Getting Started with the Software Developer's Kit (SDK)
10*g* Release 3 (10.1.3.3.1)

May 2007

ORACLE

Getting Started with the Software Developer's Kit (SDK), 10*g* Release 3 (10.1.3.3.1)

Contributing Authors: Jean Wilson

Table of Contents

Chapter 4: Altering the Interface

Chapter 5: Changing System Functionality

Chapter 6: Integration Methods

Appendix A: Third Party Licenses

Index

Chapter 1

ABOUT THIS GUIDE

OVERVIEW

While Content Server is highly functional "out-of-the-box," there are a number of ways to tailor your content management system to your site requirements. The Software Developer's Kit (SDK) provides a suite of tools that can be used for system customization.

This guide provides the background information you will need before changing your content server, and it gives pointers to other documents that provide the details.

This section covers these topics:

AUDIENCE

This guide is intended for developers and administrators who need to customize Content Server software to suit content management needs that are specific to a particular business or organization.

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 will vary depending on the complexity of the customization. For example, many customizations 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. See Chapter 3 (*Content Server Architecture)* for more information.
- **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.htm file includes 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. See Changing Configuration Information (page 5-9) for an overview of Idoc functionality.
- **JavaScript—**Most content server pages do not use JavaScript, but the Search and Checkin pages are notable exceptions. You should have an understanding of JavaScript before you create customizations that will be called in place of these pages. Also, you should 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 will 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 customizations 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 will 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 customizations 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 will 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 customizations require the development of Java code, you will need an appropriate Java development environment.

RELATED DOCUMENTATION

The Content Server online help provides user, administrator, and developer-level documentation, which describes how the standard product works and how functions can be modified from within the content server. You should become familiar with the online help and consult it for background knowledge before attempting to customize a particular area of the content server.

A documentation navigation menu is provided as a starting point for access to all PDF documentation. Installation guides, integrator and developer documentation, and all online help topics are provided in PDF format and are accessible from the documentation navigation menu.

You can access the product documentation navigation menu from the following locations:

- **Product CD:** From the Content Server Documentation CD, open the menu.pdf file.

- **Windows:** If Content Server is installed on a Windows computer, select **Start—Programs—Content Server—Utilities—Documentation**.
- **UNIX:** If Content Server is installed on a UNIX computer, open the <*install_dir*>/weblayout/help/documentation/menu.pdf file.

The following guides are referenced in this documentation and provide the details for your customization work:

- *Working with Components*, which describes how to create custom components for your content server instance.
- The *Managing Repository Content Guide*, the *Managing Security and User Access Guide,* and the *Managing System Settings and Processes Guide.* These books explain many basic administrative tasks as well as many customization options.
- The *IDOC Script Reference Guide*, which describes configuration variables and their uses.
- The *Services Reference Guide*, which describes services and their attributes and actions.
- The *Workflow Implementation Guide*, which describes how to create and use workflows.
- The *IdcCommand Reference Guide*, which describes the Idc command utility.
- *WSDL Generator and SOAP Guide*, which explains how to use WSDLs and SOAP for connectivity.
- The *Clustering Concepts Guide*, which provides an overview of clusters and their interaction of the content server.
- *Modifying the Content Server Interface*, which explains how to change the layout and skins that come with the Content Server.

DOCUMENT ORGANIZATION

This guide includes the following sections:

- Chapter 1 (*About This Guide)* outlines the audience, organization, and conventions for this guide, and describes content management product distinctions and support options.
- Chapter 2 (*Modifying Your Instance)* provides an introduction to the methods you can use to customize content server.

- Chapter 3 (*Content Server Architecture)* describes the architecture of the content server and how that affects the customizations you can make.
- Chapter 4 (*Altering the Interface)* defines the items you can adjust to change the appearance of the content server, including the portal web page.
- Chapter 5 (*Changing System Functionality)* describes how you can change the functionality of the content server with system settings, components, and configuration variables.
- Chapter 6 (*Integration Methods)* provides information about integrating the content server in other environments using tools such as SOAP and clusters.

WHAT'S NEW IN THIS GUIDE

Information has been added to Chapter 5 (*Changing System Functionality)* about enhancements in Content Server 10*g*R3 which allow you to publish web resource files as well as configure and customize Action menus.

DOCUMENT CONVENTIONS

The following conventions are used throughout this guide:

- The notation ***<install_dir>/*** is used throughout this guide to refer to the location on your system where the Content Server product is installed.
- Forward slashes (/) are used to separate the directory levels in a path name. A forward slash will always appear after the end of a directory name.
- Notes, technical tips, important notices, and cautions use these conventions:

Symbol	Description
	This is a note. It brings special attention to information.
	This is a tech tip. It identifies information that can be used to make your tasks easier.

Symbol	Description
	This is an important notice. It identifies a required step or critical information.
	This is a caution. It identifies information that might cause loss of data or serious system problems.

Chapter

2

MODIFYING YOUR INSTANCE

OVERVIEW

This chapter provides an overview of content server customizations and describes the tools you need and the resources that are available. It includes the following sections:

CUSTOMIZATION TYPES

Before undertaking any type of alteration to your content server instance, you should completely understand the structure and architecture of the content server and its interaction with other parts of your environment. See Chapter 3 (*Content Server Architecture)* for details about Content Server functionality and usage.

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. See Chapter 4 (*Altering the Interface)* for details.

- **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. See Chapter 5 (*Changing System Functionality)* for details.
- **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. See Chapter 6 (*Integration Methods)* for more details.

CUSTOMIZATION TIPS

Before approaching customization, it is important to clarify exactly *why* the customization is being undertaken. For example, if you need to add corporate branding, you can use the Modify Layout Samples to do so. Or if you need 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 a company. 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 will most please 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 customizations should be done in a test environment, separate from the site's production environment.

3. **Check to see if a solution may already be available**. The *Samples* on the Support web site contain many types of customizations. It's possible that there may already be a 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.

- ‘Extras’ are supported components that add functionality to the base Content Server, such as virtual folders, WebDAV functionality, HTML preview, threaded discussions and lightly managed content functionality. These are available on the product CD.

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 isn’t 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 customizations 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 customizations later.

TROUBLESHOOTING

The *Content Server Troubleshooting Guide* provides a problem-solving model for helping you determine where errors might be occurring.

In addition, 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 (page 2-3)

❖ Viewing Page Data (page 2-4)

❖ Monitoring Resource Loading (page 2-5)

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 are running 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 <*install_dir*>/weblayout/groups/secure/logs/ directory.

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 *ScriptDebugTrace* 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:

  ```
  &ScriptDebugTrace=1
  ```

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

  ```
  <$ScriptDebugTrace=1$>
  ```

- 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$>)$>
  ```

The *ScriptErrorTrace* setting enables a script error trace for a content server page request. Script error information is added to the bottom of the displayed page. If no errors are detected, the message "No Errors" is displayed.

- In a web browser, add the following code to the end of the page's URL:

  ```
  &ScriptErrorTrace=1
  ```

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

  ```
  <$ScriptErrorTrace=1$>
  ```

`IsJava`, `ScriptDebugTrace`, and `ScriptErrorTrace` are discussed in detail in the *Idoc Script Reference Guide*.

Monitoring Resource Loading

There are three configuration settings that 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 <*install_dir*>/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 *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.htm
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]
…
```

Chapter

3

CONTENT SERVER ARCHITECTURE

OVERVIEW

To create customizations efficiently and effectively, you should have a good understanding of how the content server works. This chapter describes the architecture of the 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

When you create custom components or dynamic server pages, you work primarily with files in the following directories:

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

/bin Directory

The <*install_dir*>/bin directory contains the executable files that run the content server services, applets, and utilities:

Element	Description
Executables	**Services** • IdcServer • IdcServerNT **Applets** • IntradocApp (launches all Admin tools) **Utilities** • Batch Loader • Installer • IdcAnalyze • Component Wizard • System Properties • IdcCommand
intradoc.cfg file	Configuration file that contains the settings for 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. A number 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:/stellent/idcm1/
WebBrowserPath=C:/Program Files/Internet Explorer/iexplore.exe
CLASSPATH=$COMPUTEDCLASSPATH;$SHAREDDIR/classes/jtds.jar

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

/config Directory

The <*install_dir*>/config directory stores global configuration information, including information about custom components:

Element	Description
config.cfg file	Defines system configuration variables.
components.hda	Identifies custom components that have been added to the content server system.

The config.cfg file is used to define global variables for the content server system. A number 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/stellent1;charset=UTF8;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=/stellent/
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
```

/shared/config Directory

The <*install_dir*>/shared/config directory contains the files that the content server uses to assemble web pages:

Element	Description
reports	Holds templates for content server reports.
resources	Holds resource definitions (queries, page resources, services, and other resource data) for the content server.
resources/lang	Holds localized string definitions for the content server.
templates	Holds templates for all content server pages (except reports).

/weblayout Directory

The <*install_dir*>/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
common	Holds applet files and other application files that are available to the master instance and its proxies.
groups	Holds the web-viewable content items and dynamic server pages.
help	Holds the online help files.
images	Holds images, such as icons and home page graphics.
resources	Holds layouts, skins, and schema information.

RESOURCES

Resources are files that define and implement the actual customizations 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 carry out content server actions, or special code to conditionally format information.

Resources are a critical part of the content server software, so you need to be familiar with them before you attempt to create a custom component or dynamic server page. There are two ways to create and edit a resource file:

- Edit the files manually, by opening the resource file in a text editor and editing the code.
- Use the Component Wizard, where you can add, edit, or remove a resource file. You can also use the Component Wizard as a starting point for creating custom resources.

Resources fall into eight distinct categories, although the first four types listed below 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 more than one content server web page.	<*install_dir*>/shared/config/resources/std_page.htm
String	Defines localized strings for the user interface and error messages.	<*install_dir*>/shared/config/resources/lang/cs_strings.htm
Dynamic Table (HDA format)	Provides dynamic (frequently changed) content in table format to the content server.	<*install_dir*>/config/active_state.hda
Static Table (HTML format)	Provides static (seldom changed) content in table format to the content server.	<*install_dir*>/shared/config/resources/std_locale.htm
Query	Defines database queries.	<*install_dir*>/shared/config/resources/query.htm
Service	Defines scripts for services that can be performed by the content server.	<*install_dir*>/shared/config/resources/std_services.htm

Resource Type	Description	Example of Standard Resource
Template	Defines templates, which contain the code that the content server uses to assemble a particular web page.	<*install_dir*>/shared/config/templates/checkin_new.htm
Environment	Defines configuration settings for the content server.	<*install_dir*>/config/config.cfg

CONTENT SERVER BEHAVIOR

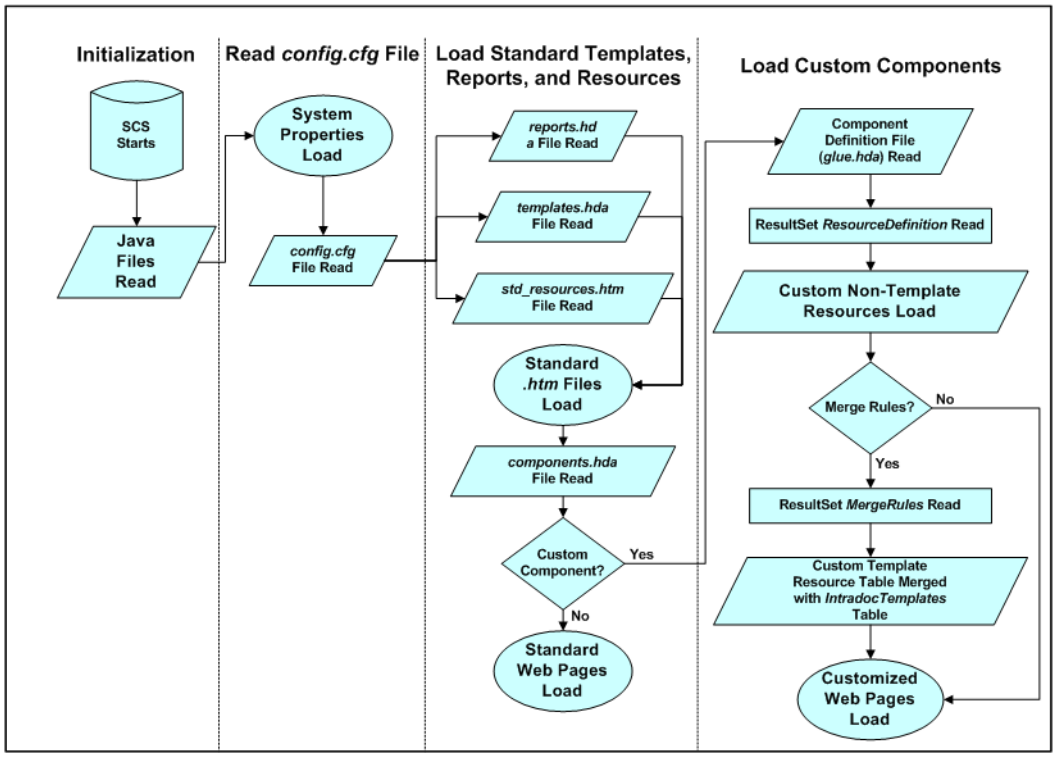
This section describes how the content server behaves in the following situations:

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).

Figure 3-1 Content server startup behavior



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 <*install_dir*>/bin/intradoc.cfg file initialize as well.
2. **Configuration Variables Load:** After initializing, the content server loads the *config.cfg* file from the <*install_dir*>/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 Templates, Resources, and Reports Load**: For the content server to function properly, a number of standard templates, resources, and reports need to be loaded. After the configuration settings have been loaded, the content server reads the entries in the <install_dir>/shared/config/templates/templates.hda file and the <install_dir>/shared/config/reports/reports.hda file. These files tell the content server which templates to load, which in turn loads any standard resources referenced in the template and report pages.
4. **Custom Components Load**: The content server loads custom components in the order specified in the <*install_dir*>/config/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.

Effects of Configuration Loading

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

The <*install_dir*>/config/config.cfg file is loaded first, and the <*install_dir*>/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. <*install_dir*>/config/config.cfg.
2. <*install_dir*>/custom/<*component_name*>/*_environment.cfg. Some components may not have this file or it may be named environment.cfg.
3. <*install_dir*>/data/components/<*component_name*>/install.cfg.
4. <*install_dir*>/data/components/<*component_name*>/config.cfg.

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

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 doesn't appear 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.

Resource Caching

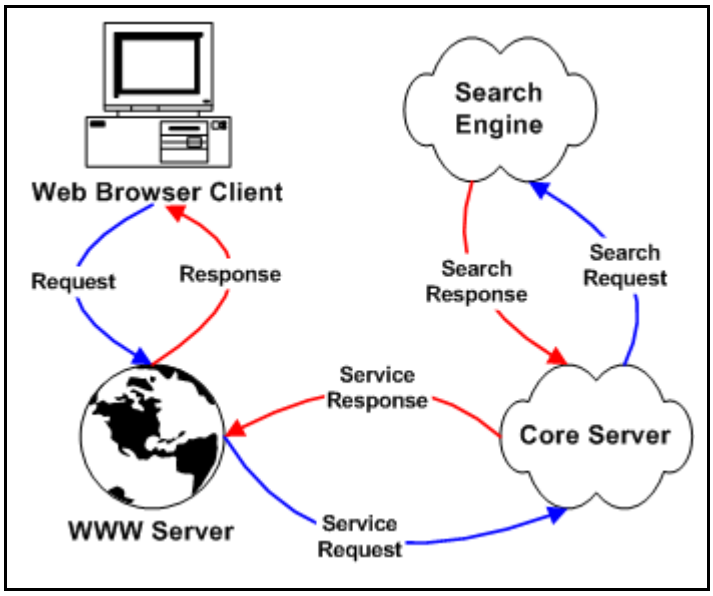
- When the content server loads template pages and resources, they are cached in memory to accelerate page presentation.
- If you change a template page, report page, or HTML include resource, or you check in a revision to a dynamic server page, your changes go into effect immediately—the next request for the associated web page or refresh of the page reflects the changes and the new information is cached. This is because pages are assembled dynamically for each page request.
- If you change any other component files (including services, queries, environment variables, strings, 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 crash if they were implemented immediately. For more information see *Working with Components.*

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 (page 3-12).
- Runs a content server service—See Content Server Services (page 3-13).
- Runs a search engine service—See Search Services (page 3-13).

Figure 3-2 Requests and responses between content management system elements.



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.name/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.
- **`idcplg`** is the path to the web server filter.
- **`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>
```

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 (<*install_dir*>/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.

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.

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.

Page Assembly

The content server uses information from the files in the <*install_dir*>/shared/config 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 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 already 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.

Database Interaction

Some databases, such as Oracle, 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 <*install_dir*>/shared/config/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 *Working with Components* for information about using the upper_clmns_map.htm file.

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.

Note: See the *Content Server International Considerations Guide* for more information on system locales, string localization and user locales.

Chapter

4

ALTERING THE INTERFACE

OVERVIEW

There are several different methods that can be used to alter the appearance of your content server installation.

This chapter provides an introduction to ways to alter the interface that is presented to users of the content server. It covers the following topics:

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 the *Managing Repository Content Guide* for details about creating and modifying metadata fields and creating content profiles.

CHANGING THE LOOK OF THE CONTENT SERVER

The User Personalization settings available on the My Profile page allow users to change the *layouts* of the Content Server or the *skin.*

Layouts define the navigation hierarchy display. There are three types of navigation hierarchies:

- **trays**, in which the high-level navigation links appear on the left side of the screen.
- **top menus**, in which the high-level links appear above the main display area.

- **classic**, in which a series of icons are displayed on the left side of the screen, representing the high-level navigation links

Skins define the color scheme and other aspects of appearance of the layout such as graphics, fonts, or font size. There are two different default skins that are provided:

- **Oracle**: Oracle branding and colors are applied.
- **Oracle2**: the default skin for Content Server 7.5 and later.

You can modify the graphics used, the fonts, colors, and the navigational layout of the display to match what is needed at your site. 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.

Custom layouts change behavior and the look-and-feel system-wide. If you want your changes to apply only in limited situations, you might want to consider dynamic server pages.

See *Modifying the Content Server Interface* for detailed examples of editing the appearance of the interface and publishing the changes.

GENERATING 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 <*install_dir*>/shared/config/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.

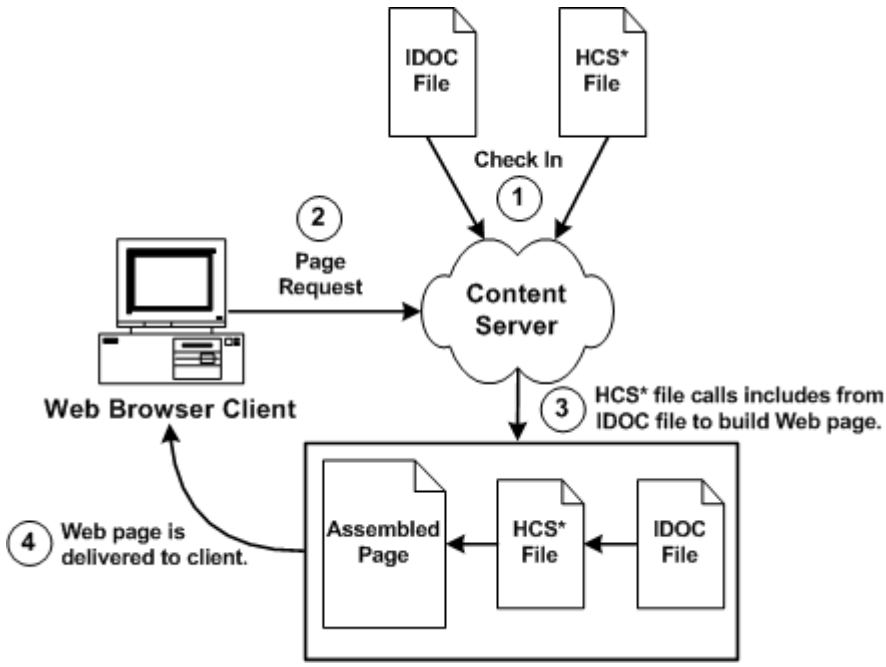
Site Studio, a separately purchased product, can also be used to create HCSP pages, changing the default view of Content Server. See the *Site Studio* documentation set for details.

When you use dynamic server pages, the content server assembles web pages dynamically using a custom template (HCST, HCSP, or HCSF file) that you checked in to the content server. The template calls HTML include code from a text file (IDOC file) you 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 and/or the IDOC file, and then check in the revised files as new revisions. Your changes are available immediately.

The following illustration shows the dynamic server page process.

Figure 4-3 Dynamic server page process



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

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.

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.

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 don't 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 don't 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.

For more details, see the *Dynamic Server Pages Guide* and the *Working with Components* guide.

Chapter

5

CHANGING SYSTEM FUNCTIONALITY

OVERVIEW

Several different methods exist that can be used to change the basic functionality of the Content Server. This chapter provides an overview of those methods. It includes these topics:

CHANGING SYSTEM SETTINGS

Content Server has a number of features that you can set up to change features system-wide 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 system-wide settings for multiple content server instances. To access these pages, click **Admin Server** from the Administration menu to display the Admin Server main page. From this page you can check the status of the different servers that are running, you can check console output, and you add or edit servers. By clicking on a server name, you can access more options for that server.
- **System Properties**: System Properties is an administration application that is used to configure system-wide settings for content security, internet settings, localization, and other types of settings. Options on these pages can be used to:
  - 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

  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, and you can filter files based on security groups or queries.
- **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.

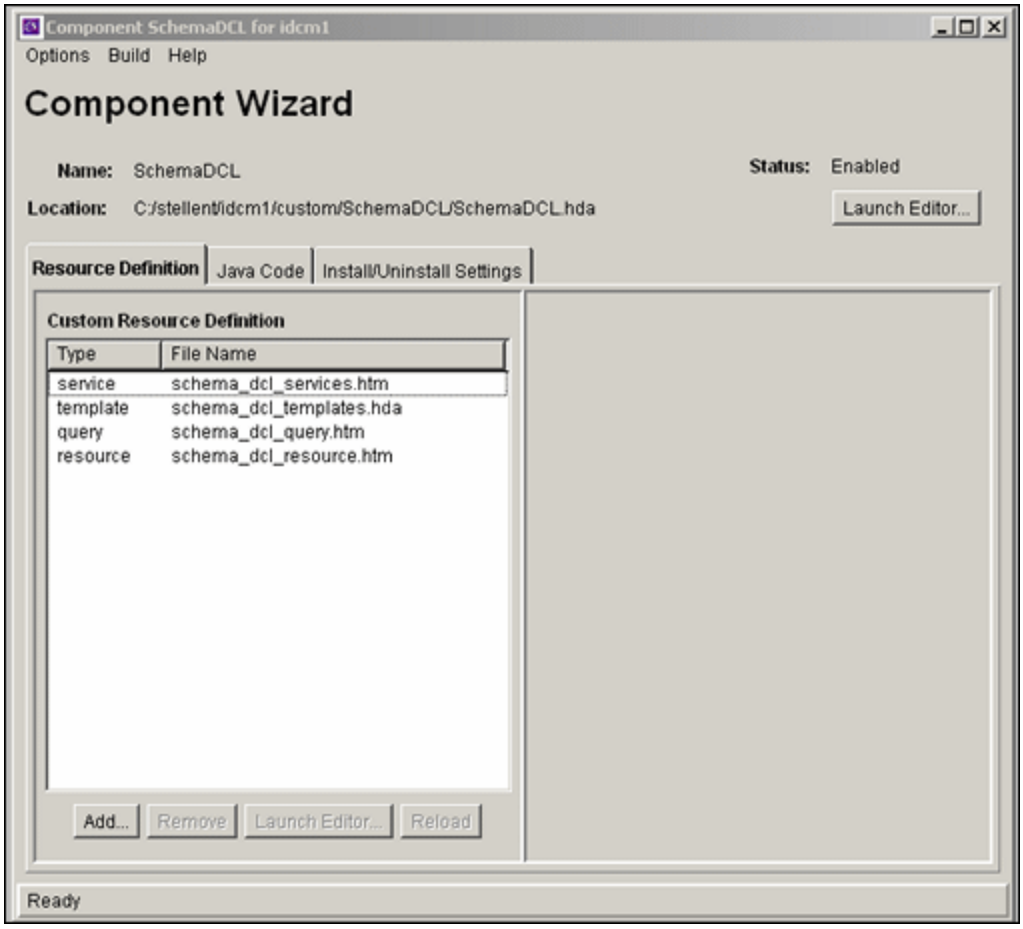
- new custom metadata fields can be created and default values set using the Configuration Manager.
- customized ‘action’ screens (such as check-in, search, and check-out) can be created using Content Profiles.

CREATING AND MANAGING COMPONENTS

Components are modular programs that are designed to interact with the content server at runtime. Custom components enable you to customize Content Server without affecting the core functionality of the software. Components can be used to alter defaults for your system, to add new functionality, or to streamline repetitive functions.

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.

Figure 5-4 Component wizard interface

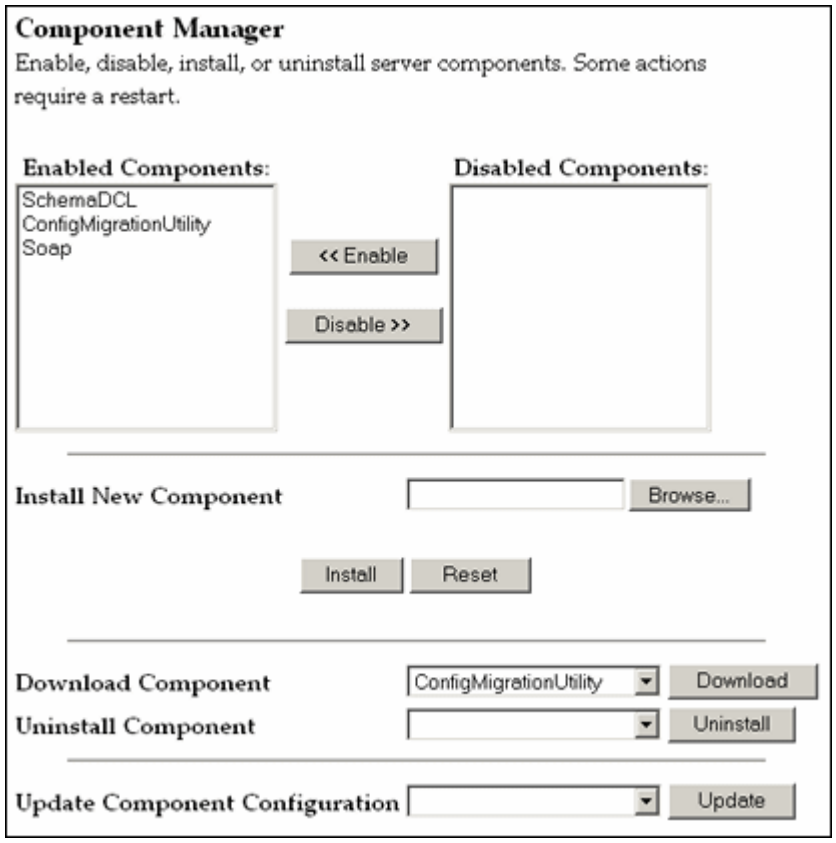


Through this interface, you can create and edit all of the files necessary for a custom component, as well as alter any components that are already installed.

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

To access the Component Manager, click **Admin Server** on the Application menu. Click on the server name. The options page for that server appears. Click **Component Manager** to display the Component Manager page.

Figure 5-5 Component Manager page



The Component Manager lists all enabled and disabled components and provides an easy interface to add new components.

See the *Working with Components* guide for details about creating and using custom components.

INSTALLING COMPONENTS

An extensive group of components are also available that can be downloaded. Documentation for components are included with the component and can be downloaded when you download the component.

You can install these components using one of two methods:

- Using Component Manager (page 5-6).
- Using Component Wizard (page 5-6).

Before installing a component, you must first download it to your instance.

Using Component Manager

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

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

   The Content Admin Server page is displayed.

2. Click the instance name of the server where the component will be installed.

   The Content Server <*instance-name*> page is displayed.

3. Click **Component Manager**.

   The Component Manager page is displayed.

4. Click the **Browse** button and find the zip file that was downloaded and saved.

5. Highlight the component name and click **Open**.

6. Click **Install**. A message is displayed, detailing what will be installed.

7. Click **Continue** to continue with installation or **Cancel** to stop installation.

8. 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. You are returned to the Content Server <*instance-name*> page, where you can restart the 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, return to the Content Server <*instance-name*> page and restart the server.

Using Component Wizard

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

1. Start the Component Wizard by selecting **Start—Programs—Content Server—** <*install_dir*>**—Utilities—Component Wizard** (Windows) or by running the ComponentWizard script in the /bin directory (in UNIX).

   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.

3. Click **Select**.
4. Navigate to the zip file that was downloaded and saved and select it.
5. Click **Open**.

   The zip file contents are added to the Install screen list.
6. Click **OK**. You are prompted to enable the component.
7. Click **Yes**. The component is listed as enabled on the Component List screen.
8. Exit the Component Wizard.

Restart the 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.

USING COMPONENTS

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.

Components are modular programs that are designed to interact with the content server at runtime. 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.

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.** Once 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 will 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 may 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 system-wide.** 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.

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

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**. Variables can be used 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**. Conditionals can be used 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 *Idoc Script Reference Guide* for details on usage and for syntax and configuration variable information.

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 <install_dir>/shared/config/resources/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 *Services Reference Guide* for more details.

GENERATING ACTION MENUS

In previous versions of the 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 <*install_dir*>/shared/config/resources/std_page.htm 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, Classic View, or Thumbnail View tables on the server, as well as any of the action menus.

Creating Display Tables

Different display tables are used for the search results page for each display type (Headline, Classic, or Thumbnail) with an API for each. Each type is discussed in the follow sections:

- Headline View Tables (page 5-12)
- Classic View Tables (page 5-14)
- Thumbnail View Tables (page 5-15)

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

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

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 a number of 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 recreate 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 /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.

Classic View Tables

Classic View table creation is almost identical to Headline View table creation. The major differences center around different variables and different include names, as shown in the following table:

Classic View Table	Headline View Table
`classic_table_header`	`slim_table_header`
`classic_table_footer`	`slim_table_footer`
`create_classic_table_row_include`	`create_slim_table_row_include`
`classic_table_row_include`	`slim_table_row_include`

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.

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

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 a number of 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 should be unique to the application, and can be used 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 originally 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 IdocScript 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. This enables Content Server administrators and component writers to hide specific actions either globally or for specific pages.

Component writers also can override a number of IdocScript resource includes to modify functionality in this area on either a global or targeted scale. The following includes are just a few of the many resource includes that can be used:

- 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

Chapter

6

INTEGRATION METHODS

OVERVIEW

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

The 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 via 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 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.

For general information on the architecture and functionality of the Content Server product, as well as a number of key concepts, refer to the *Content Server Getting Started Guide*.

This chapter discusses the following integration options:

IDC UTILITIES (COMMAND LINE/SHELL SCRIPTING)

The IdcCommand utility is a stand-alone 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. The IdcCommand utility only returns information about the success or failure of the command. To retrieve information from the content server in an interactive session, use the IdcCommandX Java COM wrapper available on Windows platforms (see COM Integration (page 6-3) for details).

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

- 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 stand-alone 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.

The IdcCommand utility returns information about only the success and failure of the command. To retrieve information from the server in an interactive session, use the Java COM wrapper IdcCommandUX available on Microsoft Windows.

For details about using Idc Utilities, see the *Idc Command Reference Guide.*

COM INTEGRATION

Content Server utilizes a Component Object Model-based API which provides the capability to call Oracle 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 Component Object Model (COM) interface.

Tech 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 (page 6-3)
- OCX Interface (page 6-4)
- ODMA Integration (page 6-5)

ActiveX Interface

The IdcCommandUX ActiveX Command Utility is an ActiveX control that allows a program to execute a service and retrieve file path information. You can call the IdcCommandUX utility from both a Microsoft Visual Basic and Microsoft Visual C++

environment. For more detailed information about IdcCommandUX, see the *IdcCommand Reference Guide*.

Run the IdcCommandUX ActiveX Command Utility setup executable located on the support site.

When executing services using the IdcCommandUX ActiveX Command Utility keep these notes 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.

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 *IdcCommand Reference Guide* for additional information. This guide also contains information about the IntradocClient OCX API specifications listing the properties, methods, and events.

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.

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.

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.

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)* (page 6-10).

This section covers the following topics:

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 IdocScript functions can also be executed from JSP pages which reside as executable content in the Content Server.

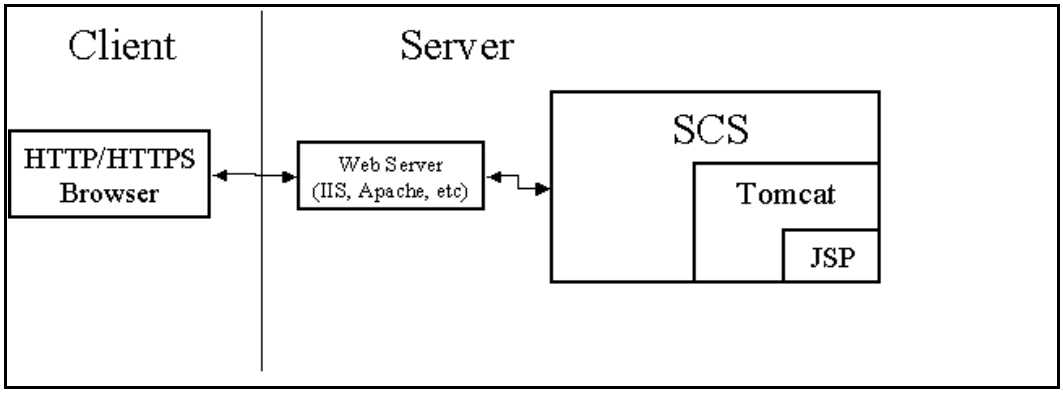
Important: JSP pages can execute IdocScript functions only when the JSP page is being served on the content server as part of the JSP Execution functionality. JSP pages served on a separate JSP server do not have this functionality. In those cases, checking a JSP page into the content server provides revision control but does not provide dynamic execution of IdocScript functions on the presentation tier (JSP server).

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.

Figure 6-6 Content Server/Tomcat integration architecture



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

Features

With JSP support enabled, custom components can include JSP pages of type `jsp` and `jspx`.

- The <*install_dir*>/weblayout/jsp/ directory is able to host JSP pages by default.
- The Content Server distribution media also includes the current Java 2 SDK.

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 steps below). This security group should be restricted to developers.

Note: This step is not required, but is recommended for developer convenience. Any security groups to be enabled for JSP must be specified in step 6.

   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 are running 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.

Loading Example Pages

Use either of the following procedures to load example pages into the Content Server:

1. Check in the .war file in the JSP security group. Make sure to check in other content to the JSP security group prior to checking in the war file.

2. Start the JSP Server Web App Admin from the Administration page.

JAVA 2 ENTERPRISE EDITION INTEGRATION (J2EE)

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

The Content Integration Suite (CIS) is based on J2EE standards for implementation. Any J2EE developer using J2EE-compliant systems will be able to develop integrations using the CIS.

Integration can be accomplished by using many different development environments and protocols. CIS also supports single sign-on capabilities, so users can log on to other systems and do not need to re-logon when those linked systems are needed to get content from the system.

This section covers these topics

See the Content Integration Suite documentation set for additional information.

Integration techniques

Content Server is set up as a series of Java services, which enable content to be accessed and manipulated as needed. These services include functions to search content, check the content in, view information about the content, approve the content through workflow, set expiration dates, and create subscriptions, to name a few applications. To use the system, everything goes through these services—even the native, web-based interface.

Content Integration Suite architecture

The Content Server Command Layer allows application developers to focus on presentation issues rather than being concerned with how to access IdcCommand services.

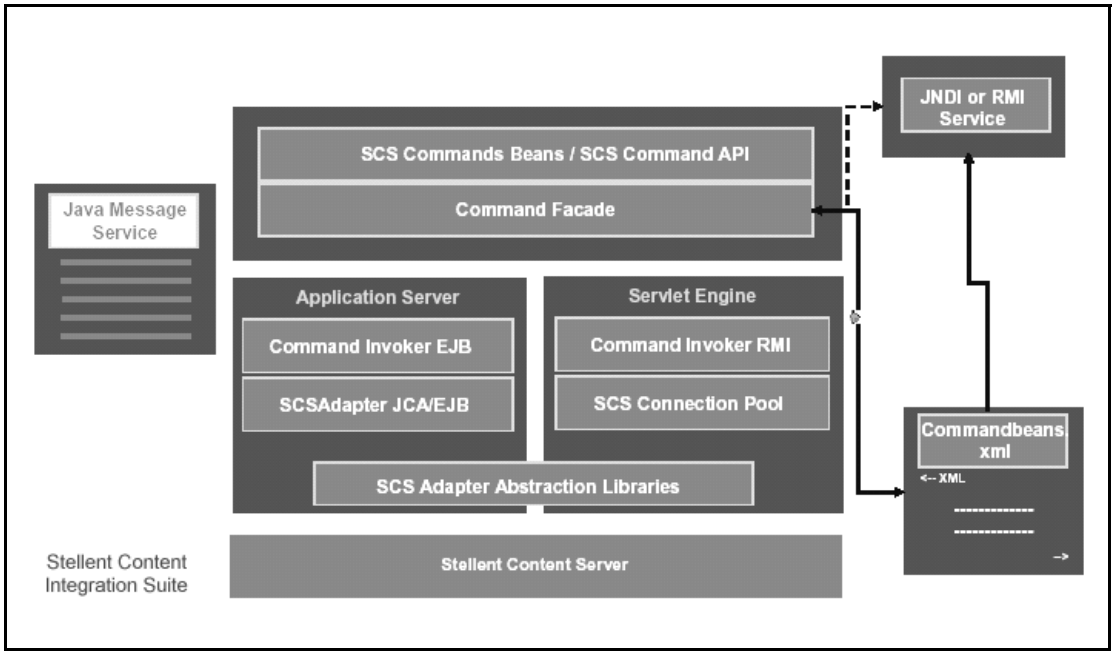
The CS Command Layer is a set of command objects, driven by Enterprise Java Beans (EJBs), that encapsulate distinct “commands” to the content server. These commands include common functions such as search, checkout, and workflow approval. Each command is tied to one or more service calls.

The CS Command Layer is based on the J2EE Command Design Pattern. It is made up of three tiers:

- CS Command Objects, via the SCS Command EJB
- The Command Facade/Invoker
- The CS Adapter EJB/JCA (J2EE Connector Architecture)

This infrastructure is deployable in any J2EE-compliant application server. The CS Command Layer encapsulates Content Server (CS) business logic and validates the parameters of the incoming calls.

Figure 6-7 SCS Command Layer



The SCS Command Facade/Invoker locates and instantiates a command by name. It executes a given command object and handles command execution (synchronously or asynchronously) based on command properties. The SCS Command Facade/Invoker can be invoked locally or remotely, so you do not have to have the CIS deployed on the same system as your requesting application server.

The SCS Adapter Layer handles communication to the content server, encapsulates socket communication logic (opening, validating, and streaming bits through the socket), and implements the JCA CCI (Command Client Interface).

See the *SCS Command Layer Developer's Guide* for information on SCS Command Layer usage, command invocation and invocation, and extending the SCS Command Layer.

Internationalization (character encoding)

The Java Virtual Machine encoding must be set to the same encoding as the Java Virtual Machine that is running the content server. If the Content Integration Suite is communicating with multiple instances of Content Server in different languages, the recommended approach is to set all the JVMs to Unicode (UTF-8) encoding.

WEB SERVICES

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

This section contains these topics:

Web Services Framework

Web services reside as a layer above 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.

Content Server provides SOAP capabilities built into the core product. Additionally, an extra component, the *WSDL Generator,* is available from the support site. For more information, see the *WSDL Generator and SOAP Guide,* which is included with the WSDL Generator component.

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).
- 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).

Note: While several protocols are available as a transport layer (e.g., HTTP, SMTP, FTP, BEEP), the HTTP protocol is most commonly used. The WSDL Generator Component relies on the HTTP protocol as the transport layer.

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 multi-level 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 Oracle Collaboration Server, WebDAV functionality, and the Oracle WebDAV Client product.
- The WebDAV component enables WebDAV (Web-Based Distributed Authoring and Versioning) functionality to remotely author and manage your Oracle 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 Oracle repository rather than using Oracle's web browser interface.

The option to install the WebDAV component is provided during the Folders/WebDAV installation process. See the *Folders and WebDAV Installation Guide, Folders and WebDAV User Guide,* and *Folders and WebDAV Administration Guide* for additional information.

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 multi-level 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 system-wide.
- There is one default system-level folder, called *Content Server Folders*. If you are using the Oracle Collaboration Server or 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 numerical "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.

WebDAV Integration

WebDAV (Web-Based Distributed Authoring and Versioning) provides a way to remotely author and manage your Oracle 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 Oracle repository rather than using Oracle's 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 (page 6-14) for additional information.

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.

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.

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 (via 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.

Appendix

A

THIRD PARTY LICENSES

OVERVIEW

This appendix includes a description of the Third Party Licenses for all the third party products included with this product.

APACHE SOFTWARE LICENSE

```
* Copyright 1999-2004 The Apache Software Foundation.

* Licensed under the Apache License, Version 2.0 (the "License");

* you may not use this file except in compliance with the License.

* You may obtain a copy of the License at

*      http://www.apache.org/licenses/LICENSE-2.0

*
```

```
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
```

W3C® SOFTWARE NOTICE AND LICENSE

```
* Copyright © 1994-2000 World Wide Web Consortium,
* (Massachusetts Institute of Technology, Institut National de
* Recherche en Informatique et en Automatique, Keio University).
* All Rights Reserved.  http://www.w3.org/Consortium/Legal/
*
* This W3C work (including software, documents, or other related items) is
* being provided by the copyright holders under the following license. By
* obtaining, using and/or copying this work, you (the licensee) agree that
* you have read, understood, and will comply with the following terms and
* conditions:
*
* Permission to use, copy, modify, and distribute this software and its
* documentation, with or without modification, for any purpose and without
* fee or royalty is hereby granted, provided that you include the following
* on ALL copies of the software and documentation or portions thereof,
* including modifications, that you make:
*
*   1. The full text of this NOTICE in a location viewable to users of the
*      redistributed or derivative work.
*
*   2. Any pre-existing intellectual property disclaimers, notices, or terms
```

```
*      and conditions. If none exist, a short notice of the following form
*      (hypertext is preferred, text is permitted) should be used within the
*      body of any redistributed or derivative code: "Copyright ©
*      [$date-of-software] World Wide Web Consortium, (Massachusetts
*      Institute of Technology, Institut National de Recherche en
*      Informatique et en Automatique, Keio University). All Rights
*      Reserved. http://www.w3.org/Consortium/Legal/"
*
*   3. Notice of any changes or modifications to the W3C files, including the
*      date changes were made. (We recommend you provide URIs to the location
*      from which the code is derived.)
*
* THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS
* MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
* NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR
* PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE
* ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
*
* COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR
* CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR
* DOCUMENTATION.
*
* The name and trademarks of copyright holders may NOT be used in advertising
* or publicity pertaining to the software without specific, written prior
* permission. Title to copyright in this software and any associated
* documentation will at all times remain with copyright holders.
*
```

ZLIB LICENSE

```
* zlib.h -- interface of the 'zlib' general purpose compression library
  version 1.2.3, July 18th, 2005

Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler
This software is provided 'as-is', without any express or implied
  warranty.  In no event will the authors be held liable for any damages
  arising from the use of this software.

  Permission is granted to anyone to use this software for any purpose,
  including commercial applications, and to alter it and redistribute it
  freely, subject to the following restrictions:

  1. The origin of this software must not be misrepresented; you must not
     claim that you wrote the original software. If you use this software
     in a product, an acknowledgment in the product documentation would be
     appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be
     misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.

  Jean-loup Gailly jloup@gzip.org
  Mark Adler madler@alumni.caltech.edu
```

GENERAL BSD LICENSE

Copyright (c) 1998, Regents of the University of California

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

"Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

"Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

"Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

GENERAL MIT LICENSE

Copyright (c) 1998, Regents of the Massachusetts Institute of Technology

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

UNICODE LICENSE

UNICODE, INC. LICENSE AGREEMENT - DATA FILES AND SOFTWARE

Unicode Data Files include all data files under the directories http://www.unicode.org/Public/, http://www.unicode.org/reports/, and http://www.unicode.org/cldr/data/ . Unicode Software includes any source code published in the Unicode Standard or under the directories http://www.unicode.org/Public/, http://www.unicode.org/reports/, and http://www.unicode.org/cldr/data/.

NOTICE TO USER: Carefully read the following legal agreement. BY DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING UNICODE INC.'S DATA FILES ("DATA FILES"), AND/OR SOFTWARE ("SOFTWARE"), YOU UNEQUIVOCALLY ACCEPT, AND AGREE TO BE BOUND BY, ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE, DO NOT DOWNLOAD, INSTALL, COPY, DISTRIBUTE OR USE THE DATA FILES OR SOFTWARE.

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1991-2006 Unicode, Inc. All rights reserved. Distributed under the Terms of Use in http://www.unicode.org/copyright.html.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Unicode data files and any associated documentation (the "Data Files") or Unicode software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that (a) the above copyright notice(s) and this permission notice appear with all copies of the Data Files or Software, (b) both the above copyright notice(s) and this permission notice appear in associated documentation, and (c) there is clear notice in each modified Data File or in the Software as well as in the documentation associated with the Data File(s) or Software that the data or software has been modified.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

_____Unicode and the Unicode logo are trademarks of Unicode, Inc., and may be registered in some jurisdictions. All other trademarks and registered trademarks mentioned herein are the property of their respective owners

MISCELLANEOUS ATTRIBUTIONS

Adobe, Acrobat, and the Acrobat Logo are registered trademarks of Adobe Systems Incorporated.

FAST Instream is a trademark of Fast Search and Transfer ASA.

HP-UX is a registered trademark of Hewlett-Packard Company.

IBM, Informix, and DB2 are registered trademarks of IBM Corporation.

Jaws PDF Library is a registered trademark of Global Graphics Software Ltd.

Kofax is a registered trademark, and Ascent and Ascent Capture are trademarks of Kofax Image Products.

Linux is a registered trademark of Linus Torvalds.

Mac is a registered trademark, and Safari is a trademark of Apple Computer, Inc.

Microsoft, Windows, and Internet Explorer are registered trademarks of Microsoft Corporation.

MrSID is property of LizardTech, Inc. It is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending.

Oracle is a registered trademark of Oracle Corporation.

Portions Copyright © 1994-1997 LEAD Technologies, Inc. All rights reserved.

Portions Copyright © 1990-1998 Handmade Software, Inc. All rights reserved.

Portions Copyright © 1988, 1997 Aladdin Enterprises. All rights reserved.

Index

D

E

F

G

H

I

J

L

M

O

P

Q

R

S

T

U

V

W

X