILEGE = 4<br>ADMIN_PRIVILEGE = 8<br>GLOBAL_PRIVILEGE = 16<br>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 (see *Resolving Localized Strings* in *Customizing* Content Server for more information). | `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. Actions 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:

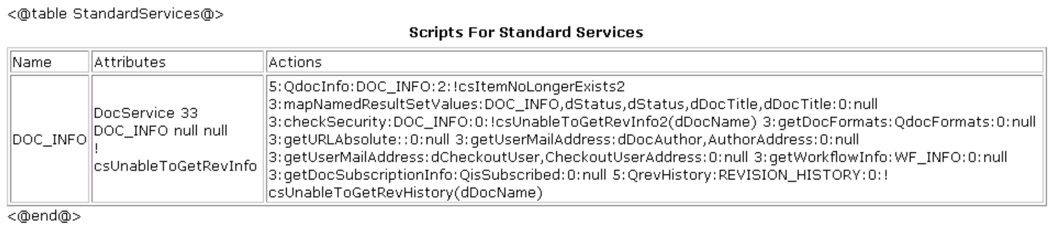
  ```
  type:name:parameters:control mask:error message
  ```

| Section | Description | Example (first action from the DELETE_DOC service) |
|---|---|---|
| Type | Defines the type of action:<br>QUERY_TYPE = 1<br>EXECUTE_TYPE = 2<br>CODE_TYPE = 3<br>OPTION_TYPE = 4<br>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:<br>No control mask = 0<br>CONTROL_IGNORE_ERROR = 1<br>CONTROL_MUST_EXIST = 2<br>CONTROL_BEGIN_TRAN = 4<br>CONTROL_COMMIT_TRAN = 8<br>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 (see *Resolving Localized Strings* in *Customizing Content Server* for more information). | `5:QdocInfo:DOC_INFO:6:!csUnableToDeleteItem(dDocName)!csRevisionNoLongerExist` |

#### 3.5.6.1 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 <*instance_dir*>/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<br>DOC_INFO null null<br>!<br>csUnableToGetRevInfo | 5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2<br>3:mapNamedResultSetValues:DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null<br>3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName) 3:getDocFormats:QdocFormats:0:null<br>3:getURLAbsolute::0:null 3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null<br>3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null 3:getWorkflowInfo:WF_INFO:0:null<br>3:getDocSubscriptionInfo:QisSubscribed:0:null 5:QrevHistory:REVISION_HISTORY:0:!<br>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>
```

#### 3.5.6.1.1 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.<br><br>1 = The user requesting the service must have Read privilege on the content item. |

| Attribute | Value | Description |
|---|---|---|
| Template page | DOC_INFO | This service uses the DOC_INFO template (*doc_ info.htm* file). The results from the actions are merged with this template and presented to the user. |
| Service type | null | This service is not a subservice. |
| Subjects notified | null | No subjects are affected by this service. |
| Error Message | !csUnableToGetRevInfo | If this service fails on an English Content Server system, it returns the error message string: *Unable to retrieve information about the revision* |

**3.5.6.1.2 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 the query must return a record, or the action fails. |
| !csItemNoLongerExists2 | If this action fails on an English Content Server system, it returns the 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. |

| Action Definition | Description |
|---|---|
| checkSecurity | This action retrieves the data assigned to the DOC_INFO parameter and evaluates the assigned security level to verify that the user is authorized to perform this action. |
| DOC_INFO | Parameter that contains the security information to be evaluated by the *checkSecurity* action. |
| 0 | No control mask is specified. |
| !csUnableToGetRevInfo2(dDocName) | If this action fails on an English Content Server system, it returns the 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 Content Server system, it returns the error message string: *Unable to retrieve revision history for ''{dDocName}.''* |

#### 3.5.6.2 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 and click **Edit**. Modify the service attributes or actions or both, and click **OK**.
   - To remove a service, select the service and click **Delete**. Click **Yes** to confirm.

### 3.5.7 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 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 the 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. See "Elements in HDA Files" on page 3-6 for more information and an example of an HDA ResultSet table. The standard templates are defined in the *IdcHomeDir*/resources/core/templates/templates.hda file.

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 is an example of 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 Content Server Web pages except search results pages. This table consists of five columns:
  - The `name` column defines the name for each template page. This name is how the template is referenced in the Content Server CGI URLs and in code.
  - The `class` column defines the general category of the template. The most common *class* type is *Document*.
  - The `formtype` column defines the specific type of functionality the page is intended to achieve. The `formtype` is typically the same as the name of the form, except in lowercase characters.
  - The *filename* column defines the path and file name of the template file. The location can be an absolute path or can be relative to the template resource file when the template page is in the same directory as the template resource file.
  - The *description* column defines a description of the template.
- The Verify Template. The Content Server no longer uses the *VerityTemplates* ResultSet table. However, 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. `SearchResultTemplates` define how query results are displayed on the search results pages in the Library. Query result pages are a special type of search results page. This table consists of six columns:
  - The `name` column defines the name for each template page. This name is how the template is referenced in the Content Server CGI URLs, in code, and in the Web Layout Editor utility.

    **Note:** The `StandardResults` template (search_results.htm file) is typically used as the global template for standard search results pages and the query results pages in the Library. You can create a new template or change the "flexdata" of the `StandardResults` template through the Web Layout Editor, but these changes are saved in a separate file (*IntradocDir*/data/results/custom_results.hda) rather than in the `SearchResultTemplates` table in the templates.hda file.

  - The `formtype` column defines the specific type of functionality the page is intended to achieve. ResultsPage is the only form type currently supported for search results pages.

- The `filename` column defines the path and file name of the template file. The location can be an absolute path or can be relative to the template resource file when the template page is in the same directory as the template resource file.
- The `outfilename` column is for future use; the value is always null.
- The `flexdata` column defines the metadata to be displayed for each row on the search results page. The format of text in the `flexdata` column is:

  ```
  Text1 "text 1 contents"%<Tab>Text2 "text 2 contents"%
  ```

  where *Text1* contents appear on the first line, and *Text2* contents appear on the second line in each search results row. `<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
```

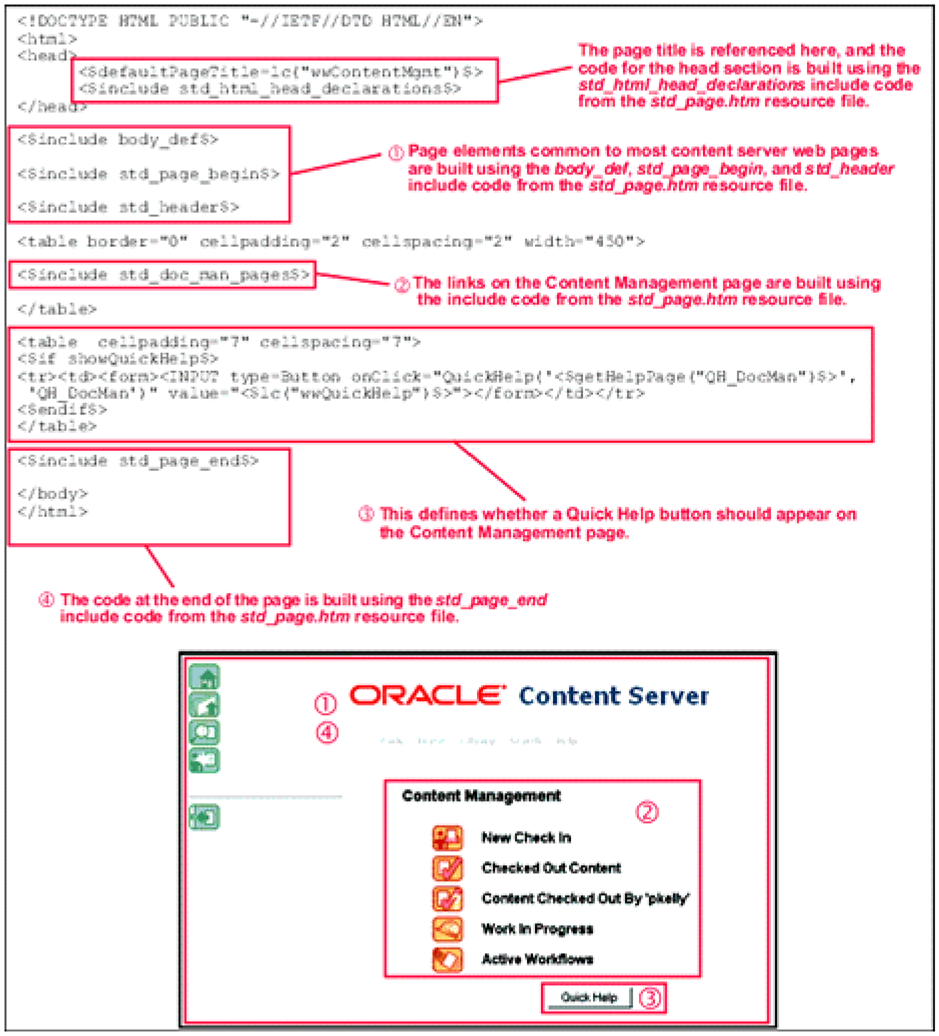
#### 3.5.7.1 Template and Report Pages

Template pages and report pages are also called "presentation" pages, because the 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.

**3.5.7.1.1 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*



```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
    <$defaultPageTitle=lc("wwContentMgmt")$>
    <$include std_html_head_declarations$>
</head>
<$include body_def$>

<$include std_page_begin$>

<$include std_header$>

<table border="0" cellpadding="2" cellspacing="2" width="450">

<$include std_doc_man_pages$>

</table>

<table  cellpadding="7" cellspacing="7">
<$if showQuickHelp$>
<tr><td><form><INPUT type=Button onClick="QuickHelp('<$getHelpPage("QH_DocMan")$>',
 'QH_DocMan')" value="<$lc("wwQuickHelp")$>"></form></td></tr>
<$endif$>
</table>
<$include std_page_end$>

</body>
</html>
```

The page title is referenced here, and the code for the head section is built using the *std_html_head_declarations* include code from the *std_page.htm* resource file.

① Page elements common to most content server web pages are built using the *body_def*, *std_page_begin*, and *std_header* include code from the *std_page.htm* resource file.

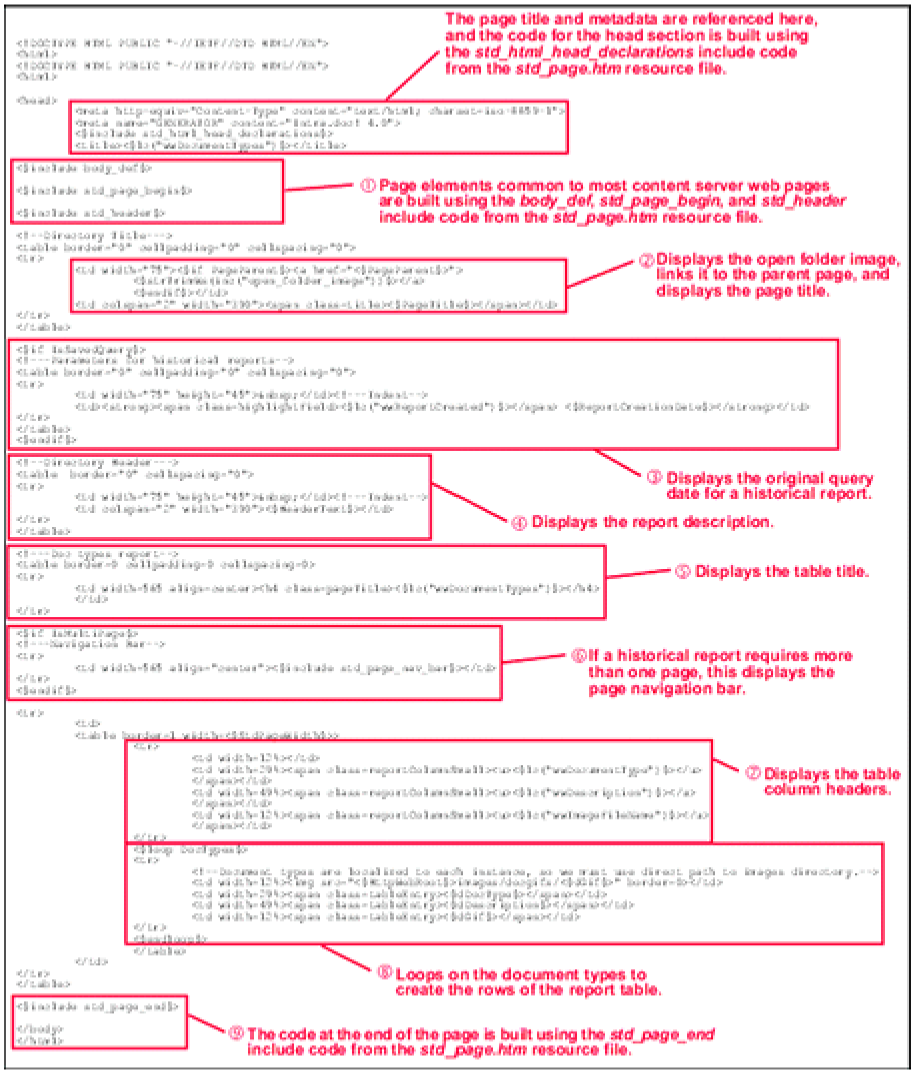
② The links on the Content Management page are built using the include code from the *std_page.htm* resource file.

③ This defines whether a Quick Help button should appear on the Content Management page.

④ The code at the end of the page is built using the *std_page_end* include code from the *std_page.htm* resource file.

**3.5.7.1.2 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*



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

<head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
        <meta name="GENERATOR" content="Intra.Doc! 4.0">
        <$include std_html_head_declarations$>
        <title><$lc("wwDocumentTypes")$></title>
<$include body_def$>

<$include std_page_begin$>

<$include std_header$>
<!--Directory Title-->
<table border="0" cellpadding="0" cellspacing="0">
<tr>
        <td width="75"><$if PageParent$><a href="<$PageParent$>">
                <$strTrimWs(inc("open_folder_image"))$></a>
                <$endif$></td>
        <td colspan="2" width="330"><span class=title><$PageTitle$></span></td>
</tr>
</table>
<$if IsSavedQuery$>
<!---Parameters for historical reports-->
<table border="0" cellpadding="0" cellspacing="0">
<tr>
        <td width="75" height="45">&nbsp;</td><!---Indent-->
        <td><strong><span class=highlightField><$lc("wwReportCreated")$></span> <$ReportCreationDate$></strong></td>
</tr>
</table>
<$endif$>
<!--Directory Header-->
<table  border="0" cellspacing="0">
<tr>
        <td width="75" height="45">&nbsp;</td><!---Indent-->
        <td colspan="2" width="330"><$HeaderText$></td>
</tr>
</table>
<!---Doc types report-->
<table border=0 cellpadding=0 cellspacing=0>
<tr>
        <td width=545 align=center><h4 class=pageTitle><$lc("wwDocumentTypes")$></h4>
        </td>
</tr>
<$if IsMultiPage$>
<!---Navigation Bar-->
<tr>
        <td width=545 align="center"><$include std_page_nav_bar$></td>
</tr>
<$endif$>
<tr>
        <td>
        <table border=1 width=<$StdPageWidth$>>
                <tr>
                        <td width=12%></td>
                        <td width=29%><span class=reportColumnHeader><u><$lc("wwDocumentType")$></u>
                        </span></td>
                        <td width=49%><span class=reportColumnHeader><u><$lc("wwDescription")$></u>
                        </span></td>
                        <td width=12%><span class=reportColumnHeader><u><$lc("wwImageFileName")$></u>
                        </span></td>
                </tr>
                <$loop DocTypes$>
                <tr>
                        <!--Document types are localized to each instance, so we must use direct path to images directory.-->
                        <td width=12%><img src="<$HttpWebRoot$>images/docgifs/<$dGif$>" border=0></td>
                        <td width=29%><span class=tableEntry><$dDocType$></span></td>
                        <td width=49%><span class=tableEntry><$dDescription$></span></td>
                        <td width=12%><span class=tableEntry><$dGif$></span></td>
                </tr>
                <$endloop$>
                </table>
        </td>
</tr>
</table>
<$include std_page_end$>

</body>
</html>
```

The page title and metadata are referenced here, and the code for the head section is built using the *std_html_head_declarations* include code from the *std_page.htm* resource file.
① Page elements common to most content server web pages are built using the *body_def*, *std_page_begin*, and *std_header* include code from the *std_page.htm* resource file.
② Displays the open folder image, links it to the parent page, and displays the page title.
③ Displays the original query date for a historical report.
④ Displays the report description.
⑤ Displays the table title.
⑥ If a historical report requires more than one page, this displays the page navigation bar.
⑦ Displays the table column headers.
⑧ Loops on the document types to create the rows of the report table.
⑨ The code at the end of the page is built using the *std_page_end* include code from the *std_page.htm* resource file.

### 3.5.7.2 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 pane.
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 and click **Edit**. Modify the parameters and click **OK**.

- To remove a template definition, select the definition and click **Delete**. Click **Yes** to confirm.

## 3.5.8 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.
```

***Example 3–2 Environment***

The following is an example of an environment resource file.

```
# Use this to turn on or off alternate row coloring
nsUseColoredRows=0

# These are the nested search field definitions.

nsFld1Caption=Document Text
nsFld1Name=
nsFld1Type=FullText
nsFld1OptionKey=
#
nsFld2Caption=Text
nsFld2Name=xtext
nsFld2Type=Text
nsFld2OptionKey=
#
nsFld3Caption=Date
nsFld3Name=xdate
nsFld3Type=Date
nsFld3OptionKey=
#
nsFld4Caption=Integer
nsFld4Name=xinteger
nsFld4Type=Int
nsFld4OptionKey=
#
nsFld5Caption=Option List
nsFld5Name=xoptionlist
nsFld5Type=OptionList
nsFld5OptionKey=optionlistList
#
nsFld6Caption=Info Topic
nsFld6Name=xwfsInfoTopic
```

```
nsFld6Type=OptionList
nsFld6OptionKey=wfsInfoTopicList
```

The colored_search_resource.htm template resource file in the Nested Search component references the `nsUseColoredRows` variable as follows:

```
<$if isTrue(#active.nsUseColoredRows)$>
    <$useColoredRows=1, bkgHighlight=1$>
<$endif$>
```

Standard configuration variables are defined in the *IntradocDir*/config/config.cfg file. See the *Oracle Fusion Middleware Idoc Script Reference Guide* for a complete list of configuration variables.

#### 3.5.8.1 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. It is recommended that you launch the text editor for easier viewing.

## 3.6 Installing Components

Server components for Content Server are installed by default, however, custom components and components downloaded from Oracle Technology Network must be installed and enabled before they can be used.

> **Note:** If you only need to enable or disable an installed component, see "Enabling and Disabling Components" on page 3-4.

You can install components using one of several methods:

- "Using Component Manager" on page 3-43
- "Using Component Wizard" on page 3-43
- "Using ComponentTool" on page 3-44

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 compoment must have a zip file with the appropriate name and be located inside the custom component or core component directory.

### 3.6.1 Using Component Manager

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

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

   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 select **Continue**, a message appears after successful installation. You can choose one of two options:
   - To enable the component and restart the 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 Using Component Wizard

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

1. Start the Component Wizard:
   - (Windows) Select **Start**, then **Programs**, then **Oracle Content Server**, then your content server , then **Utilities**, then **Component Wizard**.
   - (UNIX) Run the ComponentWizard script in the /bin directory.

   The Component Wizard main screen and the Component List screens 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 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. See the component's documentation for details.

### 3.6.3 Using 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 Content Server Interface

This chapter provides information about the several different methods that you can use to change the look and navigation of the Content Server interface. It covers these topics:

- "Modifying the Content Server Interface" on page 4-1
- "Using Dynamic Server Pages to Alter the Navigation of Web Pages" on page 4-7

---

**Note:** 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. See "Managing Repository Content" in the *Oracle Fusion Middleware Application Administrator's Guide for Content Server* for details about creating and modifying metadata fields and creating content profiles.

---

## 4.1 Modifying the Content Server Interface

This section describes how to modify the Content Server interface:

- "Skins and Layouts" on page 4-1
- "Customizing the Interface" on page 4-3

### 4.1.1 Skins and Layouts

This section provides information about available skins and layouts provided by default with your Content Server. Skins and layouts provide alternate color schemes and alternate navigation designs.

- "Types of Skins and Layouts" on page 4-1
- "Selecting Skins and Layouts" on page 4-2
- "Configuration Entries" on page 4-2
- "Anonymous User Interface" on page 4-3

#### 4.1.1.1 Types of Skins and Layouts

Several skins and layouts are provided by default with the Content Server. In addition, you can design custom skins and layouts. When users changes the skin or layout, they change the look and feel of the interface. They 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 create and make new or custom skins. See "Configuration Entries" on page 4-2 for additional information on setting the default look and feel of the user interface.

---

**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 Menu-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 allow users to change the *layouts* of the Content Server or the *skin.*

---

**Important:** This personalization functionality works with Internet Explorer 6+ or Mozilla Firefox 3+ and later versions.

---

To change the skin or layout, follow these steps:

1. On the Content Server Home page, click ***<your_user_name>*** in the top menu bar. The User Profile page displays.
2. On the 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 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 Content Server-based Web site must be available to external customers without a login, but you want employees to be able to contribute content to that Web site.

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

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

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

You can customize your Web pages to make it easy for customers to search for content, and then give employees a login that permits them to see the interface on login. To do the customization, modify the ExtranetLook.idoc file, which provides dynamic resource includes that can be customized based on user login. The idoc file is checked into the Content Server repository so it can be referenced by the 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. Select **Options**, then **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 the ExtranetLook content item out of the Content Server.
7. Check the revised ExtranetLook.idoc file back into the Content Server.

### 4.1.2 Customizing the Interface

The Top Menu 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 the Content Server or design new skins and layouts by creating components that can be shared with other users.

This section provides an overview of this process. It includes these topics:

- "About Dynamic Publishing" on page 4-4
- "Creating New Layouts" on page 4-4

#### 4.1.2.1 About Dynamic Publishing

When the 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 then merge a custom row which 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 the Content Server returns to its default state.

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

Because this Idoc Script is evaluated once during publishing, you cannot use Idoc Script as you would normally do from the *IdcHomeDir*/resources/core/idoc/std_page.idoc file. When a user requests that file, it has already been created, so the script 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 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 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.

This section discusses the following topics:

- "Bundling Files" on page 4-5
- "Referencing Published Files" on page 4-7

#### 4.1.3.1 Bundling Files

Static weblayout file contents are cached on client machines and on Web proxies, significantly lowering the amount of server bandwidth they use. Therefore, best practice indicates that these types of files should be used 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 this file. This occurs even if the file is cached. Therefore, as the number of these files grows, so does the number of pings to the server for each page request.

To help minimize the number of round-trips, 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 using the `PublishedWeblayoutBundles` table in the std_resources.htm file.

```
<@table PublishedWeblayoutBundles@>
<table border=1><caption><strong>
    <tr><td>class</td><td>bundlePath</td><td>loadOrder</td></tr>
    <tr>
        <td>javascript:common</td>
        <td>resources/layouts/commonBundle.js</td>
        <td>10</td>
    </tr>
    ...
</table>
<@end@>
```

The columns in this table are as follows:

- `class`: This refers to the same column in the `PublishedWeblayoutFiles` table and is used to determine which files are placed in which bundle.
- `bundlePath`: The eventual location where the bundle is published. This path is relative to the /weblayout directory.
- `loadOrder`: The order in which this bundle should be loaded on Content Server pages. Bundles with a lower loadOrder are loaded first.

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.

The `class` column in the `PublishedWeblayoutFiles` and `PublishedWeblayoutBundles` tables is a colon-separated classification. In the following example, two different bundles overlap. `food` accounts for all three published weblayout files, while `food:fruit` accounts for two of the three and `food:vegetable` accounts for the third.

Any given weblayout file can only be published into a single bundle, so `food:fruit` contains both `APPLE` and `PEAR`, while `food` picks up the leftover `CARROT`. The server checks each file to be published then looks for the most specific bundle in which to place it. If no bundle exists, it is published as a single file.

The order in which files are included in a bundle is determined through the `loadOrder` column in the `PublishedWeblayoutFiles` table. Files with a lower `loadOrder` are placed earlier in the bundle.

```
<@table PublishedWeblayoutFiles@>
<table border=1><caption><strong>
    <tr>
        <td>path</td>
        <td>template</td>
        <td>class</td>
        <td>loadOrder</td>
        <td>doPublishScript</td>
    </tr>
    <tr>
        <td>resources/apple</td>
        <td>APPLE</td>
        <td>food:fruit:apple</td>
        <td>10</td>
        <td><$doPublish = 1$></td>
    </tr>
    <tr>
        <td>resources/pear</td>
        <td>PEAR</td>
        <td>food:fruit:pear</td>
        <td>20</td>
        <td><$doPublish = 1$></td>
    </tr>
    <tr>
        <td>resources/carrot</td>
        <td>CARROT</td>
        <td>food:vegetable:carrot</td>
        <td>10</td><td><$doPublish = 1$></td>
    </tr>
</table>
<@end@>

<@table PublishedWeblayoutBundles@>
<table border=1><caption><strong>
    <tr>
        <td>class</td>
        <td>bundlePath</td>
        <td>loadOrder</td>
    </tr>
    <tr>
        <td>food:fruit</td>
        <td>resources/fruit</td>
       <td>20</td>
    </tr>
    <tr>
        <td>food</td>
        <td>resources/food</td>
        <td>10</td>
    </tr>
</table>
<@end@>
```

#### 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 gives you an overview of the building blocks necessary to create dynamic server pages. It includes the following sections:

- "About Dynamic Server Pages" on page 4-7
- "Page Types" on page 4-9
- "Creating Dynamic Server Pages" on page 4-10
- "Syntax" on page 4-10
- "Idoc Script Functions" on page 4-13
- "HCSF Pages" on page 4-14
- "Development Recommendations" on page 4-14

### 4.2.1 About Dynamic Server Pages

Dynamic server pages are files that are checked into the 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**: Hyper Content Server Template, similar to a standard Content Server template page stored in the *IdcHomeDir*/resources/core/templates directory.
- **HCSP**: Hyper Content Server Page, an HTML-compliant version of the HCST page, usually used for published content.
- **HCSF**: Hyper 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, the Content Server assembles Web pages dynamically using a custom template (HCST, HCSP, or HCSF file) that you have checked in to the Content Server. The template calls HTML includes from a text file (IDOC file) you have also checked in to the 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.

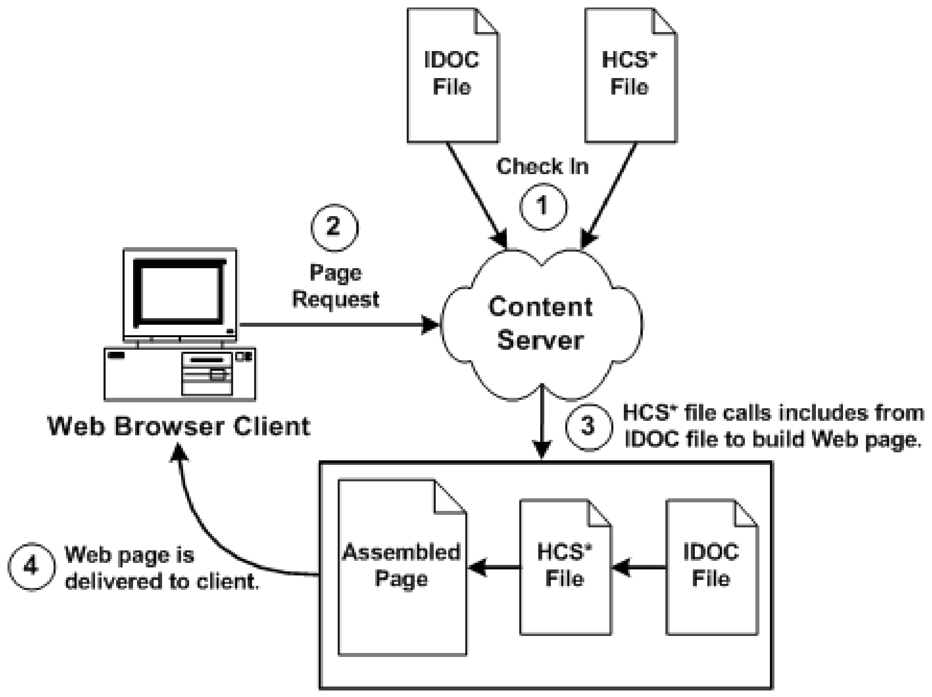
The advantages of using dynamic server pages with the Content Server include the following:

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

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

- **Dynamic server pages cannot be used to modify core functionality of the 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 the 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.

**Note:** See Chapter 3, "Working with Components" for detailed information on includes.

### 4.2.2.2 HCST File

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

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

**Note:** See "HCSF File" on page 4-10 for more detail on HCSF pages.

### 4.2.3 Creating Dynamic Server Pages

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

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

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

**Note:** Using dynamic server pages with Content Publisher can be a powerful tool for Web publishing. See the Content Publisher documentation for more information.

### 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 | <$ … $> | <$ … $> | <!--$ … --> [!--$ … --] | <!--$ … --> [!--$ … --] |

| File Type | .idoc | .hcst | .hcsp | .hcsf |
|---|---|---|---|---|
| Comparison Operators | Symbols (==) | Symbols (==) | Special operators (eq) | Special operators (eq) |
| Special Characters | Symbols **(&)** | Symbols **(&)** | Escape sequence (&amp;) | Escape sequence (&amp;) |
| Referencing Metadata | Required | Required | Required | Required |

**Notes:** Idoc uses standard HTML include coding. (See HTML Includes in the Using Components.)

HCST uses standard Content Server template coding. (See Template and Report Pages in the Using Components.)

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 variable `count` is greater than 10:

| IDOC or HCST File | HCSP or HCSF File |
|---|---|
| `<$if count > 10$>`<br>`    <$"Count is greater than"$>`<br>`<$endif$>` | `<!--$if count gt 10-->`<br>`    <!--$"Count is greater than"-->`<br>`<!--$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 `&amp;` or `&#038;`).

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:** It is especially important to use the `&amp;` escape character when you call the *docLoadResourceIncludes* function from an HCSP or HCSF page. See "docLoadResourceIncludes Function" on page 4-13.

**Tip:** 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

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

- "docLoadResourceIncludes Function" on page 4-13
- "executeService Function" on page 4-14

### 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")-->
```

- 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. See "Special Characters" on page 4-12.

#### 4.2.5.1.1 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.<br>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.<br>**Latest—**The latest checked in revision of the document is used (including revisions in a workflow).<br>**LatestReleased—**The latest released revision of the document is used.<br>**Specific—**Use only with *dID*. |
| Rendition | Specifies which rendition of the IDOC file to use.<br>**Primary—**The primary (native) file. This is the default if no *Rendition* is specified.<br>**Web—**The Web-viewable file.<br>**Alternate—**The alternate file. |

#### 4.2.5.2 executeService Function

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

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

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

- Services that can be called with the `executeService` function must be "scriptable", meaning that they do not require parameter input.
- Scriptable services have an access level of 32 or more. See Chapter 6, "Integration Methods" for more information.
- For a list of standard Content Server services, see the *IdcHomeDir*/resources/core/tables/std_services.htm file.
- For more information on the `executeService` function, see the *Oracle Fusion Middleware Idoc Script Reference Guide*.
- For more information on services, see the Chapter 6, "Integration Methods".

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

### 4.2.6 HCSF Pages

In addition to following the standard formatting rules for Content Server templates and HTML forms, HCSF pages require several special sections and tags that allow the Content Server to process them. See "HCSF Pages" on page 4-16 for more information.

**Note:** See "HCSF File" on page 4-10 for an example of a complete HCSF page.

### 4.2.7 Development Recommendations

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

- "General Tips" on page 4-15
- "HCSF Tips" on page 4-15

#### 4.2.7.1 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 the Content Server, 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 Web master. 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.

#### 4.2.7.2 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 the 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 Content Server when the form is processed by Content Server.

### 4.2.8 HCSF Pages

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

- "Load Section" on page 4-16
- "Data Section" on page 4-17
- "Form Section" on page 4-22

**Note:** See "HCSF File" on page 4-10 for an example of a complete HCSF page.

#### 4.2.8.1 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 is 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 following:

- HTML Declaration
- docLoadResourceIncludes Function
- Meta Tag
- Variables and Includes

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

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

**4.2.8.1.2 docLoadResourceIncludes Function** The docLoadResourceIncludes function loads all the includes from the specified IDOC file for use in assembling the current page. See "docLoadResourceIncludes Function" on page 4-17 for more information.

**4.2.8.1.3 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">
  ```

**4.2.8.1.4 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-->
```

### 4.2.8.2 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.

> **Note:** See *DataBinder* and *ResultSet Section* in Chapter 3, "Working with Components" for more information.

This section covers these topics:

- "Data Section Structure" on page 4-17
- "idcformrules Tag" on page 4-18
- "Metadata Tags" on page 4-18
- "Nested Tags" on page 4-19
- "Referencing XML Tags" on page 4-19
- "Form Elements" on page 4-19
- "ResultSets" on page 4-20

**4.2.8.2.1 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:
  - 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

**4.2.8.2.2 idcformrules Tag** The idcformrules tag defines Content Server-specific rules in the data section. This tag requires one attribute, either the isFormFinished Attribute or resultsets Attribute.

- **iIsFormFinished 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, the Content Server 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 on ResultSet formatting in the data section, see "ResultSets" on page 4-20.

**4.2.8.2.3 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>
```

- **Content Attribute**: 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.

**4.2.8.2.4 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.

**4.2.8.2.5 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
  ```

**4.2.8.2.6 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">
```

**4.2.8.2.7 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.
- See Example 4–2, "Repeated Tags in a ResultSet" and Example 4–3, "Nested Tags in a ResultSet" for limitations on repeating and nesting XML tags in a ResultSet.

***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 the 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.
```

#### 4.2.8.3 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.

**Note:** See "Common Code for Forms" on page 4-29 for additional code examples.

**4.2.8.3.1 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="<$SourceID$>">
```

**4.2.8.3.2 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-->
```

**4.2.8.3.3 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-->
```

**Tip:** 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 Content Server when the form is processed by Content Server.

***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 the following:

```
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.

**4.2.8.3.4 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">
```

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

## 4.2.9 Working with Dynamic Server Pages

This section presents examples that show how the dynamic server pages work together to modify Content Server behavior. It includes the following sections:

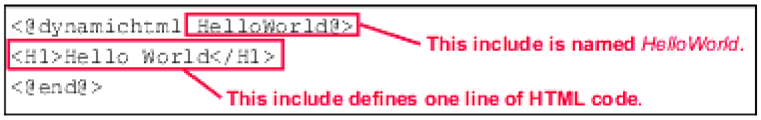
- "HCST and HCSP Example" on page 4-24
- "HCSF Example" on page 4-25
- "Common Code for Forms" on page 4-29

***Example 4–6 HCST and HCSP Example***

This example shows you how to create a simple HCST page and HCSP page:

1. Create an IDOC file with a custom include.

***Figure 4–2 Custom include***

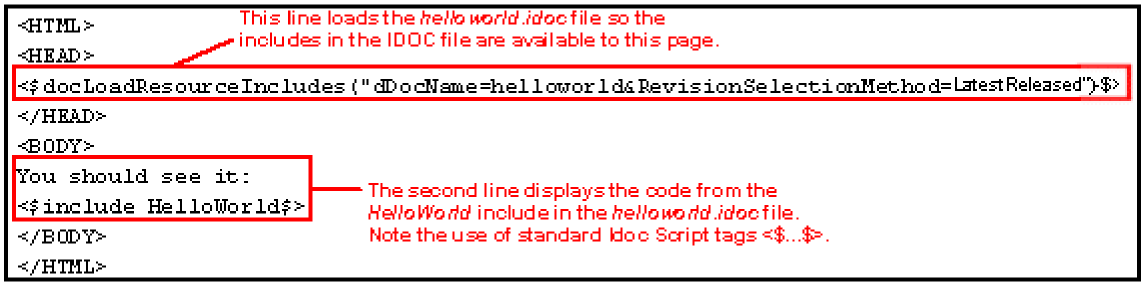


```
<@dynamichtml HelloWorld@>
<H1>Hello World</H1>
<@end@>
```

This include is named *HelloWorld*.
This include defines one line of HTML code.

2. Save the file as `helloworld.idoc`.
3. Check the IDOC file into the 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***

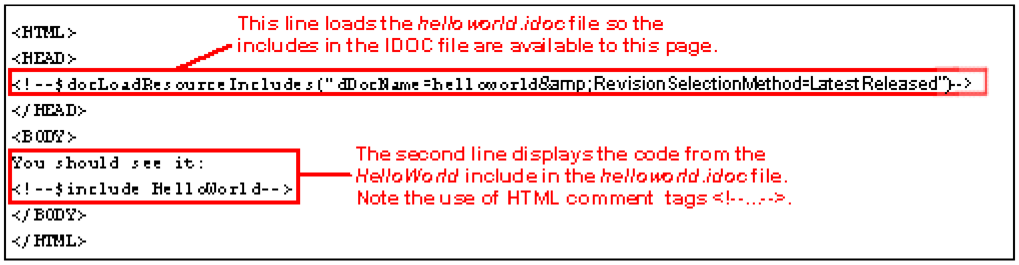


```
<HTML>
<HEAD>
<$docLoadResourceIncludes("dDocName=helloworld&RevisionSelectionMethod=LatestReleased")$>
</HEAD>
<BODY>
You should see it:
<$include HelloWorld$>
</BODY>
</HTML>
```

This line loads the *helloworld.idoc* file so the includes in the IDOC file are available to this page.
The second line displays the code from the *HelloWorld* include in the *helloworld.idoc* file. Note the use of standard Idoc Script tags <$...$>.

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

***Figure 4–4 HCSP file referencing custom include***

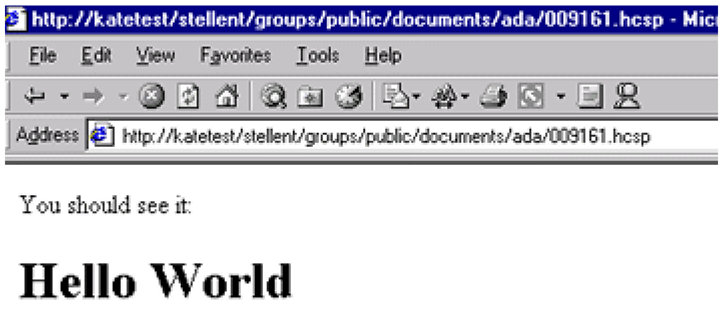


```
<HTML>
<HEAD>
<!--$docLoadResourceIncludes("dDocName=helloworld&amp;RevisionSelectionMethod=LatestReleased")-->
</HEAD>
<BODY>
You should see it:
<!--$include HelloWorld-->
</BODY>
</HTML>
```

This line loads the *helloworld.idoc* file so the includes in the IDOC file are available to this page.
The second line displays the code from the *HelloWorld* include in the *helloworld.idoc* file. Note the use of HTML comment tags <!--...-->.

8. Save the file as `helloworld.hcsp`.
9. Check the HCSP file into the Content Server.
10. Search for the `helloworld` content items in the Content Server.
11. Display the HCST file and HCSP files in your Web browser. They should both look like this:

***Figure 4–5 HelloWorld content item displayed in a Web browser.***



http://katetest/stellent/groups/public/documents/ada/009161.hcsp - Mic
File
Edit
View
Favorites
Tools
Help
Address
http://katetest/stellent/groups/public/documents/ada/009161.hcsp

You should see it:

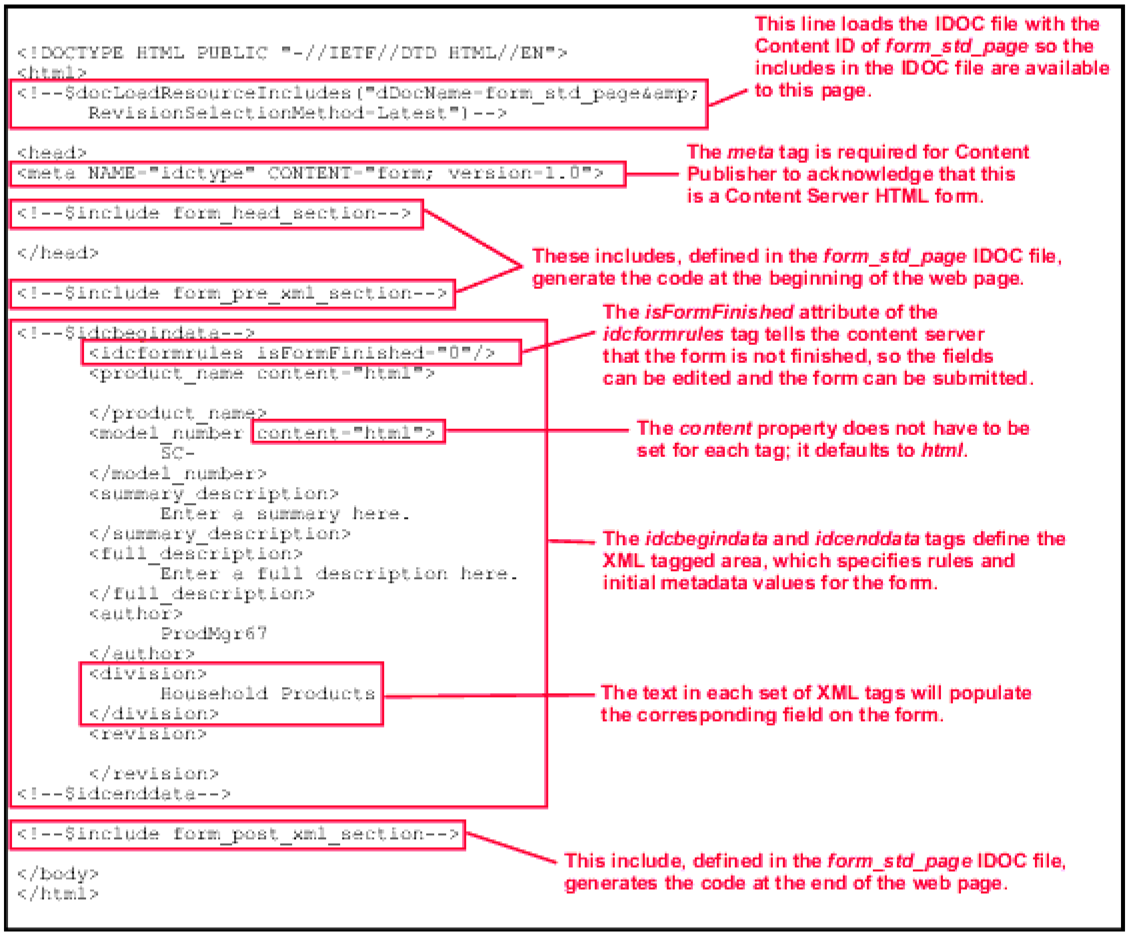
Hello World

***Example 4–7 HCSF Example***

This example shows you 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.

1. Create an HCSF file that references an IDOC file named `form_std_page`:

***Figure 4–6 Product description form HCSF file.***



```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<!--$docLoadResourceIncludes("dDocName=form_std_page&amp;
      RevisionSelectionMethod=Latest")-->

<head>
<meta NAME="idctype" CONTENT="form; version=1.0">
<!--$include form_head_section-->

</head>

<!--$include form_pre_xml_section-->

<!--$idcbegindata-->
      <idcformrules isFormFinished="0"/>
      <product_name content="html">

      </product_name>
      <model_number content="html">
            SC-
      </model_number>
      <summary_description>
            Enter a summary here.
      </summary_description>
      <full_description>
            Enter a full description here.
      </full_description>
      <author>
            ProdMgr67
      </author>
      <division>
            Household Products
      </division>
      <revision>

      </revision>
<!--$idcenddata-->

<!--$include form_post_xml_section-->

</body>
</html>
```

This line loads the IDOC file with the Content ID of *form_std_page* so the includes in the IDOC file are available to this page.

The *meta* tag is required for Content Publisher to acknowledge that this is a Content Server HTML form.

These includes, defined in the *form_std_page* IDOC file, generate the code at the beginning of the web page.

The *isFormFinished* attribute of the *idcformrules* tag tells the content server that the form is not finished, so the fields can be edited and the form can be submitted.

The *content* property does not have to be set for each tag; it defaults to *html*.

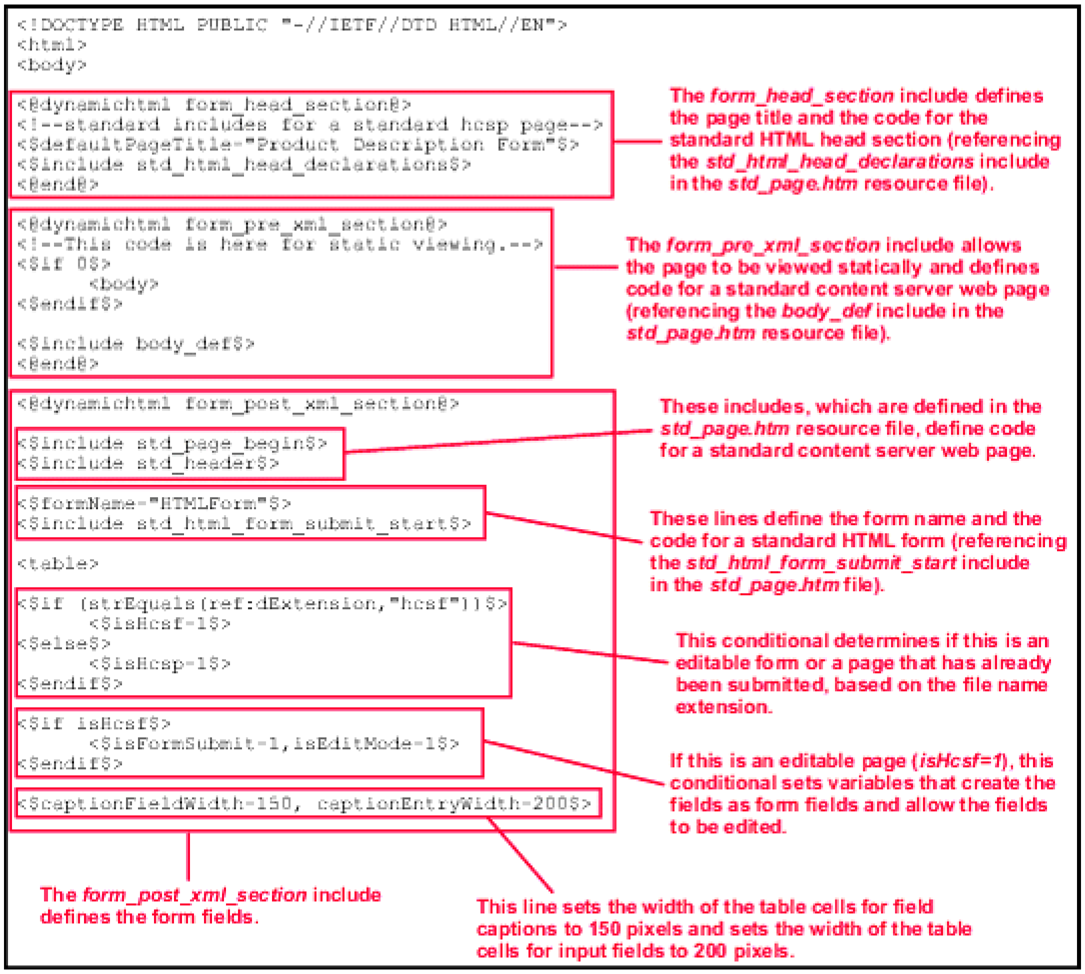
The *idcbegindata* and *idcenddata* tags define the XML tagged area, which specifies rules and initial metadata values for the form.

The text in each set of XML tags will populate the corresponding field on the form.

This include, defined in the *form_std_page* IDOC file, generates the code at the end of the web page.

2. Save the file as `product_form.hcsf`.
3. Check the HCSF file into the Content Server.
4. Create an IDOC file with custom includes:

***Figure 4–7 IDOC file with custom includes***



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

<@dynamichtml form_head_section@>
<!--standard includes for a standard hcsp page-->
<$defaultPageTitle="Product Description Form"$>
<$include std_html_head_declarations$>
<@end@>

<@dynamichtml form_pre_xml_section@>
<!--This code is here for static viewing.-->
<$if 0$>
        <body>
<$endif$>

<$include body_def$>
<@end@>

<@dynamichtml form_post_xml_section@>

<$include std_page_begin$>
<$include std_header$>

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

<table>

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

<$if isHcsf$>
        <$isFormSubmit=1,isEditMode=1$>
<$endif$>

<$captionFieldWidth=150, captionEntryWidth=200$>
```

The *form_head_section* include defines the page title and the code for the standard HTML head section (referencing the *std_html_head_declarations* include in the *std_page.htm* resource file).

The *form_pre_xml_section* include allows the page to be viewed statically and defines code for a standard content server web page (referencing the *body_def* include in the *std_page.htm* resource file).

These includes, which are defined in the *std_page.htm* resource file, define code for a standard content server web page.

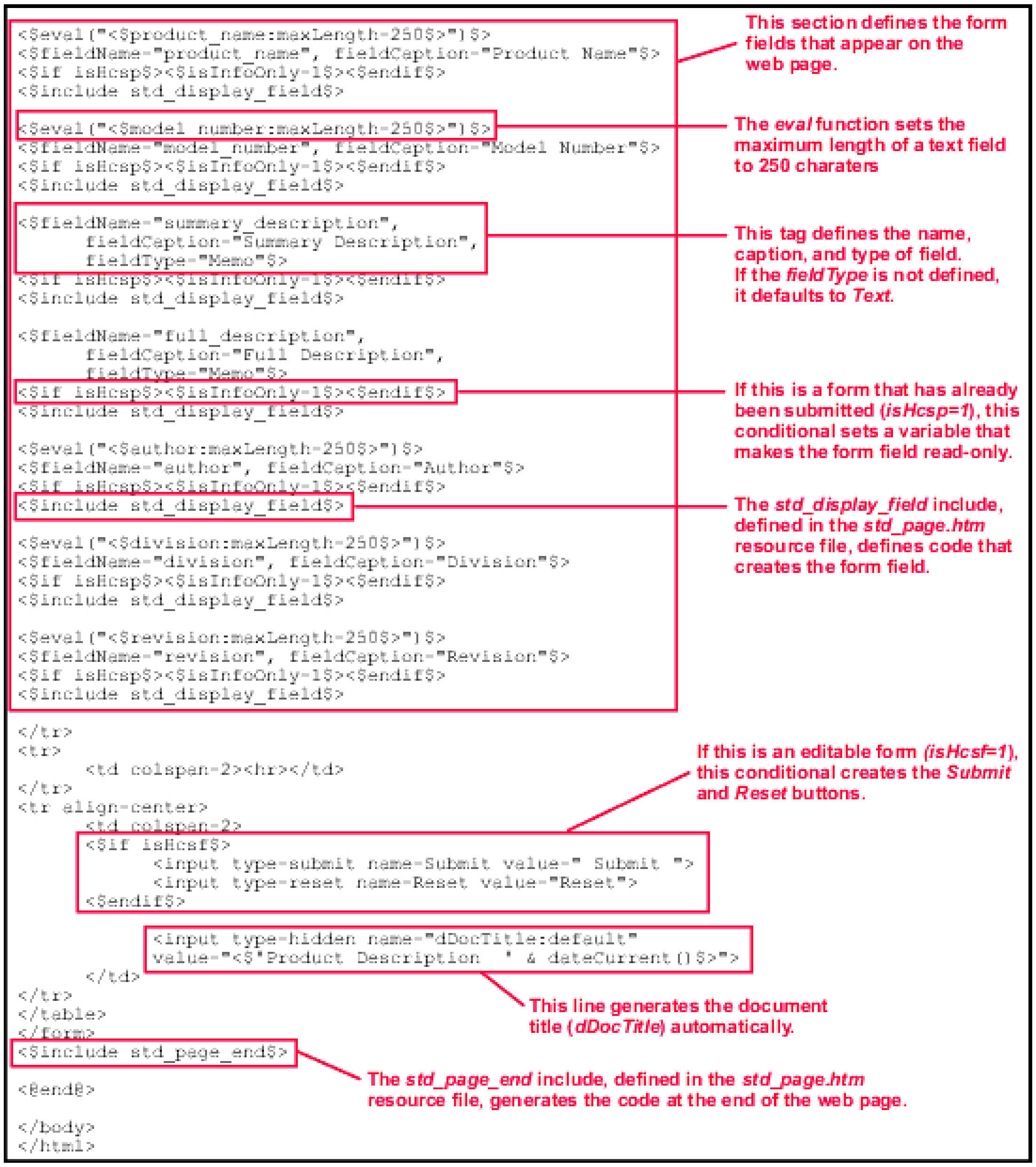
These lines define the form name and the code for a standard HTML form (referencing the *std_html_form_submit_start* include in the *std_page.htm* file).

This conditional determines if this is an editable form or a page that has already been submitted, based on the file name extension.

If this is an editable page (*isHcsf=1*), this conditional sets variables that create the fields as form fields and allow the fields to be edited.

The *form_post_xml_section* include defines the form fields.

This line sets the width of the table cells for field captions to 150 pixels and sets the width of the table cells for input fields to 200 pixels.



```
<$eval("<$product_name:maxLength=250$>")$>
<$fieldName="product_name", fieldCaption="Product Name"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$model_number:maxLength=250$>")$>
<$fieldName="model_number", fieldCaption="Model Number"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$fieldName="summary_description",
      fieldCaption="Summary Description",
      fieldType="Memo"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$fieldName="full_description",
      fieldCaption="Full Description",
      fieldType="Memo"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$author:maxLength=250$>")$>
<$fieldName="author", fieldCaption="Author"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$division:maxLength=250$>")$>
<$fieldName="division", fieldCaption="Division"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

<$eval("<$revision:maxLength=250$>")$>
<$fieldName="revision", fieldCaption="Revision"$>
<$if isHcsp$><$isInfoOnly=1$><$endif$>
<$include std_display_field$>

</tr>
<tr>
      <td colspan=2><hr></td>
</tr>
<tr align=center>
      <td colspan=2>
      <$if isHcsf$>
            <input type=submit name=Submit value=" Submit ">
            <input type=reset name=Reset value="Reset">
      <$endif$>

            <input type=hidden name="dDocTitle:default"
            value="<$'Product Description  ' & dateCurrent()$>">
      </td>
</tr>
</table>
</form>
<$include std_page_end$>

<@end@>

</body>
</html>
```

This section defines the form fields that appear on the web page.

The *eval* function sets the maximum length of a text field to 250 charaters

This tag defines the name, caption, and type of field. If the *fieldType* is not defined, it defaults to *Text*.

If this is a form that has already been submitted (*isHcsp=1*), this conditional sets a variable that makes the form field read-only.

The *std_display_field* include, defined in the *std_page.htm* resource file, defines code that creates the form field.

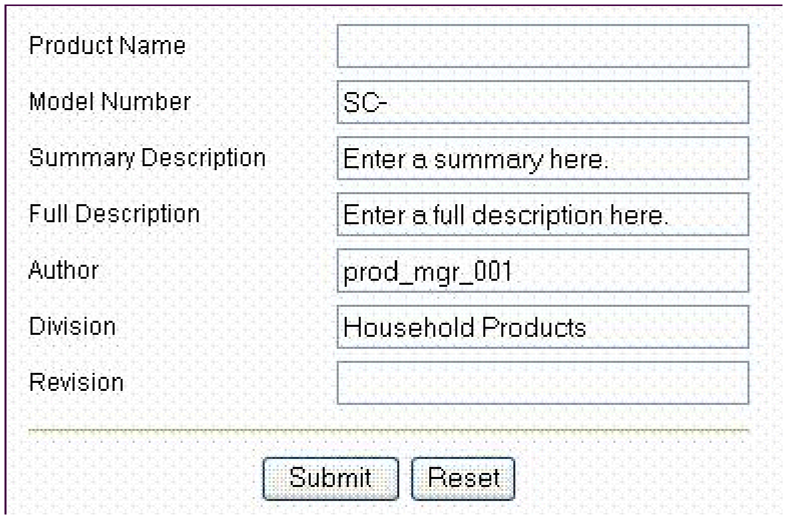
If this is an editable form (*isHcsf=1*), this conditional creates the *Submit* and *Reset* buttons.

This line generates the document title (*dDocTitle*) automatically.

The *std_page_end* include, defined in the *std_page.htm* resource file, generates the code at the end of the web page.

5. Save the file as `form_std_page.idoc`.
6. Check the IDOC file into the 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 the Content Server.
8. Click the link to display the HCSF page in your Web browser. It should look like this:

*Figure 4–8 Sample form displayed in a Web browser.*



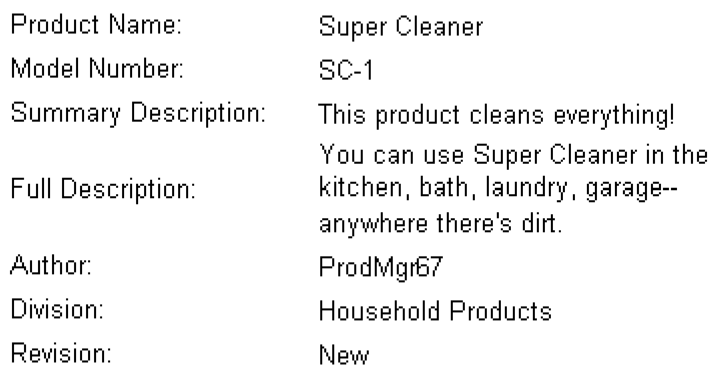
| | |
|---|---|
| Product Name | |
| Model Number | SC- |
| Summary Description | Enter a summary here. |
| Full Description | Enter a full description here. |
| Author | prod_mgr_001 |
| Division | Household Products |
| Revision | |

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 the Content Server.
11. Click the link to display the HCSP page in your Web browser. It should look like this:

*Figure 4–9 Link displaying 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 |

### 4.2.9.1 Common Code for Forms

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

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

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

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

```
<$if (strEquals(ref:dExtension,"hcsf"))$>
```

```
    <$isHcsf=1$>
<$else$>
    <$isHcsp=1$>
<$endif$>
```

---

**Note:** See "Referencing Metadata" on page 4-12 for information on the *ref:* prefix.

---

**4.2.9.1.3 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%>
<$isEditMode=1,isFormSubmit=1$>
<$captionFieldWidth="25%", captionEntryWidth="75%"$>
```

**4.2.9.1.4 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@>
```

**4.2.9.1.5 Defining Hidden Fields** You can specify metadata for a submitted form (hcsp) by defining a hidden field, which contributors cannot change. For example, 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">
```

**4.2.9.1.6 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 provides an overview of several methods for changing the basic functionality of the Content Server. It includes these topics:

## 5.1 Changing System Settings

Content Server has a number of features that you can set up to change features systemwide according to your needs. For example, you can use the following administration tools within the 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 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 the server that is running, and you can check console output.
- **System Properties:** System Properties is an administration application that is used to configure systemwide Content Server settings for content security, internet settings, localization, and other types of settings. Options on the application can:
  - set optional functionality for the Content Server instance
  - set options related to content item security
  - set options related to the Internet and Web interaction
  - set JDBC connectivity options
  - set functionality such as time zones and IP filters
  - set localization features
  - set 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, select **Web Layout Editor** from 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 the *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, click **Configuration Manager** from the **Admin Applets**, then select **User Admin** from the **Apps** menu. 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 the Content Server at run time. The **component architecture** model is derived from object-oriented technologies, and encourages the use of small modules to customize the 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 the Content Server. For example, a component could be used to:

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

The advantages of using component architecture with the Content Server include the following:

- **You can modify source code without compromising the integrity of the product.** The 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 Content Server instances. Many custom components can work on 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 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 a Content Server without compromising customizations.** Custom components override existing product resources rather than replace them. Replacing the standard 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 Content Server core functionality.** Because new functionality may change the way your components behave, customizations are not guaranteed to work for future 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 Content Server. Components provided with 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 Content Server instances. For details see "Component Wizard" on page 3-1.
- The Advanced Component Manager provides a way to manage custom components in the Content Server. By using the Advanced Component Manager, you can add new components and enable or disable components on the Content Server. For details see "Advanced Component Manager" on page 3-2.
- The ComponentTool is a command-line utility for installing, enabling, and disabling components in Content Server.

For information on component architecture and creation, see Chapter 3, "Working with Components."

## 5.3 Changing Configuration Information

For advanced customizations and integration with other business systems, 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 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 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 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 Content Server behavior. Variable values can be stored in an environment resource, such as the config.cfg file and many are predefined in the Content Server. You can also define your own custom variables.
- for **functions**. Many built-in global functions are used in the 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.

See the *Oracle Fusion Middleware Idoc Script Reference Guide* for details on usage and for syntax and configuration variable information.

## 5.4 Customizing Services

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

Any service can be called externally (from outside the Content Server) or internally (from within the Content Server). Client services are usually called externally while administrative services are called internally. The service uses its own attributes and actions to execute the request, based on any parameters passed to the service.

The standard Content Server services are defined in the StandardServices table in *DomainHome*/resources/core/tables/std_services.htm. A service definition contains three main elements:

- The service **name**.
- The service **attributes**. 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 sub-service 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 the Content Server. See the Chapter 6, "Integration Methods" for more details.

## 5.5 Generating Action Menus

In previous versions of Content Server, when a component writer wanted to create an HTML table like those used on the search results page, HTML code had to be copied and pasted. The information in the tables was mixed with the HTML, with no separation between data and display.

The same issue was true for action menus. Data and display for the tables and menus were tightly coupled, making it impossible to perform global changes to all tables in the 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. Each type is discussed in the follow sections:

- "Headline View Tables" on page 5-6
- "Thumbnail View Tables" on page 5-8

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 IdocScript 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 IdocScript 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 are:

- `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 IdocScript 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 IdocScript 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 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

# Integration Methods

This chapter describes options for integrating Content Server with enterprise applications. It covers the following topics:

## 6.1 Overview of Integration Methods

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

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 content server services. The various Content Server services are the "window" for accessing the content and content management functions within Content Server. 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 COM integrations, 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 Content Server within network system environments using various protocols, interfaces, and mapping services.

## 6.2 IdcCommand Utility

This section provides information on using the IdcCommand utility to access content server services from other applications. It covers these topics:

### 6.2.1 Overview of IdcCommand

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

The program reads a 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 the Content Server in an interactive session, use the Java COM wrapper IdcCommandX, available on Microsoft Windows platforms. See "IdcCommand Utility" on page 6-2 for additional information.

To run the IdcCommand utility, the following parameters must be specified on the command line or in the intradoc.cfg configuration file:

- A command file containing the service commands and parameters.
- A content server user name. 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 on using services in custom components, see the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management* and Chapter 3, "Working with Components".

### 6.2.2 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. See "Running IdcCommand" on page 6-6 for more information.

## 6.2.3 Command File

The command file defines the service commands and parameters that are executed by the IdcCommand utility. The command file follows these rules:

- Command File Syntax
- Precedence
- Special Tags and Characters

### 6.2.3.1 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@email.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@email.com
@end
@ResultSet UserAttribInfo
2
dUserName
AttributeInfo
pwallek
role,contributor,15,account,marketing,7
@end
<<EOD>>
```

#### 6.2.3.2 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.

#### 6.2.3.3 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. See "Command File Syntax" on page 6-3 for an example. |
| # | 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. |

### 6.2.4 Configuration Options

To run the IdcCommand utility, specify the following 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.

#### 6.2.4.1 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. See "Command File" on page 6-3 for more information.

#### 6.2.4.2 User

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

#### 6.2.4.3 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.

#### 6.2.4.4 Connection Mode

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

| Connection Mode | Description |
|---|---|
| auto | IdcCommand attempts to connect to the Content Server instance. If this fails, services are executed in standalone mode.<br>This is the default connection mode. |
| server | IdcCommand executes services only through the content server. |
| standalone | IdcCommand executes services in a standalone session.<br>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. |

### 6.2.5 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. See Configuration Options for more information.

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

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

   ```
   IdcCommand.exe
   ```

### 6.2.6 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 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 Content Server may change or remove these variable names.

If the intradoc.cfg file does not contain a value for `IdcHomeDir`, the Launcher sets `IdcHomeDir` to be `$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. $IdcResourcesDir/core/config/launcher-local.cfg
8. `$BIN_DIR/intradoc.cfg`
9. `$BIN_DIR/intradoc-$PLATFORM.cfg`
10. All files specified on the command line, using the `-cfg` option.

**Note:** You can assign variable values directly on the command line by using the `-cfg` option *NAME*=*VALUE*.

#### 6.2.6.1 Quoting

The Launcher uses Bourne Shell-like quoting rules. A string can be quoted inside double quotes (") 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 non-quoted spaces. Each string is then unquoted and used as an entry in the `argv` array for the command.

#### 6.2.6.2 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=$JAVA_HOME
OS_DIR=$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 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 classpath 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 JAVA_CLASSPATH is set to a very long classpath.

- 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` 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`.

#### 6.2.6.3 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 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 timeout used for waiting for a Java process to successfully start on win32.

**Tip:** By using the `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 the many ways you could configure the Launcher.

### 6.2.6.4 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.

#### 6.2.6.5 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` or `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, other entries can be included to alter the way the Java application is launched.

#### 6.2.6.6 Configuration File Example

Configuration file example entries:

```
<?cfg jcharset="Cp1252"?>
#Content Server Directory Variables
IntradocDir=C:/domain/idcm1/
BASE_JAVA_CLASSPATH_source=$IdcResourcesDir/classes
BASE_JAVA_CLASSPATH_serverlegacy=$SharedDir/classes/server.zip
BASE_JAVA_CLASSPATH_server=$JLIB_DIR/idcserver.jar
```

This is sufficient to launch nearly all Content Server applications. Others, such as the Inbound Refinery, require additional classes in the classpath. 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. See the source code for `IdcServer.java` or `IdcAdmin.java` for more details on how to make any Java application run as a Windows service with the Launcher.

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 JDBC drivers for their particular database upon install. If you want to use alternate JDBC drivers, place them outside of the Content Server's `IdcHomeDir`, and alter the JDBC_JAVA_CLASSPATH_customjdbc entry in the intradoc.cfg file with the location of the driver.

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

```
#customized for running IBM's VM
JAVA_EXE=full path
```

When using a custom JVM, use the full path to the Java executable file to be used.

**Caution:** It is recommended that you not override the JVM command line. If you do so, start with the $IdcHomeDir/resources/core/config/launcher.cfg file. Customization is more complicated because of the custom classloader.

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`* or another similar value.

The following are commonly used command line options. Those options noted with an asterisk (*) are available on Windows platforms only. Unmarked options are available for Windows or UNIX platforms:

| Option | Description |
|---|---|
| `-console` | * Forces the Launcher to keep a Windows console window open so that the Java output and error streams are printed to the console. |
| `-debug` | Shows paths and variables in use at startup, and startup errors. Also enables Java debugging in Content Server; when repeated this increases verbosity. |
| `-fileDebug` | Similar to the `-debug` option but this option dumps debug data to the debug.log file. It is usually only set in `JAVA_OPTIONS` or `JAVA_SERVICE_EXTRA_OPTIONS` in the intradoc.cfg file to debug Windows services. |
| `-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. |

| Option | Description |
|---|---|
| `-remove` | * Same as `-uninstall`. |
| `-dependent` *service-name* | * Makes the Windows service dependent on whether the service *service-name* is also running.<br><br>This command is useful when you want to make a dependent call for each service.<br><br>For example, if you want to launch a database before starting the content server, you can specify the content server startup to be dependent on the database startup. |
| `-dependent` *user password* | * Used with `-install`, installs the service with the credentials of the user specified by *user* with password *password*.<br><br>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.<br><br>For certain services, such as the 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 Content Server service without using that Content Server's Launcher (for example, if an entire installation directory has been removed). |

**Tip:** To customize the classpath to alter the system path to load Oracle .dll files, you can set teh pathway to:

```
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 .dlls, you should put them in the *IdcHomeDir*/native/win32/lib/ directory.

### 6.2.7 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 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

## 6.3 COM Integration

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.

You can use a COM interface to integrate Content Management with Microsoft environments and applications. An ActiveX control and an OCX component are provided as interface options to gain access to the content and content management functions within Content Server. Additionally, you can communicate with ODMA-aware applications through a COM interface.

**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.

This section covers these topics:

- "ActiveX Interface" on page 6-14
- "OCX Interface" on page 6-32
- "ODMA Integration" on page 6-58

### 6.3.1 ActiveX Interface

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

**Note:** A Visual Basic or Visual C++ development environment is required to use 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 it fails.
- 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.

The following sections cover these topics:

#### 6.3.1.1 Setting Up IdcCommandUX

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

#### 6.3.1.2 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 content server directory that contains the intradoc.cfg configuration file.

   Example:

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

#### 6.3.1.3 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.

#### 6.3.1.4 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 on HDA formats, see Chapter 3, "Working with 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 content server. 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.

  See the *Oracle Fusion Middleware Idoc Script Reference Guide* for more information.

  See "Using the Launcher" on page 6-6 for information on using the Launcher (a native C++ application that allows a Java program to start as a Windows service).

#### 6.3.1.5 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" on page 6-17
2. "Initializing the Connection" on page 6-17
3. "Defining Services and Parameters" on page 6-18
4. "Referencing Custom Resources" on page 6-18
5. "Executing the Service" on page 6-18
6. "Retrieving Results" on page 6-19

***Example 6–1 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 6–2 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 6–3 Creating the COM Object***

The first line of code creates the COM object:

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

***Example 6–4 Initializing the Connection***

To initialize the connection to the Content Server, call the initRemote function (see "initRemote" on page 6-32 for details about all parameters). In this example:

- The `HttpWebRoot` parameter specifies a value for the Web root as defined in the config/config.cfg file.
- The `idcReference` parameter specifics a string containing information on connection to the Content Server instance. This is specified as "socket" followed by the IntradocServerHostName and the IntradocServer Port address.
- The `idcUser` is the user you are connecting as.
- 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.

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

***Example 6–5 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. See "Formatting with a Resource Include" on page 6-19.

***Example 6–6 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. See "Formatting with a Resource Include" on page 6-19 for more information.
- 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. See "Defining Services and Parameters" on page 6-18.

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

***Example 6–7 Executing the Service***

To execute the service, call the executeCommand method.

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

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

***Example 6–8 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)
```

#### 6.3.1.6 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 "Calling IdcCommandUX from an Active Server Page (ASP)" on page 6-16, 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@>
```

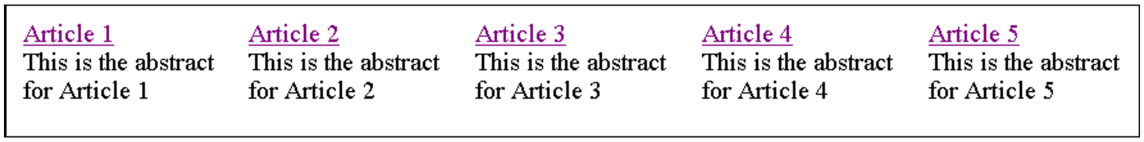
- The `<@dynamichtml ASP_SearchResults@>` entry defines the name of the resource include. The `<@end@>` entry ends the resource definition.
- The code defined between the `<$loop SearchResults$>` and `<$endloop$>` entries is executed for each content item in the SearchResults ResultSet, which includes 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 this:



| | | | | |
|---|---|---|---|---|
| Article 1<br>This is the abstract for Article 1 | Article 2<br>This is the abstract for Article 2 | Article 3<br>This is the abstract for Article 3 | Article 4<br>This is the abstract for Article 4 | Article 5<br>This is the abstract for Article 5 |

### 6.3.1.7 Connect to Content Server from a Remote System

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

1. "Creating Variables" on page 6-21
2. "Creating a COM Object" on page 6-21
3. "Initializing the Connection" on page 6-21
4. "Returning the Connection Status" on page 6-22
5. "Defining the Service and Parameters" on page 6-22
6. "Executing the Service" on page 6-23
7. "Retrieving Results" on page 6-23

***Example 6–9 Coding the ASP Page***

This example calls the CHECKIN_UNIVERSAL service to provide a checkin 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 6–10 Creating Variables***

The following variables must be created:

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

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

***Example 6–11 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 Content Server instance.
- **str**: The HDA-formatted string that defines the service and its parameters.

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

***Example 6–12 Initializing the Connection***

Initialize the connection to the Content Server instance.

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

***Example 6–13 Returning the Connection Status***

In this example, the connectToServer and closeServerConnections 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 6–14 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. See the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management* for additional information.

```
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 6–15 Executing the Service***

To execute the service, call the executeCommand method.

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

***Example 6–16 Retrieving Results***

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

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

## 6.3.2 IdcCommandUX Methods

This section describes the following IdcCommandUX methods:

- "addExtraheadersForCommand" on page 6-23
- "closeServerConnections" on page 6-24
- "computeNativeFilePath" on page 6-24
- "computeURL" on page 6-26
- "computeWebFilePath" on page 6-28
- "connectToServer" on page 6-29
- "executeCommand" on page 6-30
- "executeFileCommand" on page 6-31
- "forwardRequest" on page 6-31
- "getLastErrorMessage" on page 6-31
- "initRemote" on page 6-32

**Important:** All parameters are required unless otherwise indicated.

### 6.3.2.1 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)
```

### 6.3.2.2 closeServerConnections

```
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
```

### 6.3.2.3 computeNativeFilePath

```
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 the 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 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 */stellen*t/. This entry is required for a relative URL, and is optional for an absolute URL.
  - **HttpServerAddress**: The domain name of the Content Server, such as *testserver17* or *mycomputer.com*. (The server address is specified as a partial URL such as *mycomputer.com* rather than a full address such as *http://www.mycomputer.com*/). This entry is required for an absolute URL, and 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* 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";
```

### 6.3.2.4 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 the 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 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 Content Server, such as *testserver17* or *mycomputer.com*. (The server address is specified as a partial URL such as *mycomputer.com* rather than a full address such as *http://www.mycomputer.com/*). This entry is required for an absolute URL, and 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* 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";
```

#### 6.3.2.5 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* 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.oracle.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";
```

### 6.3.2.6 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
```

### 6.3.2.7 executeCommand

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

**Description**

Executes a 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 the previous example ("SOAP Example" on page 6-16). See also "initRemote" on page 6-32.

**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)
```

### 6.3.2.8 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 the 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 the 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.
- See "executeCommand" on page 6-30 for more information.

#### Parameters

None

### 6.3.2.9 forwardRequest

```
forwardRequest()
```

#### Description

This function is used to forward a multipart form post to the Content Server. This is useful for executing checkins.

#### Parameters

None

### 6.3.2.10 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 host name for making a connection, this method returns the connection error. It does not return a value if the error is returned by the 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
```

### 6.3.2.11 initRemote

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

#### Description

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

#### Required Parameters

- **HttpWebRoot**: The IdocScript value for `HttpWebRoot`.
- **idcReference**: A string containing information on how to connect to the Content Server 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")
```

## 6.3.3 OCX Interface

The IntradocClient OCX component is used within a Windows Visual Basic development environment to gain access to the content and content management functions within Content Server. The OCX integration is designed to call services in a visual development environment, or to connect to a remote Content Server.

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.

See the *Oracle Fusion Middleware Services Reference Guide for Universal Content Management* for additional information. This guide also contains information about the IntradocClient OCX API specifications listing the properties, methods, and events.

## 6.3.4 IdcClientOCX Component

An Object Linking and Embedding Control Extension (OCX) control is provided for connecting to a remote Content Server.and executing Content Server services. The IdcClient OCX control is used within a Windows Visual Basic development environment to gain access to the content and content management functions within Content Server.

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 Content Server and perform typical server functions.

This section covers the following topics:

- "IdcClient OCX Description" on page 6-33
- "IdcClient OCX Control Setup" on page 6-35
- "Events, Methods, and Properties" on page 6-34

**Note:** A Visual Basic or Visual C++ development environment is required to use the IdcClient OCX component.

### 6.3.4.1 IdcClient OCX Description

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

**6.3.4.1.1 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 the 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*.

**6.3.4.1.2 Events, Methods, and Properties** The IdcClient OCX control is used to connect to a remote Content Server 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.

  See "Events, Methods, and Properties" on page 6-34 for additional information.

***Example 6–17 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.

See "Events, Methods, and Properties" on page 6-34 for additional information.

***Example 6–18 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.

See "Events, Methods, and Properties" on page 6-34 for additional information.

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

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.

#### 6.3.4.2 IdcClient OCX Control Setup

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

##### 6.3.4.2.1 Component Setup

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** then select **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 check box for *IdcClient ActiveX Control* module is enabled and click **OK**.

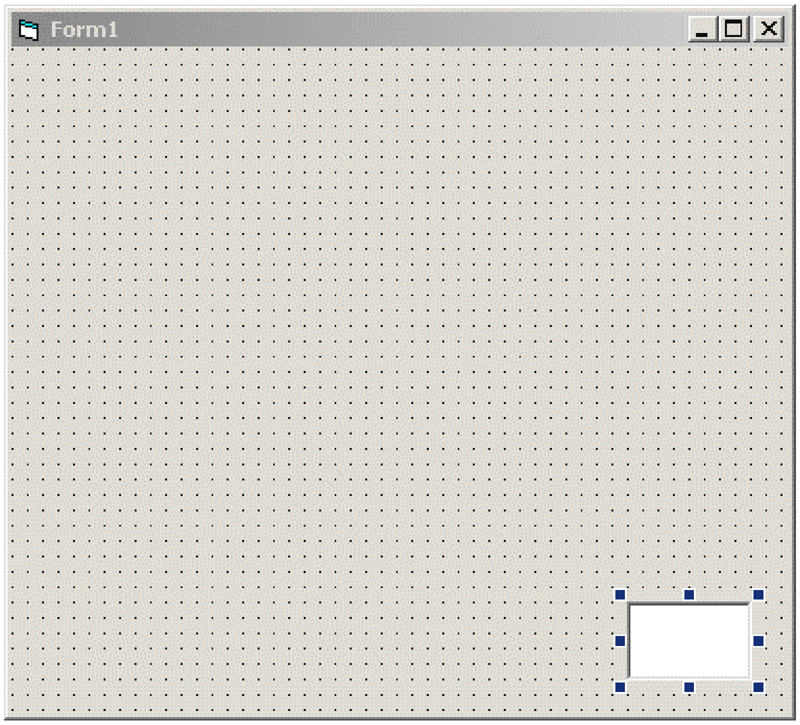
   The IdcClient OCX control is placed in the list of controls.
5. (Optional) You can use the Visual Basic development environment to build your own visual interface or follow the steps provided in "Creating a Visual Interface" on page 6-35 to build a basic visual interface.

##### 6.3.4.2.2 Creating a Visual Interface

**Note:** It is assumed that a Visual Basic project has been created and the IdcClient OCX control has been placed in the list of controls. See "Component Setup" on page 6-35 for additional information.

Follow these steps to build a basic visual interface:

***Figure 6–1 OCX control drawn on a Visual Basic form***

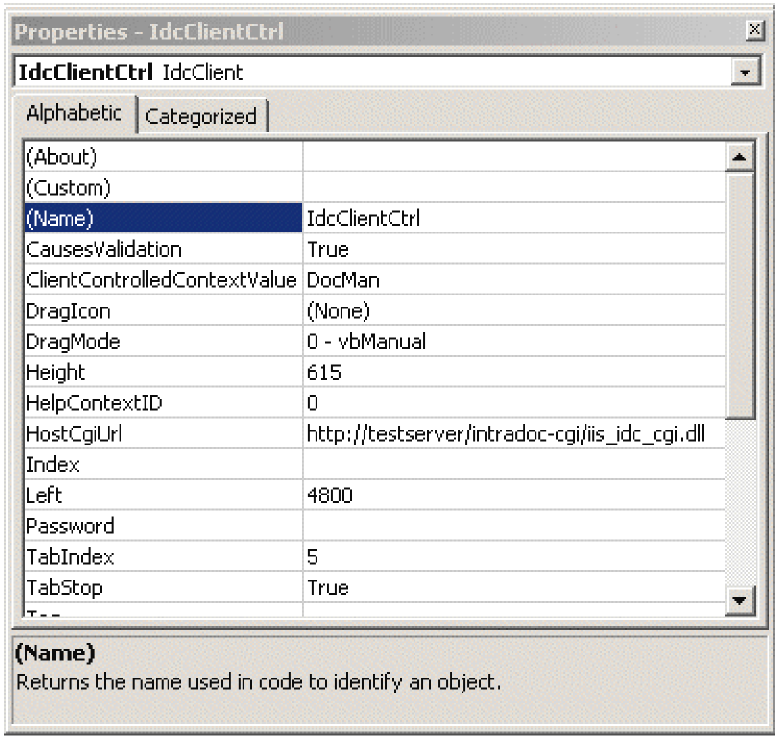


1. Select the control and draw it on the Visual Basic form.
2. From the drop-down list of the Properties Window, select the IdcClient OCX control (if the Properties Window is not currently displayed select **View** then select **Properties Window** from the main menu).
3. Rename the IdcClient OCX control IdcClientCtrl.
4. Define the HostCgiUrl to reference the iss_idc_cgi.dll for your particular instance.

   For example:

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

*Figure 6–2 Edited IdcClient Properties*



| | |
|---|---|
| (About) | |
| (Custom) | |
| (Name) | IdcClientCtrl |
| CausesValidation | True |
| ClientControlledContextValue | DocMan |
| DragIcon | (None) |
| DragMode | 0 - vbManual |
| Height | 615 |
| HelpContextID | 0 |
| HostCgiUrl | http://testserver/intradoc-cgi/iis_idc_cgi.dll |
| Index | |
| Left | 4800 |
| Password | |
| TabIndex | 5 |
| TabStop | True |

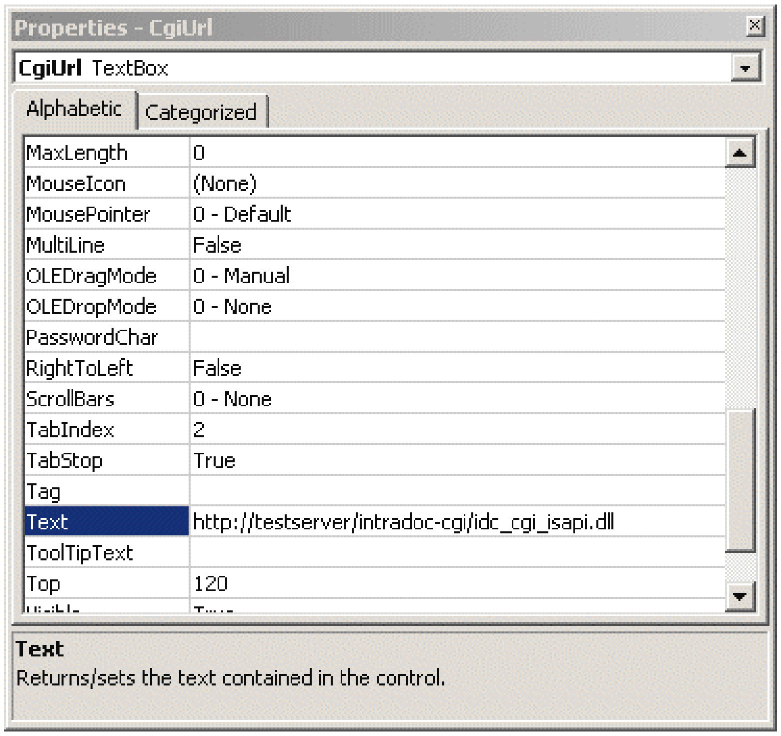
**(Name)**
Returns the name used in code to identify an object.

5. On the form, draw a textbox 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 6–3 Edited CgiUrl TextBox properties*


Properties - CgiUrl

CgiUrl TextBox

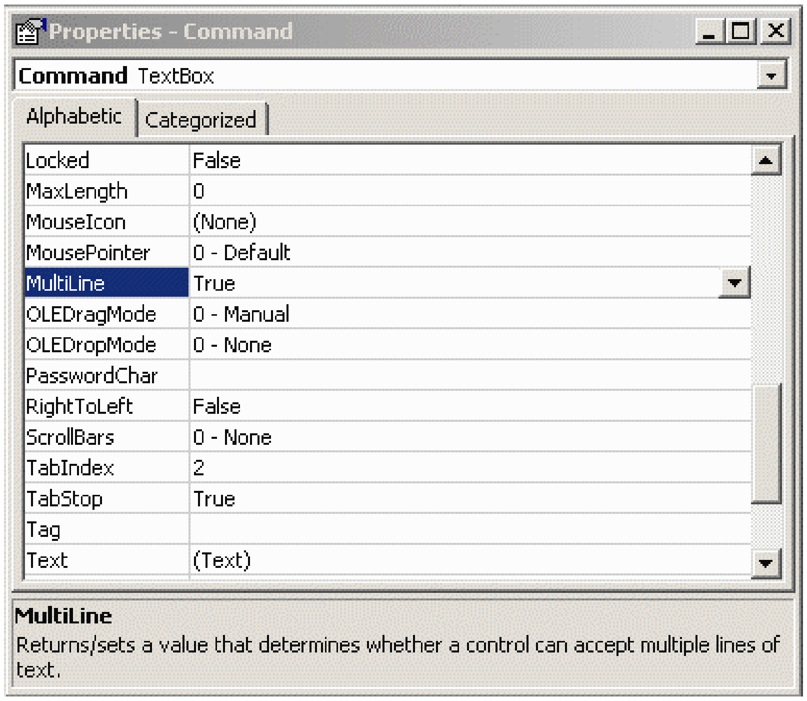
Alphabetic | Categorized

| | |
|---|---|
| MaxLength | 0 |
| MouseIcon | (None) |
| MousePointer | 0 - Default |
| MultiLine | False |
| OLEDragMode | 0 - Manual |
| OLEDropMode | 0 - None |
| PasswordChar | |
| RightToLeft | False |
| ScrollBars | 0 - None |
| TabIndex | 2 |
| TabStop | True |
| Tag | |
| Text | http://testserver/intradoc-cgi/idc_cgi_isapi.dll |
| ToolTipText | |
| Top | 120 |

**Text**
Returns/sets the text contained in the control.

7. On the form, draw a textbox and name it **Command**.
8. Clear the entry for the Text field (leave blank) and set MultiLine to **true**.

*Figure 6–4 Edited Command TextBox properties*


Properties - Command

Command TextBox

Alphabetic | Categorized

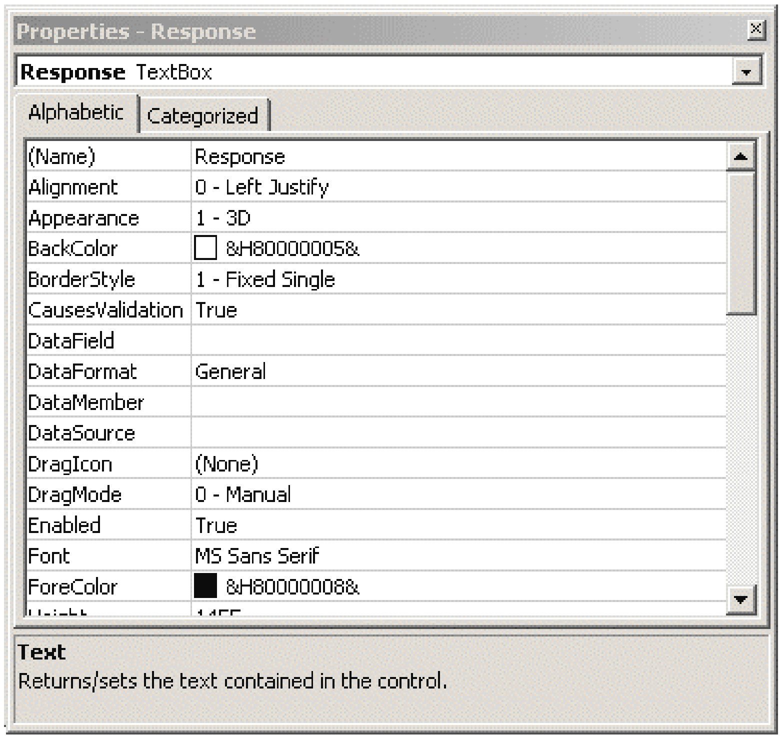
| | |
|---|---|
| Locked | False |
| MaxLength | 0 |
| MouseIcon | (None) |
| MousePointer | 0 - Default |
| MultiLine | True |
| OLEDragMode | 0 - Manual |
| OLEDropMode | 0 - None |
| PasswordChar | |
| RightToLeft | False |
| ScrollBars | 0 - None |
| TabIndex | 2 |
| TabStop | True |
| Tag | |
| Text | (Text) |

**MultiLine**
Returns/sets a value that determines whether a control can accept multiple lines of text.

9. On the form, draw a textbox and name it **Response**.

**10.** Clear the entry for the Text field (leave blank).

***Figure 6–5 Edited Response TextBox properties***



Properties - Response

**Response** TextBox

Alphabetic | Categorized

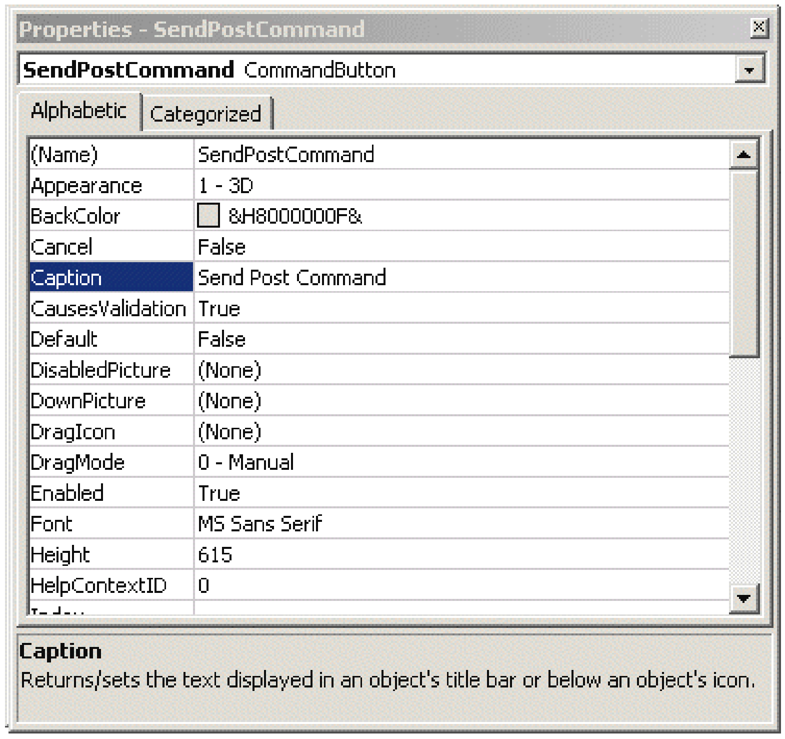
| | |
|---|---|
| (Name) | Response |
| Alignment | 0 - Left Justify |
| Appearance | 1 - 3D |
| BackColor | □ &H80000005& |
| BorderStyle | 1 - Fixed Single |
| CausesValidation | True |
| DataField | |
| DataFormat | General |
| DataMember | |
| DataSource | |
| DragIcon | (None) |
| DragMode | 0 - Manual |
| Enabled | True |
| Font | MS Sans Serif |
| ForeColor | ■ &H80000008& |

**Text**
Returns/sets the text contained in the control.

**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 6–6 Edited SendPostCommand CommandButton properties.***



Properties - SendPostCommand

**SendPostCommand** CommandButton

Alphabetic | Categorized

| | |
|---|---|
| (Name) | SendPostCommand |
| Appearance | 1 - 3D |
| BackColor | ☐ &H8000000F& |
| Cancel | False |
| Caption | Send Post Command |
| CausesValidation | True |
| Default | False |
| DisabledPicture | (None) |
| DownPicture | (None) |
| DragIcon | (None) |
| DragMode | 0 - Manual |
| Enabled | True |
| Font | MS Sans Serif |
| Height | 615 |
| HelpContextID | 0 |

**Caption**
Returns/sets the text displayed in an object's title bar or below an object's icon.

13. On the form, select **View** then select **Code**.

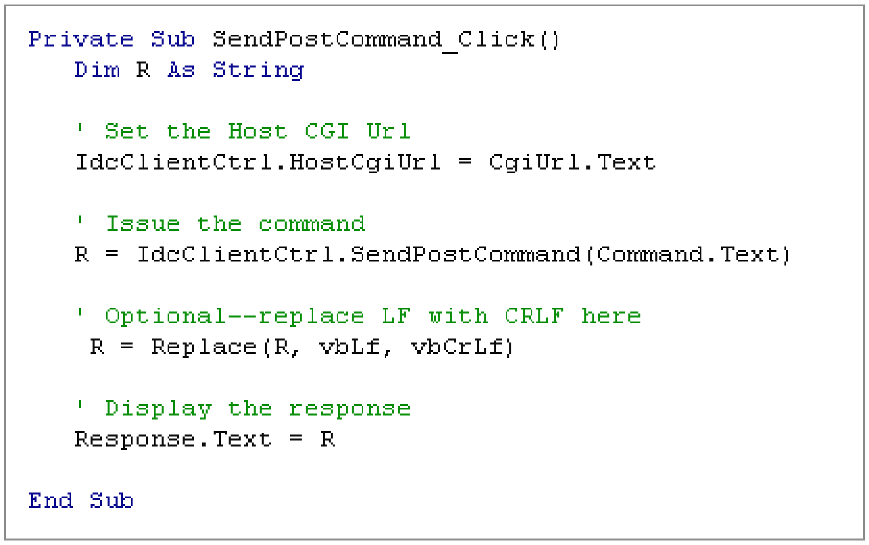
14. Select **SendPostCommand** and then click from the drop-down lists and modify the code to perform these actions.

    - Set the Host Cgi Url
    - 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 6–7 Edited SendPostCommand_Click code***

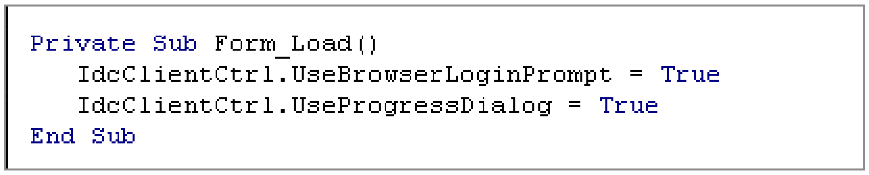


```
Private Sub SendPostCommand_Click()
   Dim R As String

   ' 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. Select **Form** and then **Load** from the drop-down lists and add the following lines to set the login prompt for the Content Server:

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

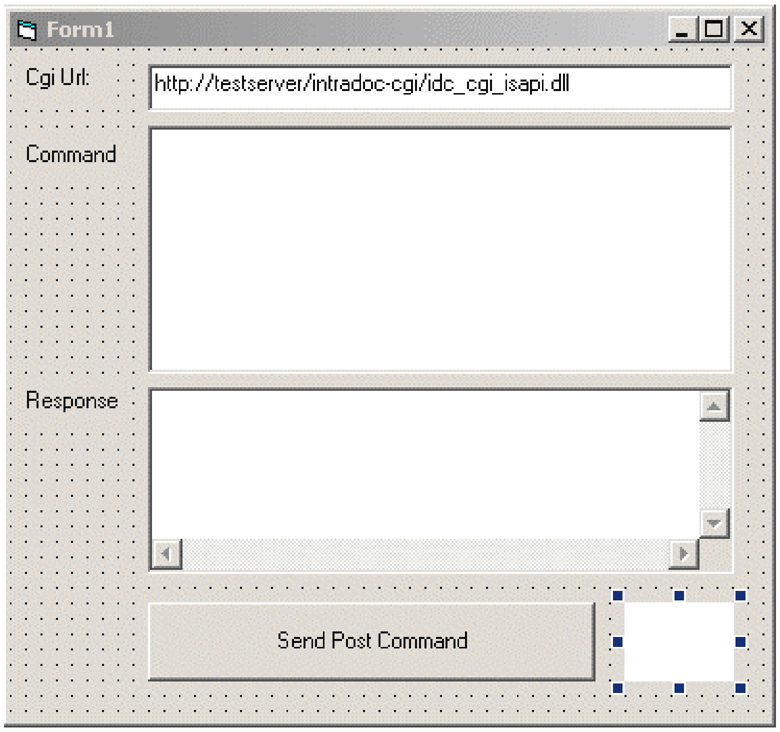
***Figure 6–8 Edited Form_Load code***



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

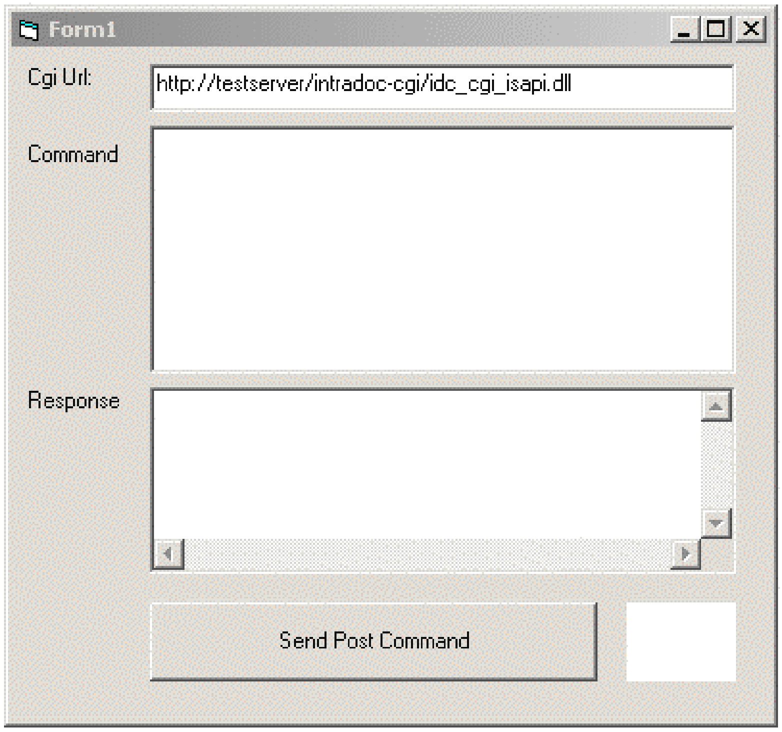
16. (Optional) Add appropriate descriptive labels such as Cgi Url, Command, and Response.

***Figure 6–9 Visual interface with descriptive label***



17. Select **Run** then select **Start** to test the visual interface.

***Figure 6–10 Completed visual interface***



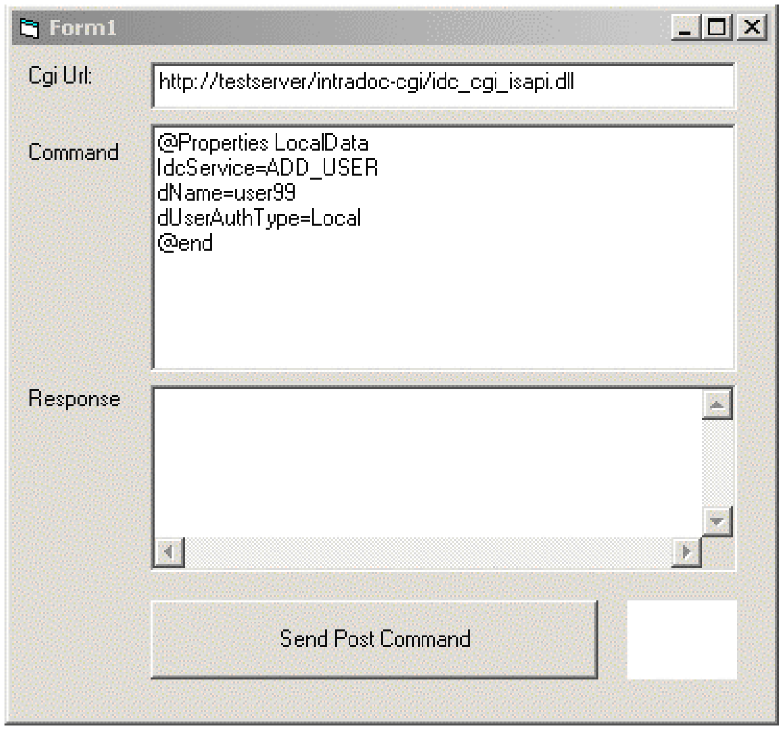
18. Enter a formatted command in the Command field.

    For example, this command adds a user:

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

See the *Oracle Fusion Middleware Services References Guide* for additional information about the ADD_USER service.

***Figure 6–11 Visual interface with defined command.***

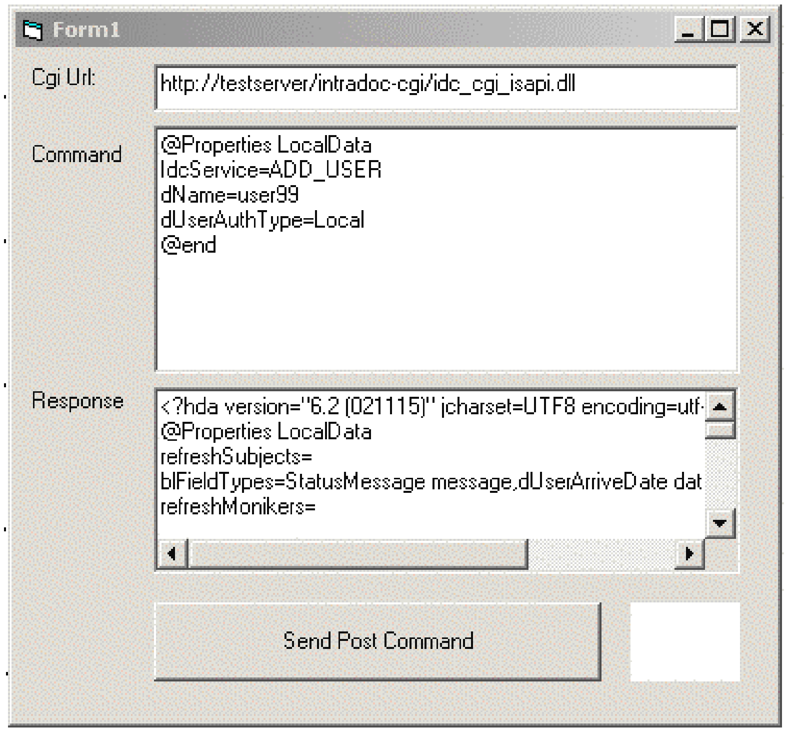


```
http://testserver/intradoc-cgi/idc_cgi_isapi.dll
```

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

**19.** Click the **Send Post Command** button to execute the command. The returned results are displayed in the Response field.

***Figure 6–12 Visual interface with returned results***



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

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

```
<?hda version="6.2 (021115)" jcharset=UTF8 encoding=utf-
@Properties LocalData
refreshSubjects=
blFieldTypes=StatusMessage message,dUserArriveDate dat
refreshMonikers=
```

**Verify the Command**

1. In a Web browser, login to the Content Server as the administrator.
2. Select **Administration** then select **Admin Applets**.
3. Click the **User Admin** link. The applet launches and displays the added user (for example, *user99*).

## 6.3.5 IdcClient Events

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

- "IIntradocBeforeDownload" on page 6-44
- "IIntradocBrowserPost" on page 6-45
- "IntradocBrowserStateChange" on page 6-45
- "IIntradocRequestProgress" on page 6-45
- "IntradocServerResponse" on page 6-45

### 6.3.5.1 IIntradocBeforeDownload

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

#### 6.3.5.2 IIntradocBrowserPost

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

#### 6.3.5.3 IntradocBrowserStateChange

Executes whenever the browser state changes.

**Parameters**

The event passes these parameters:

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

#### 6.3.5.4 IIntradocRequestProgress

Executes a request for a progress report to be sent from the server. This event only occurs after a method has been called.

**Parameters**

The event passes these parameters:

- ByVal `statusData` As String
- ByVal `isDone` As Boolean

#### 6.3.5.5 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

### 6.3.6 IdcClient Methods

The following IdcClient OCX Methods are available:

- "AboutBox" on page 6-46
- "Back" on page 6-47
- "CancelRequest" on page 6-47*
- "DoCheckoutLatestRev" on page 6-47
- "DownloadFile" on page 6-48
- "DownloadNativeFile" on page 6-48

- "Drag" on page 6-49
- "EditDocInfoLatestRev" on page 6-49
- "Forward" on page 6-49
- "GoCheckinPage" on page 6-50
- "Home" on page 6-50
- "InitiateFileDownload" on page 6-51*
- "InitiatePostCommand" on page 6-51*
- "Move" on page 6-52
- "Navigate" on page 6-52
- "NavigateCgiPage" on page 6-52
- "Refresh Browser" on page 6-52
- "SendCommand" on page 6-53*
- "SendPostCommand" on page 6-53*
- "SetFocus" on page 6-53
- "Show DMS" on page 6-54
- "ShowDocInfoLatestRev" on page 6-54
- "ShowWhatsThis" on page 6-54
- "StartSearch" on page 6-54
- "Stop" on page 6-55
- "UndoCheckout" on page 6-55
- "ViewDocInfo" on page 6-55
- "ViewDocInfoLatestRev" on page 6-56
- "ZOrder" on page 6-56

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.

### 6.3.6.1 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

### 6.3.6.2 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

### 6.3.6.3 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.

### 6.3.6.4 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. See "IdcClient Events" on page 6-44 for details.

> **Note:** The `curID` is the content item version label, not the generated content item revision ID.

This function returns the following:

- 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.

#### 6.3.6.5 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 post-conversion file (compare `DownloadNativeFile`).
- Executes the `IntradocBeforeDownload` event. The event is executed before the method occurs. See "IdcClient Events" on page 6-44 for details.

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.

#### 6.3.6.6 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. See "IdcClient Events" on page 6-44 for details.

---

**Note:** The `id` 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.

#### 6.3.6.7 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.

#### 6.3.6.8 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` 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*.

#### 6.3.6.9 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

### 6.3.6.10 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` 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 pre-fill the Check In page.

### 6.3.6.11 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

#### 6.3.6.12 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.
- See "IdcClient Events" on page 6-44 for details.

#### Parameters

- command: The currently-associated command.
- filename: The file format. This is the file type such as PDF, HTM, or other supported format.

#### Output

- Returns serialized HDA containing the requested information.
- Returns the data that was passed in as parameters.

#### 6.3.6.13 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. See "IdcClient Events" on page 6-44 for details.

#### 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.

#### 6.3.6.14 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.

#### 6.3.6.15 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.

#### 6.3.6.16 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.

#### 6.3.6.17 Refresh Browser

**Description**

Refreshes the browser.

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

#### Parameters
None

### 6.3.6.18 SendCommand

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

#### Description
Issues a service request to the Content Server.

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

### 6.3.6.19 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. See "IdcClient Events" on page 6-44 for details.

#### 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.

### 6.3.6.20 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

### 6.3.6.21 Show DMS

```
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

### 6.3.6.22 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` 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.*

### 6.3.6.23 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.

### 6.3.6.24 StartSearch

```
Sub StartSearch()
```

**Description**

Displays the query page in the browser control.

- Preforms browser manipulation.

**Parameters**

None

### 6.3.6.25 Stop

```
Sub Stop()
```

**Description**

Stops the browser.

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

**Parameters**

None

### 6.3.6.26 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.
- See "IdcClient Events" on page 6-44 for details.

**Note:** The `curID` 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.*

### 6.3.6.27 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` is the generated content item revision ID, not the content item version label.

**Parameters**

- id: The unique identifier for the latest revision.

#### 6.3.6.28 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` 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.

#### 6.3.6.29 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.

### 6.3.7 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" on page 6-57
- "HostCgiUrl" on page 6-57
- "Password" on page 6-57
- "UseBrowserLoginPrompt" on page 6-57
- "UseProgressDialog" on page 6-57
- "UserName" on page 6-58
- "Working Directory" on page 6-58

#### 6.3.7.1 ClientControlledContextValue

Provides the user-supplied context value. This value becomes available to Idoc Script as the variable `ClientControlled` in any Web page delivered by the Content Server.

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

#### 6.3.7.2 HostCgiUrl

Provides the complete URL path of the host CGI bin.

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

#### 6.3.7.3 Password

Provides the assigned user password.

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

#### 6.3.7.4 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
- Default is TRUE.

Returns a Boolean value:

- Returns TRUE if the login was successful.
- Returns FALSE if the login was denied.

#### 6.3.7.5 UseProgressDialog

Allows 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.

#### 6.3.7.6 UserName

Provides the assigned user name.

Returns the value as a string.

Takes no parameters.

#### 6.3.7.7 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.

### 6.3.8 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 Content Server is available with Desktop, a separate product. Use the ODMA-integration products to gain access to the content and content management functions within Content Server (for ODMA-compliant desktop applications).

You can publish files to your Web repository directly from any ODMA-compliant application, such as Microsoft Word, Corel WordPerfect, and Adobe FrameMaker. With the Web centric adoption of ODMA, you can check in and publish information directly to the Web. This is a significant advancement over traditional ODMA client/server implementations, where information is published first to a server and is not immediately available on the Web for consumption.

For more information, refer to the ODMA or ODMA/FrameMaker online help.

#### 6.3.8.1 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 Content Server, for example:

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

#### 6.3.8.2 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 the Content Server or the local file system.
- **ODMA Desktop Shell Interface**: The Client Desktop Shell provides "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 the Content Server.
  - Select and open a file from the Recent Files list or from the Content Server.
- **Content Server Interface with ODMA**: You can open and check out an ODMA file directly from the Content Server Content Information page. When you open a file from the Content Server, it opens in its native application so you can edit it and quickly check the file back into the 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. See the ODMA Online Help for more information.

## 6.4 RIDC Integration

Remote Intradoc Client (RIDC) provides a thin communication API for communication with Content Server. This API removes data abstractions to the 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.

Key features of RIDC include:

- Support is provided for Intradoc socket-based communication and the HTTP protocol.
- Client configuration parameters include setting the socket timeouts, 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 the content server and the client. For HTTP URLs, the context requires credentials.
- To invoke a service, 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 the content server via 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 Content Server.

For details, see *Oracle Fusion Middleware Developer's Guide for Remote Intradoc Client for Oracle Enterprise Content Management Suite*.

## 6.5 JSP Integration

You can access Content Server core functionality from a Java Server Page 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 "Java 2 Enterprise Edition Integration (J2EE)" on page 6-62.

This section covers the following topics:

- "JSP Execution" on page 6-60
- "Tomcat" on page 6-60
- "Features" on page 6-61
- "Configuring JSP Support" on page 6-61

### 6.5.1 JSP Execution

The JSP Execution functionality uses the built-in Apache Jakarta Tomcat Servlet/JSP Server to access the content and content management functions within Content Server.

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

Using JSP Execution functionality enables developers to access and modify Content Server content, ResultsSets, personalization and security definitions, and predefined variables and configuration settings through Java Server 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 the Content Server.

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

### 6.5.2 Tomcat

Capability for JSP to call services is provided by integrating the Tomcat 5.025 server with Content Server core functionality.

- Tomcat is a free, open-source server of Java Server and Java Server Pages; version 5.025 complies with Servlet 2.4 and JSP 2.0 specifications.

- The main benefit of integrating Tomcat into Content Server is the increase in performance of delivering content. The direct integration eliminates the need for a socket-based interface and enables use of all Content Server core capabilities.
- Although Tomcat is embedded in content server, you can use server.xml as the configuration file to modify the internal Tomcat engine to suit your needs.

**Note:** This product includes software developed by the Apache Software Foundation (`http://www.apache.org/`).

## 6.5.3 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 Content Server distribution media also includes the current Java 2 SDK.

## 6.5.4 Configuring JSP Support

Use the following procedure to enable and configure JSP support.

1. In Content Server, create a new security group to be used for JSP pages (called `jsp` in the subsequent steps). This security group should be restricted to developers. This step is not required but it is recommended for developer convenience. Any security groups to be enabled for JSP must be specified in step 5.
   a. Display the User Admin screen.
   b. Select **Security**, **Permissions by Group**.
   c. Click **Add Group.**
   d. Enter `jsp` as the group name, enter a description, and 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. To get the internal JSP engine to run on these, a 1.5 JDK must be present on the server and the CLASSPATH 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. Select one of the following:
   - From the Admin Server, select the General Configuration page.
   - From the System Properties utility, select 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 the Content Server.

### 6.5.5 Loading Example Pages

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

## 6.6 Java 2 Enterprise Edition Integration (J2EE)

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

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

This section covers these topics

- "Content Integration Suite Architecture" on page 6-62
- "Accessing the UCPM API" on page 6-63
- "UCPM API Methodology" on page 6-63

See the Content Integration Suite documentation set for additional information.

### 6.6.1 Content Integration Suite 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 API (UCPM API) which uses the SCS API for communication to the Content Server. The SCS API wraps communication from the 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 Content Server services (IdcCommand services). The UCPM API comprises a set of command objects that encapsulate distinct actions that are passed to the UCPM API and then mapped to the Content Server. These commands include common functions such as search, checkout, 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 will leverage the features in the environment, whether this is a J2EE application server or non-J2EE.

The UCPM API encapsulates Content Server business logic and validates the parameters of the incoming calls. The UCPM API handles communication to the Content Server, encapsulates socket communication logic (opening, validating, and streaming bits through the socket), and provides a strongly typed API to the available services.

### 6.6.2 Accessing the UCPM API

The Universal Content and Process Management API (UCPM API) offers access to servers by exposing their services and data in a unified object model. The UCPM API is modeled into a set of Services APIs, 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 getUCPMAPI() method. The getUCPMAPI() method returns a reference to the IUCPMAPI object, allowing access to all UCPM API objects. Public interface IUCPMAPI 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 the Content Server.

### 6.6.3 UCPM API Methodology

The Universal Content and Process Management API (UCPM API) is stateless; all method calls pass in the necessary state to the method. You can share the reference to the CISApplication class across threads.

- ISCSContext for the SCS API. The interface ISCSContext is the context object used when communicating with the Content Server.
- ICISCommonContext for calling some of the CIS APIs. The interface ICISCommonContext 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 username 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.

## 6.7 Web Services

This section provides an overview of Web services, and general information on the SOAP protocol. In addition, several basic implementation architectures are described.

This section contains these topics:

- "Web Services Framework" on page 6-63
- "Virtual Folders and WebDAV Integration" on page 6-64

### 6.7.1 Web Services Framework

Web services reside as a layer on top of existing software systems such as application servers, .NET servers, and the 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 WebLogic Server provides SOAP capabilities, and Content Server supports several SOAP requests through Oracle WebLogic Server. For more information, see Chapter 7, "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) Content Servers 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 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 **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).

### 6.7.2 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 the Content Server in the form of virtual folders that enables 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 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. See the *Oracle Fusion Middleware Application Administrator's Guide for Content Server* for additional information.

#### 6.7.2.1 Virtual Folders

The Folders component sets up an interface to the 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 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 Content Server 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.

#### 6.7.2.2 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.

See the WebDAV Resources Page at `http://www.webdav.org` for more information

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 Content Server, the WebDAV client serves as an alternate user interface to the native files in the content repository. The same versioning and security controls apply, whether an author uses the Content Server Web browser interface or a WebDAV client.

In Content Server, the WebDAV interface is based on the hierarchical Folders interface. See "Virtual Folders" on page 6-65 for additional information.

**6.7.2.2.1 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 specific versions supported. This is not the same as the Content Server WebDAV Client, which is a product that enhances the WebDAV interface to the 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 and out of the Content Server repository.

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

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

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

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

# 7 Using Oracle UCM Web Services

This chapter describes using Oracle Universal Content Management (Oracle UCM) Web services with Oracle WebLogic Server Web services to manage Content Server. It covers these topics:

## 7.1 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 the 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 Universal Content Management (UCM) Web services work with Oracle WebLogic Server Web services to perform management functions for 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 WSM and WS-SecurityPolicy policies can work with each other.

**Note:** It is recommended that you use OWSM policies over Oracle WebLogic Web services whenever possible. You cannot mix your use of OWSM and Oracle WebLogic Web service policies on the same Web service.

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 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 Oracle 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*.

## 7.2 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 are:

- `/idcws/` - Use this context root for general access to a Oracle UCM 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 the Oracle UCM Content Server when you set IsSoap=1. Details of the format can be found in the published WSDL at /idcws/GenericSoapPort?WSDL.<br><br>You can apply WS-Security to GenericSoapService through WS-Policy. The Content Server supports Oracle Web Services Manager (OWSM) policies for Security Assertion Markup Language (SAML) and username-token.<br><br>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 the Oracle UCM Content Server, wrapped in a SOAP request. Requests are sent to the Content Server using streaming Message Transmission Optimization Mechanism (MTOM) in order to support large files.<br><br>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.<br><br>If you want to use WS-Security with IdcWebRequestService, then apply it here. The Content Server supports Oracle Web Services Manager (OWSM) policies for Security Assertion Markup Language (SAML) and username-token. |

## 7.3 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 the Oracle UCM content server will use the “anonymous” user. Additional configuration is required to enable authentication.

## 7.4 Security

The following topics covers security configuration for Oracle UCM Web services.

- "Configuring WS-Security through WS-Policy" on page 7-4
- "Configuring SAML Support" on page 7-4

### 7.4.1 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. Click **IdcWebLogicService** or **GenericSoapService**, then select the **Configuration** tab, then 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.

### 7.4.2 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.

#### 7.4.2.1 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 to generate a self-signed certificate. Note: many of the values used in the following example are the defaults for the domain's config/fmwconfig/jps-config.xml (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 within the keytool, but the specifics do not matter except for the password for the keystore and the certificate, which the client uses.

#### 7.4.2.2 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 the <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>
```

#### 7.4.2.3 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.

In order to use WLST to create a credential, you must be connected to the Oracle WebLogic Server domain. Note that the resulting wallet can only be used 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.

#### 7.4.2.4 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>
```

# Index