Working with Content Server Components
10*g* Release 3 (10.1.3.3.0)

March 2007

ORACLE

Working with Components, 10*g* Release 3 (10.1.3.3.0)

Contributing Authors: Jean Wilson

Table of Contents

Chapter 1: About This Guide

What's New 1-1

Audience 1-2

Document Organization 1-2

Conventions 1-3

Chapter 2: Component Overview

Component Wizard 2-1

Component Manager 2-2

Component Files Overview 2-3

Enabling and Disabling a Component 2-4

Chapter 3: Directories and Files

HDA Files 3-1

- Elements in HDA files 3-2
- The components.hda File 3-5
- Glue Files. 3-6

Custom Resource Files 3-7

DataBinder 3-8

- LocalData. 3-9
- Active or Non-active ResultSets 3-9
- Environment. 3-9

Manifest File 3-9

Other Files 3-11

- Customized Site Files 3-11
- Component Zip File 3-11
- Custom Installation Parameter Files 3-12

Typical Directory Structure 3-12

Chapter 4: Development Recommendations

Creating a Component 4-1
Working with Component Files 4-2
Using a Development Instance 4-2
Component File Organization 4-3
Naming Conventions 4-4

Chapter 5: Using the Component Wizard

Overview 5-1
Working with Java Code 5-2
Editing the Readme File 5-3
Creating a Custom Component 5-3
Additional Component Wizard Tasks 5-15
Building a Component Zip File 5-15
Working With Installation Parameters 5-16
Enabling and Disabling a Component 5-18
Removing a Component 5-19
Opening a Component 5-19
Configuring the Default HTML Editor 5-20
Unpackaging a Component 5-20
Adding an Existing Component 5-21

Chapter 6: Using the Component Manager

Component Manager Main Page 6-1
Component Manager Tasks 6-3
Enabling and Disabling a Component 6-3
Uploading a Component 6-4
Downloading a Component 6-5

Chapter 7: Component File Details

The components.hda file 7-1
Components ResultSet 7-2
Component Definition (Glue) File 7-3
ResourceDefinition ResultSet 7-5
MergeRules ResultSet 7-6

Filters ResultSet 7-8
ClassAliases ResultSet 7-8

Chapter 8: Resources Detail

HTML Include 8-2
The Super Tag 8-2
Editing an HTML Include Resource 8-3
String 8-4
String Parameters 8-6
Editing a String Resource 8-8
Dynamic Tables 8-8
Editing a Dynamic Table Resource 8-8
Static Tables 8-9
Editing a Static Table Resource 8-9
Query 8-9
Editing a Query Resource 8-11
Service 8-12
Service Example 8-16
Editing a Service Resource 8-23
Templates 8-23
Template and Report Pages 8-26
Editing a Template Resource 8-32
Environment 8-32
Environment Example 8-33
Editing an Environment Resource 8-34

Chapter 9: Component Interface Screens

Options Menu 9-5
Build Menu 9-5
Help Menu 9-6
Component Creation Screens 9-6
Build Screens 9-41

Chapter

1

ABOUT THIS GUIDE

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.

This document describes the details of *component architecture*, which is the use of modular components to customize standard content server functionality.

This chapter discusses these topics:

- What's New (page 1-1)
- Audience (page 1-2)
- Document Organization (page 1-2)
- Conventions (page 1-3)

WHAT'S NEW

A new online tool is available to view details about services which can be used in new components. See Creating a Custom Component (page 5-3) for information about using the Preview Information for Service screen. See Preview Information for Service Screen (page 9-34) and Preview Action Information Screen (page 9-35) for screen shots and a description of this screen's use.

AUDIENCE

This guide is intended for developers who need to customize the Content Server software to suit content management needs specific to a particular business or organization. You should be familiar with content server architecture and have a working knowledge of adding custom services and resources into your current environment.

DOCUMENT ORGANIZATION

This manual contains reference information as well as a tutorial to assist you in learning about creating custom components. It includes the following chapters:

- Chapter 1 (*About This Guide)* outlines the audience, organization, and conventions for this guide, and describes content management product distinctions.
- Chapter 2 (*Component Overview)* provides an overview of the building blocks you'll need to create a custom component.
- Chapter 3 (*Directories and Files)* describes the types of files and data you will work with when you create a custom component.
- Chapter 4 (*Development Recommendations)* provides an overview of the development process and tips on getting started with component development.
- Chapter 5 (*Using the Component Wizard)* describes the process of creating components and provides a tutorial for component creation.
- Chapter 6 (*Using the Component Manager)* provides details of tasks you can perform with the Component Manager interface.
- Chapter 7 (*Component File Details)* describes the files that define and implement the actual customizations you make to the content server.
- Chapter 8 (*Resources Detail)* describes the custom resources that are created when components are created.
- Chapter 9 (*Component Interface Screens)* provides copies of the interface screens used to create components and a description of the options on those screens.

An index is provided at the end of this guide.

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.
	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.
	This is a notice that the information is new for this release.

Chapter

2

COMPONENT OVERVIEW

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

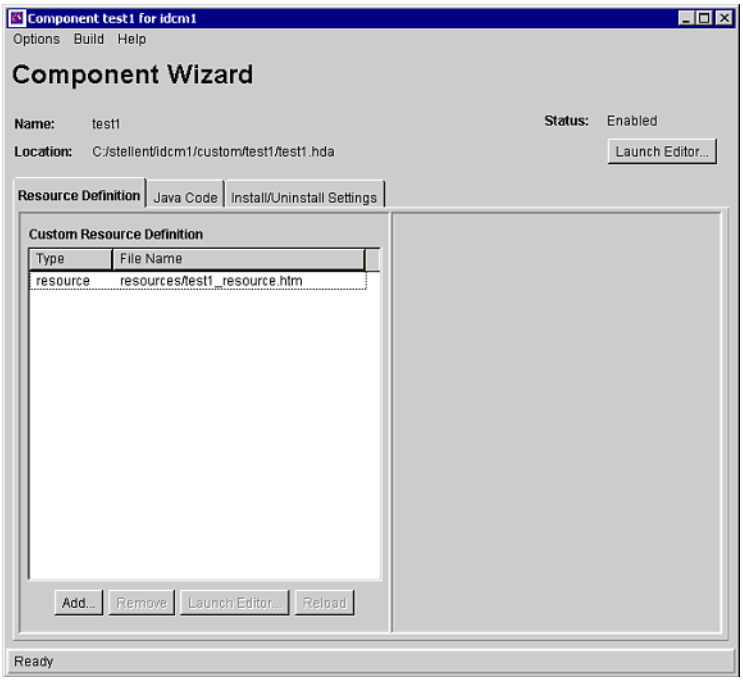
- Component Wizard (page 2-1)
- Component Manager (page 2-2)
- Component Files Overview (page 2-3)
- Enabling and Disabling a Component (page 2-4)

COMPONENT WIZARD

The Component Wizard is a development tool that 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.

To access the Component Wizard in a Windows environment, click **Start—<*install_dir*>—Utilities—Component Wizard**. The Component Wizard main page is displayed. To access the Wizard in a UNIX environment, run ComponentWizard.exe, stored in <*install_dir*>/bin.

Figure 2-1 Component Wizard main screen



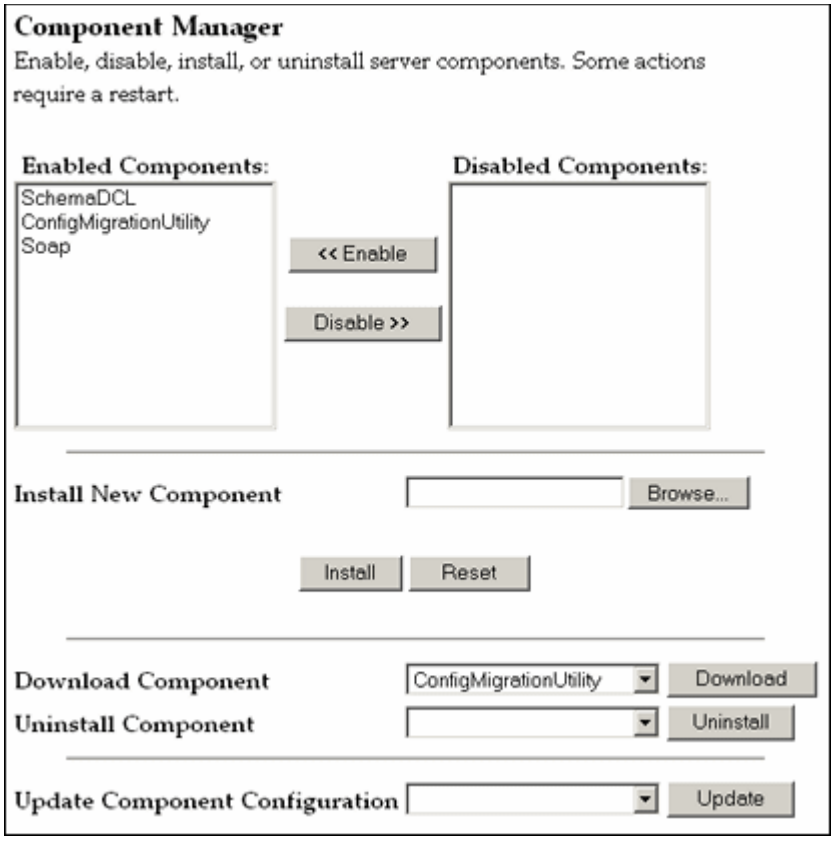
The Component Wizard is discussed in more detail in Chapter 5 (*Using the Component Wizard)* and Chapter 9 (*Component Interface Screens)*.

COMPONENT MANAGER

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. You can also access and edit some of the files used in components.

To use the Component Manager, click **Admin Server** on the Application menu in your Content Server. Click on the server name. The Options page for that server appears. Click **Component Manager** to display the Component Manager page.

Figure 2-2 Component Manager page



The Component Manager is discussed in more detail in Chapter 6 (*Using the Component Manager)*.

COMPONENT FILES OVERVIEW

When you define a custom component, you'll create or make changes to the following files:

- The component.hda file, which tells the Content Server what components are enabled and where to find each "glue" file.
- The component "glue" file, which tells the Content Server where to find the resources for the custom component.
- Different custom resource files, which define your customizations to standard Content Server resources.

- Template files, which define custom template pages.
- Other files which contain customizations to Content Server graphics, Java code, help files, and so forth.

These files are all discussed in more detail in Chapter 3 (*Directories and Files)*.

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

- HDA
- HTM
- CFG
- Java CLASS

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

- A compressed zip file used to deploy a component on other Content Servers.
- A manifest.hda file that tells the Content Server where to place the files that are unpackaged or uploaded from a component zip file.

ENABLING AND DISABLING A COMPONENT

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

There are three ways to enable or disable a component:

- **Manual editing—**Open the components.hda file in a text editor and add or delete the two-line entry for the component.
- **Component Wizard—**Select **Enable** or **Disable** from the Options menu. See Chapter 5 (*Using the Component Wizard)* for details.
- **Component Manager—**Select **Enable** or **Disable** from the Component Manager Instance screen. See Chapter 6 (*Using the Component Manager)* for details.

Note: You must restart the Content Server after changing enabling or disabling a component.

Chapter

3

DIRECTORIES AND FILES

This chapter provides information about the files used in component creation and the directory structure used to store those files. It discusses the following topics:

- HDA Files (page 3-1)
- Custom Resource Files (page 3-7)
- DataBinder (page 3-8)
- Manifest File (page 3-9)
- Other Files (page 3-11)
- Typical Directory Structure (page 3-12)

HDA FILES

A HyperData File (HDA) is used to define properties and tabular data in a simple, structured ASCII file format. It is a text file that is used by the Content Server to determine which components are enabled and disabled and where to find the glue files for that component.

The HDA file format is useful for data that changes frequently because the compact size and simple format make data communication faster and easier for the Content Server.

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

- components.hda file
- Component definition (glue) file
- Manifest file

- Dynamic table resource file
- Template resource file

The following example file is a components.hda file that points to a component called *customhelp*.

Figure 3-3 Sample components.hda file

```
<?hda version="5.1.1 ALPHA-011029" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
@end
@ResultSet Components
2
name
location
customhelp
custom/customhelp/customhelp.hda
@end
```

Elements in HDA files

Each HDA file contains a *header line* and one or more *sections*. The header line identifies the Content Server version, character set, and Java encoding for the HDA file. If an HDA file contains double-byte (Asian language) characters, the correct character set and encoding must be specified so the Content Server can read the file properly. The header line is not required for single-byte characters, but it is a good practice to include it in your HDA files. See the *Content Server Installation Guide for Windows* or the *Content Server Installation Guide for UNIX* for more information on locales.

The Properties Section and ResultSet Section are the two section types that are relevant to component development. These are used to define the Properties of the file (name, location, and so on) and the ResultSet which defines a table or columns and rows of data. ResultSets often represent the results of a query. All other sections tags are for internal application use only.

Comments are not allowed within a section of an HDA file. However, you can place comments in the HDA file before the first section, between sections, or after the last section. Blank lines within a section of an HDA file are interpreted as a NULL value. Blank lines before the first section, between sections, or after the last section are ignored. None of the section types are mandatory in an HDA file, so unused sections can be deleted.

- The Properties section contains a group of name/value pairs. For a custom component, the most common name for a Properties section is *LocalData*, which means that the name/value pairs are valid only for the current HDA file.

  You can also define global name/value pairs in a Properties section called *Environment*, but this section type is rarely used. The recommended practice is to define global environment variables in a configuration file in an Environment resource.

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

Figure 3-4 Properties in HDA file

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

- The ResultSet section of an HDA file defines a table, or columns and rows of data. A ResultSet can be used to pass information to a database or to represent the results of a database query. A ResultSet section has the following structure:
  - The first line defines the name of the ResultSet table using the format `@ResultSet` *resultset_name*.
  - The second line defines the number of columns.
  - The next *n* lines define the column names.
  - The remaining lines define the values in each cell of the table.
  - The last line of the section ends the table using the format `@end`.

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

Figure 3-5 Sample ResultSet

```
@ResultSet Scores
4
name
match1
match2
match3
Margaret
68
67
72
Sylvia
70
66
70
Barb
72
71
69
@end
```

The following table shows the ResultSet data in a columnar form. A ResultSet can be given any name.

name	**match1**	**match2**	**match3**
Margaret	68	67	72
Sylvia	70	66	70
Barb	72	71	69

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

ResultSet Name	**Location**	**Purpose**
Components	<*install_dir*>/config/components.hda	Defines the name and location of any custom components you have created.

ResultSet Name	Location	Purpose
IntradocReports	<*install_dir*>/shared/config/reports/reports.hda	Specifies the default report templates for the Content Server.
IntradocTemplates	<*install_dir*>/shared/config/templates/templates.hda	Specifies all of the default templates for the Content Server (except for search results and report templates).
ResourceDefinition	<*install_dir*>/custom/*component_name*/*component_name*.hda	Defines resources for a custom component.
SearchResultTemplates	<*install_dir*>/shared/config/templates/templates.hda	Specifies the default search results templates for the Content Server.

The components.hda File

The components.hda file is a text file that tells the Content Server which components are enabled and where to find the “glue” file for each component.

The components.hda file is always stored in the <*install_dir*>/config/ directory. The Component Wizard and Component Manager can be used to make changes to this file if needed.

The following is an example of a components.hda file, listing several enabled components such as schema, configuration migration, and SOAP.

Figure 3-6 Coponents.hda file with enabled components

```
@properties LocalData
blDateFormat=M/d/yy
@end
@ResultSet Components
2
name
location
SchemaDCL
custom/SchemaDCL/SchemaDCL.hda
ConfigMigrationUtility
custom/ConfigMigrationUtility/Cmu.hda
Soap
custom/Soap/Soap.hda
@end
```

Glue Files

A *component definition file* points to the custom resources that you have defined. This file specifies information about custom resources, ResultSets, and merge rules. Because it serves as the "glue" that holds a component together, the component definition file is also called the *glue file*.

The glue file for a component is typically named *component_name*.hda, and is located in the <*install_dir*>/custom/*component_name*/ directory. The Component Wizard can be used to create and make changes to a glue file.

Note: Do not confuse the components.hda file with the *component_name*.hda file. The components.hda file is used to track all installed components. The *component_name*.hda file contains information that is specific to a single component.

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

Figure 3-7 Sample glue file

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
blFieldTypes=
@end
@ResultSet ResourceDefinition
4
type
filename
tables
loadOrder
environment
customhelp_environment.cfg
null
1
@end
```

CUSTOM RESOURCE FILES

Custom resource files define your Content Server customizations. They are usually HDA files but some are HTM files.

The custom resource files for a component are typically located in the <*install_dir*>/custom/*component_name*/ directory. Some resource files may be placed in subdirectories such as /resources or /templates.

The following table describes these resources:

Resource Type	File Type	Contents
HTML include	HTM	“Include” definitions
String	HTM	Localized string definitions
Dynamic table	HDA	Tables for data that changes often
Static table	HTM	Tables for data that seldom changes
Query	HTM	Tables that define queries

Resource Type	File Type	Contents
Service	HTM	Tables that define service scripts
Template	HDA	Tables that specify location and file name for template pages
Environment	CFG	Configuration variable name/value pairs

These files are all discussed in detail in Chapter 8 (*Resources Detail)*.

In addition, a template.htm page is used by the Content Server to assemble web pages. See Templates (page 8-23) for details about the template.hdm file.

A ResultSet HTM table file is used by other resources. A ResultSet table in an HTM file is similar to the The ResultSet section of an HDA file defines a table, or columns and rows of data. A ResultSet can be used to pass information to a database or to represent the results of a database query. A ResultSet section has the following structure: (page 3-3) of an HDA file, except that it uses HTML table tags to lay out the data. Static table resources, service resources, and query resources all use this table format.

A ResultSet table in an HTM file begins with `<@table` *table_name*`@>` and ends with `<@end@>`. The markup between the start and end tags is an HTML table. Unlike a ResultSet in an HDA file, the number of columns is implied by the table tags.

Any HTML syntax that does not define the data structure is ignored when the table is loaded. Therefore, HTML comments are allowed within tables in an HTM file, and HTML style attributes can be used to improve the presentation of the data in a web browser.

DataBinder

The Content Server caches data (such as variable values and lookup keys) internally in a DataBinder. All data in the DataBinder is categorized according to where it came from and how it was created. When a value is required to fulfill a service request, the data in the DataBinder is evaluated in the following default order:

1. LocalData (page 3-9)
2. Active or Non-active ResultSets (page 3-9)
3. Environment (page 3-9)

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

LocalData

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

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

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

Active or Non-active ResultSets

Each `@ResultSet` section of an HDA file maps to a named result in the DataBinder. A ResultSet becomes active when the ResultSet is looped on during page assembly. The active ResultSet take precedence over any other ResultSets during a value search of the DataBinder. When a service request requires data and the value is not found in the LocalData or an active ResultSet, the non-active ResultSet values are searched next.

Environment

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

MANIFEST FILE

Manifest files are used to upload or unpackage a component zip file on the Content Server. This file tells the Content Server where to place the individual files that are included in the component zip file. A manifest file is created automatically when you build a component in the Component Wizard, or when you download a component using the Admin Server Component Manager.

All manifest files must be called manifest.hda. The manifest.hda file is included in the component zip file along with the other component files. It must be at the top level of the zip file directory structure.

The manifest.hda file contains a ResultSet table called *Manifest*, which consists of two columns:

❖ The `entryType` column defines the type of entry in the manifest file.

Entry Type	Description	Default Path
Classes	Java class files	<*install_dir*>/classes
Common	Common files	<*install_dir*>/weblayout/common
Component	Component resource files	<*install_dir*>/custom
ComponentExtra	Associated files, such as a readme	<*install_dir*>/custom
Help	Online help files	<*install_dir*>/weblayout/help
Images	Graphics files	<*install_dir*>/weblayout/images
Jsp	Java Server Pages	<*install_dir*>/weblayout/jsp

❖ The `location` column defines the directory where the files associated with the entry will be installed, and specifies the file name for some entry types.

- For a *Component* entry type, the location is the path and file name for the glue file. The glue file then tells the Content Server which resource files are included in the component.
- For other entry types, the location can be a path (to specify all files in a particular sub-directory) or a path and file name (to specify an individual file).
- The location should be a path relative to the <*install_dir*>/custom/ directory. You can use an absolute path, but then the component can only be installed on Content Servers with the same installation directory path.

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

Figure 3-8 Example manifest.hda file

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

OTHER FILES

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

Customized Site Files

You can add customized files for your site in order to change the look or actions of the Content Server. For example, the following types of files are often referenced in custom resources:

- **Graphics—**Replace the icons, backgrounds, and logos that make up the standard Content Server interface.
- **Help—**With the assistance of Consulting Services, help files can be customized for your content management system.
- **Classes—**Java code can change or extend the functionality of the Content Server. Java class files must be packaged into directories before placing them in the <*install_dir*>/classes/ directory.

Component Zip File

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

Custom Installation Parameter Files

When you define one or more custom installation parameters, several additional files are created in addition to the files that compose the basic component file structure.

If installation parameters are created for the component, then during the component installation process the component installer automatically places two files in the component directory within the data/components directory. These files hold the preference data as follows:

- config.cfg: Contains the parameters that can be reconfigured after installation.
- install.cfg: Contains the preference data definitions and prompt answers.
- Backup zip file: A backup file that is created if the component is already installed and is being reinstalled.

TYPICAL DIRECTORY STRUCTURE

If you use the Component Wizard to create custom components, your files will be stored in the appropriate directory.

The component.hda file is always stored in the <*install_dir*>/config/ directory.

Different component directories are established for each custom component in the <*install_dir*>/custom directory. Within each component directory, separate subdirectories are established for reports, templates, and resources, all named appropriately (for example, /Resources). The *component_name*.hda file (the 'glue' file) is stored in the /*component_name* directory.

Chapter

4

DEVELOPMENT RECOMMENDATIONS

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

- Creating a Component (page 4-1)
- Working with Component Files (page 4-2)
- Using a Development Instance (page 4-2)
- Component File Organization (page 4-3)
- Naming Conventions (page 4-4)

See Chapter 5 (*Using the Component Wizard)* for detailed instructions on creating or modifying components.

CREATING A COMPONENT

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

1. Create a glue file.
2. Add a reference to the glue file in the components.hda file to enable the component.
3. Restart the Content Server to apply the component.
4. Create resources and other files to define your customizations. A good approach is to copy, rename, and modify standard Content Server files to create your custom resource files.

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

WORKING WITH COMPONENT FILES

There are two ways to work with component files:

- **Component Wizard—**The Component Wizard is a Content Server utility that helps you create and edit component files. You can also use the Component Wizard to package, unpackage, enable, and disable components. See Chapter 5 (*Using the Component Wizard)* for more information.
- **Text editor—**Because most component files are plain text files, you can create and edit the files in your favorite text editor.

You should use the Component Wizard as much as possible when working with custom components.

The Component Wizard does a number of tasks for you and minimizes the amount of work you need to do in a text editor. Using the Component Wizard will help you follow the recommended file structure and naming conventions. The Component Wizard automatically adds a *readme* text file when you build a component, thus helping you to document your customizations. You should also include comments within your component files.

For instructions on using the Component Wizard to create components, see Chapter 5 (*Using the Component Wizard)*.

USING A DEVELOPMENT INSTANCE

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

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

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

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

COMPONENT FILE ORGANIZATION

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

Note: If you use the Component Wizard, it creates component directories for you and places the component files in the correct directories.

Place each custom component in its own directory within a directory called <*install_dir*>/custom/. If your custom component includes resource- and/or template-type resources, the component directory should have subdirectories that follow the structure of the <*install_dir*>/shared/config/ directory:

- **resources/** to hold HTML include and table resource files
- **resources/lang/** to hold string resource files
- **templates/** to hold template files
- **reports/** to hold report files

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

- Place the glue file for each custom component at the top level of the component's directory.
- When referring to other files within a component, use relative path names instead of absolute path names. This enables you to move the component to a different location without having to edit all of the files in the component.
- The Content Server is a Java-based application, so forward slashes must be used in all path names.
- Custom components do not have to be stored on the same machine as the Content Server, but all component files must be accessible to the Content Server.
- Images and other objects that are referenced by Content Server web pages must reside somewhere in the <*install_dir*>/weblayout/ directory (so they can be accessed by the web server).

NAMING CONVENTIONS

To keep your component files organized and make sure that the files work properly in the Content Server, follow these naming conventions for directories, individual files, and file contents.

- You should give all of your component directories and files unique and meaningful names. Keep in mind that as each component is loaded in the Content Server, it overrides any resources with the same file names, so you should use duplicate file names only if you want certain components to take precedence.
- If you are copying a standard Content Server file, a common practice is to place the prefix *custom_* in front of the original file name. This ensures that you do not overwrite any default templates, and your customizations are easy to identify.
- HTM file types should have a *.htm* extension, and HDA file types should have a *.hda* extension.
- If you are creating a new component file with a text editor like WordPad, place the file name within quotation marks in the Save dialog box so the proper file extension will be assigned to it (for example, *"myfile.hda"*). Failure to use quotation marks to define the file name may result in a file name such as *myfile.hda.txt*.
- The Content Server is case sensitive even if your file system is not. For example, when a file is named *My_Template*, the Content Server does not recognize case variations such as *my_template* or *MY_TEMPLATE*.
- For localized string resources, you must follow the standard file naming conventions for the Content Server to recognize the strings. You should also use the standard two-character prefix (cs, sy, ap, or ww) when naming your custom strings. See *Resolving Localized Strings* in *Customizing Content Server* for more information.

Chapter

5

USING THE COMPONENT WIZARD

This chapter describes how to use the Component Wizard to create new components. It also contains a tutorial on the creation of custom components. It contains the following major sections:

- Overview (page 5-1)
- Creating a Custom Component (page 5-3)
- Additional Component Wizard Tasks (page 5-15)

Note: When using the Component Wizard with Red Hat Linux ES 3, set UseCustomModaling=FALSE in your <*install_dir*>/bin/intradoc.cfg file. This variable allows a modal dialog to lock only one frame, instead of all frames. Setting the varaible in the intradoc.cfg file ensures that other applets are unaffected by this action. See the *Idoc Script Reference Guide* for details on its usage.

OVERVIEW

Use the following procedure to create a new component using the Component Wizard:

1. Start the Component Wizard by selecting **Start—<***install_dir***>—Utilities—Component Wizard**.
2. The Component List screen is displayed, showing all components and their status (enabled or disabled).

Note: If no components are installed, this screen does not appear.

3. If the Component List screen is displayed, select **Add**. If not, select **Add** from the Options menu on the Component Wizard main screen.

   The Add Component screen is displayed.

4. Enter a name for the new component in the Name field.
5. Accept the default directory (*custom*), or enter a new location for the component. This can be either an absolute path or can be a path relative to the Content Server install directory.
6. To use an existing component as a starting point, select the **Copy Existing** check box, click **Browse**, and navigate to and select the glue file (*component_name*.hda) for the component.
7. Click **OK**.

   A new component glue file is created. If you copied an existing component, the resource files are renamed with the new component name and copied to the new component directory.

8. Add and edit custom resources and other files as necessary as described in these sections:
   - Creating an Environment Resource (page 5-4)
   - Creating a Template Resource (page 5-5)
   - Creating a Query Resource (page 5-7)
   - Creating a Service Resource (page 5-8)
   - Creating an HTML Include (page 5-10)
   - Creating a String Resource (page 5-11)
   - Creating a Dynamic Table Resource (page 5-13)
   - Creating a Static Table Resource (page 5-14)

See Chapter 8 (*Resources Detail)* for details about editing custom resources.

Working with Java Code

If your custom component includes Java code, you can use the Java Code tab of the Component Wizard to view the contents of the ClassAliases table and the Filters table.

You can also remove classes and filters from the component glue file, although the file that is associated with the class or filter will not be deleted from your system. Select the class or filter and click the associated **Remove** button to remove it from the list.

Editing the Readme File

The Component Wizard provides a convenient way to create a "Readme" file for your custom component. Use the following procedure to edit a Readme file:

1. Open the component in the Component Wizard.
2. Select **Options—Edit Readme File**.

   The text editor opens a readme.txt file, with the name of the component entered on the first line.
3. Enter text to document your component.
4. Save and close the file.

   The readme.txt file is saved in the same directory as the component definition file, and will be included as a "ComponentExtra" entry if you use the Component Wizard to build a component zip file.

CREATING A CUSTOM COMPONENT

The following steps provide a general overview on using the Component Wizard to create a custom component.The screens used to create this component are described in detail in Chapter 9 (*Component Interface Screens)* and are referenced throughout the text. See Chapter 8 (*Resources Detail)* for information about editing any of these custom components.

1. Launch the Component Wizard.

   The Component Wizard main screen is displayed or the Component List screen is displayed if other components are already available (Component List Screen (page 9-2).
2. Click **Add** on the Component List screen.

   The Add Component screen is displayed (Add Component Screen (page 9-6)).
3. Make sure the **Create New Component** option is selected and enter the name of the new component.

4. Click **OK**.

   A confirmation screen is displayed.

5. Click **OK**.

   The Component List screen closes, and the new component is opened in the Component Wizard screen, as indicated by its name in the Location field.

Creating an Environment Resource

An environment resource defines configuration variables, either by creating new variables or replacing the value in existing variables.

Follow these steps to create an environment resource:

6. Make sure that the Resource Definition tab is selected on the Component Wizard main screen. Click **Add**.

   The Add Resource screen is displayed (Add Resource Screen (page 9-29)).

7. Select the **Environment** option.

8. Enter the file name for the resource file. The default file name is *componentname*_templates.hda.

   - ❖ If a resource file has been created, you can add to the file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
   - ❖ To create a new resource file with a different file name, enter the file name.

9. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

10. Click **Finish**.

    A dialog box asks if you want to launch the text editor to continue editing.

11. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The file now appears in the Custom Resource Definition list.

Note: If an HTML editor is not defined, select **Options—Configuration** from the Component Wizard main menu and enter the path and file name of the desired editor, or click **Browse** and navigate to the executable of the desired editor (for example, C:/Program Files/Windows NT/Accessories/wordpad.exe). See Configuring the Default HTML Editor (page 5-20) for details.

After saving, the new environment resource is displayed on the Component Wizard screen.

Creating a Template Resource

A template resource file defines names, types, and locations of custom templates to be loaded for the component. Follow these steps to add a template page:

12. Make sure that the Resource Definition tab is selected on the Component Wizard main screen. Click **Add**.

    The Add Resource screen is displayed (Add Resource Screen (page 9-29)).

13. Select the Template option. The Add Template Table Information screen is displayed (Add Template Table Information Screen (page 9-23)).

14. Enter the file name for the resource file. The default file name is *componentname*_templates.hda.

    - ❖ You can enter templates/ before the file name to create a new /templates directory in your component directory.
    - ❖ If a template resource file has been created, you can append a new template table to the existing file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
    - ❖ To create a new resource file with a different file name, enter the file name.

15. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

16. Click **Next**.

    The Add Template Table Information Screen (page 9-23) is displayed.

17. Enter a name for the template table.

    - ❖ It is a good idea to leave the name of the component as a prefix.

- ❖ Each template table in a component must have a unique name, even if the tables are in different resource files.

18. Select which standard table to merge the new template table into: either *IntradocTemplates* or *SearchResultTemplates*.
19. Click **Next**.

    The Add IntradocTemplate screen is displayed (Add/Edit Intradoc Template Screen (page 9-38)).
20. To start with an existing template definition:
    a. Click **Select**.

       A list of commonly used templates is displayed.
    b. Select the **Show All** check box to show the entire list of predefined templates.
    c. Select a template from the list.
    d. Click **OK**.

       The template parameters are filled in.

Note: You can also use an existing custom template file as a starting point. Select the **Copy From** check box, and navigate to and select the template file. The template parameters will not be filled in automatically, but you could select a standard template to fill in the fields before selecting the template file.

21. Edit the template parameters as necessary.

Note: If you do not change the name of the template and this component is loaded last, the custom template will override the standard template and any other custom templates with the same name.

22. Click **Finish**.

    A dialog box asks if you want to launch the text editor to continue editing.
23. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The file now appears in the Custom Resource Definition list, and the template table appears in the Table Name list in the right pane.

Creating a Query Resource

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

Follow these steps to add a query:

24. On the Component Wizard main screen, click **Add** in the Resource Definition pane.

    The Add Resource screen is displayed (Add Resource Screen (page 9-29)).

25. Select the **Query** option.

26. Enter the file name for the resource file. The default file name is resources/*componentname*_query.htm.

    - If a query resource file has been created with the default file name, the new default file name will have a number (1, 2, etc.) appended to it. You cannot append a query table to the existing default file unless you edit the resource file manually.
    - If a query resource file has been created with a file name other than the default, you can append a new query table to the existing file.
    - To create a new resource file with a different file name, enter the file name.

27. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

28. Click **Next**.

    The Add Query Table Information screen is displayed (Add Query Table Information Screen (page 9-16)).

29. Enter a name for the query table. It is a good idea to leave the name of the component as a prefix.

    - If you are appending to an existing query resource file, you must enter a new table name. You cannot append a query definition to the existing table unless you edit the resource file manually.

30. Click **Next**.

    The Add Query screen is displayed (Add/Edit Query Screen (page 9-27)).

31. To start with an existing query definition:

    a. Click **Select**.

A list of predefined queries is displayed.

b. Select a query from the list.

c. Click **OK**.

The query expression and parameters are displayed and the Name field is filled in.

Note: If you do not change the name of the query and this component is loaded last, the custom query will override the standard query and any other custom queries with the same name.

32. Edit the query expression and parameters as necessary.
    - ❖ Parameters must appear in the Parameters list in the order they appear in the query expression. Use the **Up** and **Down** buttons to move the selected parameter.
    - ❖ To add a parameter, click **Add**. Enter a parameter Name, select the parameter Type, and click **OK**.
    - ❖ To edit a parameter type, select the parameter and click **Edit**. Select the parameter Type, and click **OK**.
    - ❖ To remove a parameter, select the parameter and click **Delete**.
33. Click **Finish**.

    A dialog box asks if you want to launch the text editor to continue editing.
34. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The query resource file now appears in the Custom Resource Definition list, and the query table appears in the Table Name list in the right pane.

Creating a Service Resource

A service resource defines a function or procedure that is performed by the Content Server.

Use the following procedure to create a service resource using the Component Wizard.

35. In the Component Wizard, open the component the resource will be created for.
36. On the Resource Definition tab, click **Add**.

    The Add Resource Screen (page 9-29) is displayed.
37. Select the **Service** option.
38. Enter the file name for the resource file. The default file name is resources/*componentname*_service.htm.

- ❖ If a resource file has been created for services, you can append the new service table to the existing file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
- ❖ To create a new resource file with a different file name, enter the file name.

39. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

40. Click **Next**.

    The Add Service Table Information screen is displayed (Add Service Table Information Screen (page 9-17)).

41. Enter a name for the service table.
    - ❖ It is a good idea to leave the name of the component as a prefix.
    - ❖ Each service table in a component must have a unique name, even if the tables are in different resource files.
42. Click **Next**.

    The Add/Edit Service Screen (page 9-31) is displayed.

43. To start with an existing service definition:
    a. Click **Select**.

       A list of commonly used services is displayed.

    b. Select the **Show All** check box to show the entire list of predefined services.

    c. Select a service from the list.

       To view a service's details, click **Preview**. The Preview Information for Service Screen (page 9-34) is displayed. Use this screen to view information about the service and the service actions.

    d. Click **OK**.

       The service attributes and actions are filled in.

Note: If you do not change the name of the service and this component is loaded last, the custom service will override the standard service and any other custom services with the same name.

44. Edit the service attributes and actions as necessary.

- ❖ Actions must appear in the Actions list in order of execution. Use the **Up** and **Down** buttons to move the selected action.
- ❖ To add an action, click **Add**. The Add Action screen is displayed (Add/Edit Action Screen (page 9-10)). Enter the action definition and click **OK**.
- ❖ To edit an action, select the action and click **Edit**. Modify the action definition and click **OK**.
- ❖ To remove an action, select the action and click **Delete**.

45. Click **Finish**.

    A dialog box asks if you want to launch the text editor to continue editing.

46. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The service resource file now appears in the Custom Resource Definition list, and the service table appears in the Table Name list in the right pane.

Creating an HTML Include

An HTML include is a piece of reusable code that is referenced from a placeholder in another file or from another location in the same file. An include resource defines pieces of code that are used to build the Content Server web pages. Includes are resolved by the Content Server each time a web page is assembled. For this reason, includes are sometimes called dynamic content resources.

Follow these steps to add an HTML include resource:

47. On the Component Wizard main screen in the Resource Definition section, click **Add**.

    The Add Resource screen is displayed (Add Resource Screen (page 9-29)).

48. Select the **Resource - HTML Include/String** option.

49. Enter the file name for the resource file. The default file name is *componentname*_resource.htm.

    - ❖ If a resource file has been created for includes, strings, and/or static tables, you can append the include to the existing file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
    - ❖ To create a new resource file with a different file name, enter the file name.

50. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

51. Click **Next**.

    The Add HTML Resource Include/String screen is displayed (Add/Edit HTML Resource Include/String Screen (page 9-25)).

52. Select the **Include** option.

53. To start with the code from an existing HTML include:

    a. Click **Select**.

       A list of commonly used includes is displayed.

    b. Select the **Show All** check box to show the entire list of predefined includes.

    c. Select an include from the list.

    d. Click **OK**.

       The include code is displayed and the Name field is filled in.

Note: If you do not change the name of the include and this component is loaded last, the custom include will override the standard include and any other custom includes with the same name.

54. Edit the include code as necessary.

55. Click **Finish**.

    A dialog box asks if you want to launch the text editor to continue editing.

56. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The resource file now appears in the Custom Resource Definition list, and the include appears in the Custom HTML Includes list.

Creating a String Resource

A string resource defines locale-sensitive text strings that are used in error messages and on Content Server web pages and applets.

Use the following procedure to create a string resource using the Component Wizard.

57. In the Component Wizard, open the component the resource will be created for.
58. On the Resource Definition tab, click **Add**.

    The Add Resource Screen (page 9-29) is displayed.
59. Select the **Resource - HTML Include/String** option.
60. Enter the file name for the resource file. The default file name is *componentname*_resource.htm.
    - If a resource file has been created for includes, strings, and/or static tables, you can append the include to the existing file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
    - To create a new resource file with a different file name, enter the file name.
61. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

62. Click **Next**.

    The Add/Edit HTML Resource Include/String Screen (page 9-25) screen is displayed.
63. Select the **String** option.
64. Enter the name of the string in the **Name** field (for example, *myString*.)

Note: If you enter the name of an existing string and this component is loaded last, the custom string will override the standard string and any other custom strings with the same name.

65. Edit the string code as necessary (for example, *This is my string text*.)
66. Click **Finish**.

    A dialog box asks if you want to launch the text editor to continue editing.
67. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The resource file now appears in the Custom Resource Definition list, and the string appears in the Custom Strings list.

Creating a Dynamic Table Resource

A dynamic table provides dynamic (often changed) content in table format to the Content Server.

Use the following procedure to create a dynamic table resource using the Component Wizard.

68. In the Component Wizard, open the component the resource will be created for.
69. On the Resource Definition tab, click **Add**.

    The Add Resource Screen (page 9-29) is displayed.
70. Select the **Resource - Dynamic Table (Hda Format)** option.
71. Enter the file name for the resource file. The default path and file name is resources/*componentname*_resource.hda.
    - ❖ If a resource file has been created for dynamic tables, you can append the new table code to the existing file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
    - ❖ To create a new resource file with a different file name, enter the file name.
72. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

73. Click **Next**.

    The Add Dynamic Resource Table Information Screen (page 9-18) is displayed.
74. Enter a name for the dynamic table. It is a good idea to leave the name of the component as a prefix.
75. To merge the new table with an existing table, select the **Merge To** check box and select a table from the list.
76. Click **Finish**.
    - ❖ If you selected a table to merge to, a dialog box asks if you want to launch the text editor to continue editing.
    - ❖ If you did not select a table to merge to, the Column Information Screen (page 9-37) is displayed.
        a. Enter a column name in the Column Name field.

b. Click **Insert**. Repeat these steps until all of the table columns have been entered.

c. Click **OK**.

A dialog box asks if you want to launch the text editor to continue editing.

77. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

The resource file now appears in the Custom Resource Definition list, and the table appears in the right pane of the Resource Definition tab.

Creating a Static Table Resource

Use the following procedure to create a static table resource using the Component Wizard.

78. In the Component Wizard, open the component the resource will be created for.
79. On the Resource Definition tab, click **Add**.

The Add Resource Screen (page 9-29) is displayed.

80. Select the **Resource - Static Table (HTML Format)** option.
81. Enter the file name for the resource file. The default file name is *componentname_resource.htm*.
    - ❖ If a resource file has been created for static tables, includes, and/or strings, you can append the static table code to the existing file by selecting the file name. Any changes you make to the load order at this time will apply to the entire resource file.
    - ❖ To create a new resource file with a different file name, enter the file name.
82. If you want the new resource file to be loaded in a particular order, enter the number in the **Load Order** field.

Note: Unless you have a particular reason for the resource file to be loaded after other resources, you should leave the load order set to 1.

83. Click **Next**.

The Add Static Resource Table Information Screen (page 9-20) is displayed.

84. Enter a name for the static table. It is a good idea to leave the name of the component as a prefix.
85. To merge the new table with an existing table, select the **Merge To** check box and select a table from the list.

86. Click **Finish**.
    - ❖ If you selected a table to merge to, a dialog box asks if you want to launch the text editor to continue editing.
    - ❖ If you did not select a table to merge to, the Column Information Screen (page 9-37) is displayed.
        a. Enter a column name in the Column Name field.
        b. Click **Insert**.
        c. Repeat steps a and b until all of the table columns have been entered.
        d. Click **OK**.

        A dialog box asks if you want to launch the text editor to continue editing.
87. Click **Yes** to open the resource file in the text editor. Click **No** to return to the Component Wizard.

    The resource file now appears in the Custom Resource Definition list, and the table appears in the Resource Tables list.

Enabling the Component

After creating a component, you should enable it and test it.

- ❖ From the Component Wizard main screen, select **Options—Enable**.
- ❖ Restart the Content Server.
- ❖ Test the newly created component.

ADDITIONAL COMPONENT WIZARD TASKS

In addition to creating custom components, you can use the Component Wizard to build zip files of your components and create custom installation parameters.

Building a Component Zip File

The Build function of the Component Wizard enables you to build a component zip file (or 'package'), which can then be saved as a backup or unpackaged to deploy the component on other Content Servers.

Use the following procedure to build a component zip file:

1. Open the component in the Component Wizard.

2. Select **Build—Build Settings**.

   The Build Settings screen is displayed (Build Settings Screen (page 9-47)).

   A Component entry for the glue file and a ComponentExtra entry for a readme.txt file are created automatically. You should not remove the glue file entry, but you can delete the readme.txt entry.

3. Click **Add**.

   The Add screen is displayed (Add Screen (page 9-14)).

4. Select an Entry Type. See Manifest File (page 3-9) for more information.

5. In the Sub Directory or File field, enter the location of the files for the selected entry type.

   - For the Component entry type, this setting is the file name for the glue file.
   - For other entry types, enter a path to select all files in a particular sub-directory, or enter a path and file name to select an individual file.
   - The location should be a path relative to the <*install_dir*>/custom/ directory. You can use an absolute path (such as *C:/stellent/custom/my_component/*), but then the component can only be installed on Content Servers with the same installation directory path.

Note: Always use forward slashes in the path.

6. Continue adding entry types and specifying the subdirectories until all of the files of your component are included.

7. Click **OK**.

8. Select **Build—Build**.

   The Build screen is displayed (Main Build Screen (page 9-44)).

9. Click **OK**.

   The Component Wizard builds the component zip file in the <*install_dir*>/custom/*component_name*/ directory.

Working With Installation Parameters

The Install/Uninstall Settings tab is used to create customized installation components that can include preference data parameters. These parameters can be user prompts and

messages. Depending on how they are defined, the prompts and messages are displayed during the installation processes. These custom installation parameters allow the component author to ask for information from users before the component is installed.

To define custom installation parameters for a component:

1. In the Component List screen (Component List Screen (page 9-2)), select the component that will have custom installation parameters defined.
2. Click Open.
3. Select the Install/Uninstall Settings tab (Add/Edit Preference Screen (page 9-40)) and select the appropriate check boxes:
   - ❖ Has Install/Uninstall Filter
   - ❖ Has Install Strings

   Generally, both options will be used to create the desired installation parameters.
4. Click the **Launch Editor** for the Install/Uninstall Filter option to open a Java code template file. Edit the existing code and include additional Java code to the template as necessary to create the filter procedures.

   Each filter procedure will run once during the component installation and/or uninstall procedure. The values of user responses are saved in the installation configuration (install.cfg and config.cfg) files. See Custom Installation Parameter Files (page 3-12) for details.
5. Save and close the Install/Uninstall Filter Java code file.
6. Click the **Add** button on the Preference Data Setup pane to open the Add Preference screen (Add/Edit Preference Screen (page 9-40)).
7. Click the **Launch Editor** for the Install Strings option to open a Java code template file. Edit the existing code and include additional Java code to the template as necessary to define the set up prompts or messages.

   Keep both the Add Preference screen and the Install Strings HTML template open to use simultaneously. Complete the fields on the Add Preference screen as necessary. Add the actual message or prompt text to the Install Strings HTML.
8. Save and close the Install Strings Java code file.
9. Open the Build Settings screen (Build Settings Screen (page 9-47)) by selecting **Build—Build Settings**.
10. Complete the fields on the Build Settings screen as necessary.

11. If components have been specified to be included in the component zip file, they will need to be added as component extras using the Add screen (Add Screen (page 9-14)).

    Click the **Add** button to open the Add screen. Add each component individually.

12. Click **OK**.
13. If necessary, add more components to the zip file as component extras.
14. On the Build Settings screen, click **OK** to create the component zip file.

The zip file can be shipped to clients and can be installed using either the Component Wizard or the Component Manager within the Content Server.

Enabling and Disabling a Component

Use one of the following procedures to enable or disable a component from the Component Wizard:

Tech Tip: Components can also be enabled and disabled using the Component Manager. See Enabling and Disabling a Component (page 6-3).

Option 1

1. Open the component in the Component Wizard.
2. From the The Component Wizard Main Screen (page 9-3), select **Options—Enable** or **Options—Disable**.
3. Restart the Content Server.

   The component is now enabled or disabled.

4. Navigate to the pages affected by the component to ensure that the addition or removal of the customizations is working as you expected.

Option 2

1. Use either of the following methods to display the Component List Screen (page 9-2):
   - Start the Component Wizard.
   - From the The Component Wizard Main Screen (page 9-3), select **Options—Open**.
2. Select the component to be enabled or disabled.
3. Click **Enable** or **Disable**.

4. Restart the Content Server.

   The component is now enabled or disabled.

5. Navigate to the pages affected by the component to ensure that the addition or removal of the customizations is working as you expected.

Removing a Component

Use the following procedure to remove a component from the Content Server:

Note: Removing a component means that the Content Server no longer recognizes the component, but the component files are not deleted from the file system.

1. Disable the component you want to remove.
2. If the component to be removed is open in the Component Wizard, open a different component or close and restart the Component Wizard. (A component cannot be removed if it is open.)
3. Use either of the following methods to display the Component List Screen (page 9-2):
   - Launch the Component Wizard.
   - From the The Component Wizard Main Screen (page 9-3), select **Options—Open**.
4. Select the component to be removed.
5. Click **Remove**.

   A confirmation screen is displayed.

6. Click **Yes**.

   The component no longer appears in the Component List.

Opening a Component

Use the following procedure to open a component that has already been added to the Content Server:

1. Use either of the following methods to display the Component List Screen (page 9-2):
   - Start the Component Wizard. See The Component Wizard Main Screen (page 9-3).

- From the The Component Wizard Main Screen (page 9-3), select **Options—Open**.

2. Select the component to be opened.
3. Click **Open**.

   The component resources are shown in the Custom Resource Definition list on the Component Wizard main screen.

Configuring the Default HTML Editor

You can edit text-based component files directly from the Component Wizard by launching the HTML editor.

- For Windows, Microsoft WordPad (*wordpad.exe*) is the default.
- For UNIX, vi is the default.

Important: Specify a text editor (such as WordPad) rather than a graphical HTML editor (such as FrontPage). Graphical editors can insert or change HTML tags and might cause Idoc Script tags to be converted into a string of characters that will not be recognized by the Content Server.

Use the following procedure to define the default HTML editor:

1. Display the The Component Wizard Main Screen (page 9-3).
2. Select **Options—Configuration**.

   The Component Configuration Screen (page 9-8) is displayed.
3. Click **Browse**.
4. Navigate to and select the executable file for the HTML editor you want to use.
5. Click **Open**.
6. Click **OK**.

   When you click any Launch Editor button in the Component Wizard, the file will open in the selected program.

Unpackaging a Component

Use the following procedure to unpackage a component Zip file:

Note: If you unpackage a component with the same name as an existing component on the Content Server, the older component will be zipped and copied to the <*install_dir*>/bin/ directory, with a filename beginning with "backup" and ending with a time stamp (such as *backup1008968718221.zip*).

1. Use either of the following methods to display the Unpackage Screen (page 9-8):
   - From the The Component Wizard Main Screen (page 9-3), select **Options—Install**.
   - From the Component List Screen (page 9-2), click **Install**.
2. Click **Select**.

   The Zip File Path screen is displayed.
3. Navigate to and select the component zip file.
4. Click **Open**.

   The contents of the component zip file are listed on the Unpackage screen.
5. Click **OK**.

   The component files are copied to the correct locations (there might be a short delay while the files are unzipped), the Unpackage screen closes, and the component resources are shown in the Custom Resource Definition list on the Component Wizard main screen. The component is also added to the Component List.

Note: Unpackaging a component does not enable it. See Enabling and Disabling a Component (page 5-18).

Adding an Existing Component

Use the following procedure to add an existing **unpackaged** component to the Content Server:

1. Use either of the following methods to display the Add Component Screen (page 9-6):
   - From the The Component Wizard Main Screen (page 9-3), select **Options—Add**.
   - From the Component List Screen (page 9-2), click **Add**.
2. Select the **Use Existing Component** option.
3. Click **Browse**.
4. Navigate to and select the component definition (hda) file (components.hda).

5. Click **Open**.

   The path and file name are displayed in the FilePath field.

6. Click **OK**.

   The component resources are shown in the Custom Resource Definition list on the Component Wizard main screen. The component is also added to the Component List.

Note: Adding an existing component does not enable it. See Enabling and Disabling a Component (page 5-18).

Chapter

6

USING THE COMPONENT MANAGER

This chapter discusses using the Component Manager to upload, install, and enable components. It has the following sections:

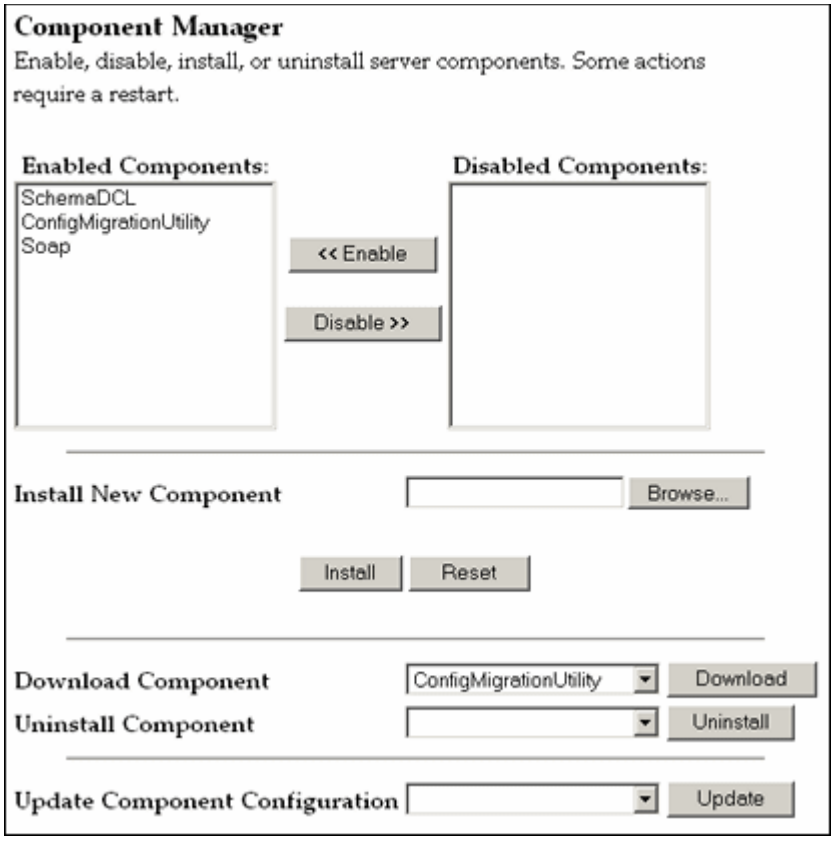
- Component Manager Main Page (page 6-1)
- Component Manager Tasks (page 6-3)

COMPONENT MANAGER MAIN PAGE

The Component Manager page is used to enable, disable, upload, and download components. Use the following procedure to access the Component Manager page of the Admin Server.

1. Log into the Content Server as the system administrator.
2. Click the **Administration** link in the portal navigation bar.
3. Scroll down, and click the **Admin Server** link.
4. Click the <*instance_name*> button to display the options for the Content Server.
5. Click the **Component Manager** link in the left sidebar.

Figure 6-9 Component Manager screen



Feature	Description
Enabled Components list	Shows the components that are currently enabled. Components will be loaded in the order shown in this list.
Disabled Components list	Shows the components that are installed but currently disabled.
Enable button	Moves the selected component from the Disabled Components list to the Enabled Components list.
Disable button	Moves the selected component from the Enabled Components list to the Disabled Components list.
Install New Component field	Enter the path of the component Zip file to be installed to the Content Server or use the corresponding Browse button.

Feature	Description
Browse button	Used to navigate to and select an existing component Zip file.
Install button	Installs the component Zip file specified in the Install New Component field.
Reset button	Clears the Install New Component field.
Download Component field	Select a component to be downloaded to a component Zip file.
Download button	Displays a File Download screen, which is used to save the selected component as a component Zip file.
Uninstall Component field	Select a component to be uninstalled from the Content Server. The listed components are derived from the Disabled Components list.
Uninstall button	Uninstalls the component listed in the Uninstall Component field.
Update Component Configuration field	Select the component to update component configuration parameters. The listed parameters are those that are defined as being editable after the component is installed. This does not require a Content Server restart
Update button	Displays the Update Component Configuration page

COMPONENT MANAGER TASKS

This section describes the following tasks you can perform with the Component Manager:

- Enabling and Disabling a Component (page 6-3)
- Uploading a Component (page 6-4)
- Downloading a Component (page 6-5)

Enabling and Disabling a Component

Use the following procedure to enable or disable a component from the Component Manager:

1. Display the Component Manager page.

2. Select the component to enable or disable.
3. Click the **Enable** or **Disable** button.

   The selected component moves to the other list, and a reminder to restart the Content Server is displayed.
4. Click the **Start/Stop Content Server** link in the left sidebar.

   The Specific Instance Page is displayed.
5. Click the **Restart** icon.

   The Content Server restarts, and the component is now enabled or disabled.
6. Navigate to the pages affected by the component to ensure that the addition or removal of the customizations is working as you expected.

Note: When the Content Server is started, enabled components are loaded in the order shown in the Enabled Components list.

Uploading a Component

Use the following procedure to upload a component zip file using the Component Manager:

Tech Tip: Components can also be uploaded (unpackaged) using the Component Wizard. See Chapter 5 (*Using the Component Wizard)* for details.

1. Display the Component Manager page.
2. Click the **Browse** button next to the Upload New Component field.
3. Navigate to and select the component zip file.
4. Click **Open**.

   The path and file name appears in the Upload New Component field.
5. Click **Upload**.

   The component files are unpackaged on the Content Server, and the name of the component appears in the Disabled Components list.

Note: Uploading a component does not enable it. See Enabling and Disabling a Component (page 6-3) for details.

6. If you are having difficulty uploading the component, check the Content Server output messages by clicking the **View Server Output** link in the left sidebar. The Content Server Output Page is displayed where you can verify the recent actions.

Downloading a Component

Use the following procedure to package a component as a component zip file:

1. Display the Component Manager page.
2. Select the component to be packaged from the **Download Component** list.
3. Click **Download**.

   The File Download screen is displayed.
4. Select the **Save this file to disk** option and click **OK**.

   The Save As screen is displayed.
5. Navigate to the directory where you want to save the component zip file.
6. Change the name of the component zip file as necessary.
7. Click **Save**.

   The component is saved as a component zip file.

Chapter

7

COMPONENT FILE DETAILS

This chapter discusses the HDA file type and the Component Definition (glue) file in more details. The following topics are discussed:

- The components.hda file (page 7-1)
- Component Definition (Glue) File (page 7-3)

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

You may want to refer to the files that were created during the tutorial (Creating a Custom Component (page 5-3)) as you read through this information.

THE COMPONENTS.HDA FILE

The components.hda file tells the Content Server which components are enabled and where to find the component definition ("glue") file for each component. It is always stored in the <*install_dir*>/config/ directory.

The file always includes a ResultSet called *Components* that defines the name and file path of each glue file. You can use the Component Wizard or the Component Manager to make changes to the components HDA file. See Enabling and Disabling a Component (page 2-4) for more information.

In the following example of a components.hda file, two components called "My Component" and "CustomHelp" are enabled.

Figure 7-10 Component.hda with enabled components

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
blFieldTypes=
@end
@ResultSet Components
2
name
location
MyComponent
custom/MultiCheckin/my_component.hda
CustomHelp
custom/customhelp/customhelp.hda
@end
```

Components ResultSet

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

A *Components* ResultSet has two columns:

- The *name* column provides a descriptive name for each component, which is used in the Component Wizard, Component Manager, and Content Server error messages.
- The *location* column defines the location of the glue file for each component. The location can be an absolute path or can be a path relative to the Content Server install directory.

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

COMPONENT DEFINITION (GLUE) FILE

A component definition file, or *glue file*, points to the custom resources that you have defined. The glue file for a component is named *component_name*.hda, and is typically located in the <*install_dir*>/custom/*component_name*/ directory. The Component Wizard can be used to create and make changes to a glue file.

A glue file includes a ResourceDefinition ResultSet (page 7-5), and may contain a MergeRules ResultSet (page 7-6), a Filters ResultSet (page 7-8), and/or a ClassAliases ResultSet (page 7-8).

The following example shows a typical component glue file.

Figure 7-11 Typical component glue file

```
@ResultSet ResourceDefinition
4
type
filename
tables
loadOrder
template
dcl_templates.hda
DCLCustomTemplates
1
resource
dcl_resource.htm
null
1
resource
dcl_upper_clmns_map.htm
DCLColumnTranslationTable
1
resource
dcl_data_sources.htm
dclDataSources
1
service
dcl_services.htm
CustomServices
1
query
dcl_query.htm
CustomQueryTable
1
resource
dcl_checkin_tables.hda
null
1
@end

@ResultSet MergeRules
3
fromTable
toTable
column
DCLCustomTemplates
IntradocTemplates
name
DCLColumnTranslationTable
ColumnTranslation
alias
DCLDataSources
DataSources
name
CustomDCLServiceQueries
ListBoxServiceQueries
dataSource
@end

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

ResourceDefinition ResultSet

The ResourceDefinition ResultSet table defines the type, file name, table names, and load order of custom resources. The following example shows the structure of a *ResourceDefinition* ResultSet.

Figure 7-12 ResultSet structure

```
@ResultSet ResourceDefinition
4
type
filename
tables
loadOrder
template
dcl_templates.hda
DCLCustomTemplates
1
resource
dcl_resource.htm
null
1
resource
dcl_upper_clmns_map.htm
DCLColumnTranslationTable
1
resource
dcl_data_sources.htm
dclDataSources
1
service
dcl_services.htm
CustomServices
1
query
dcl_query.htm
CustomQueryTable
1
resource
dcl_checkin_tables.hda
null
1
@end
```

ResourceDefinition ResultSet Columns

A ResourceDefinition ResultSet consists of four columns:

- The `type` column defines the resource type, which must be one of the following values:
  - *resource*, which points to an HTML include (HTM), string (HTM), dynamic table (HDA), or static table (HTM) resource file.
  - *environment*, which points to an environment resource (CFG) file.
  - *template*, which points to a template resource (HDA) file.
  - *query*, which points to a query resource (HTM) file.
  - *service*, which points to a service resource (HTM) file.
- The `filename` column defines the path and file name of the custom resource file. This can be an absolute path or a relative path. Relative paths are relative to the <*install_dir*>/custom/*component_name*/ directory.
- The `tables` column defines the ResultSet tables to be loaded from the resource file. ResultSet names are separated with a comma. If the resource file does not include ResultSets, this value is *null*. For example, HTML include resources do not include table definitions, so the value for the *tables* column will always be *null* for an HTML include file.
- The `loadOrder` column defines the order in which the resource is loaded. Resources are loaded in ascending order, starting with resources that have a loadOrder of 1. If more than one resource has the same loadOrder, the resources are loaded in the order they are listed in the ResourceDefinition ResultSet. If there is more than one resource with the same name, the last resource loaded is the one used by the system. Normally, you should set the loadOrder to 1, unless there is a particular reason to always load one resource after the others.

MergeRules ResultSet

The MergeRules ResultSet table identifies new tables that are defined in a custom component, and specifies which existing tables the new data will be loaded into. MergeRules are required for custom template resources but are optional for custom dynamic table and static table resources. MergeRules are not required for custom service, query, HTML include, string, and environment resources.

The following example shows a MergeRules ResultSet.

Figure 7-13 MergeRules ResultSet

```
@ResultSet MergeRules
4
fromTable
toTable
column
loadOrder
DCLCustomTemplates
IntradocTemplates
name
1
DCLColumnTranslationTable
ColumnTranslation
alias
1
DCLDataSources
DataSources
name
1
CustomDCLServiceQueries
ListBoxServiceQueries
dataSource
1
@end
```

MergeRules Columns

A MergeRules ResultSet consists of three columns:

- The `fromTable` column specifies a table that was loaded by a custom resource and contains new data to be merged with the existing data. To properly perform a merge, the `fromTable` must have the same number of columns and the same column names as the `toTable`.
- The `toTable` column specifies the name of the existing table into which the new data will be merged. Usually, the `toTable` value is one of the standard Content Server tables, such as *IntradocTemplates* or *QueryTable*.
- The `column` column is the name of the table column that the Content Server uses to compare and update data.
  - The Content Server compares the values of the specified `column` in the `fromTable` and `toTable`. For each `fromTable` value that is identical to a value already in the `toTable`, the row in the `toTable` is replaced by the row in the `fromTable`. For each

fromTable value that is not identical to a value already in the toTable, a new row is added to the toTable and populated with the data from the row of fromTable.

- The column value will usually be name. Setting this value to *null* defaults to the first column, which is generally a name column.

Filters ResultSet

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

Figure 7-14 Filters ResultSet

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

ClassAliases ResultSet

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

Figure 7-15 ClassAliases ResultSet

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

Chapter

8

RESOURCES DETAIL

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

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

This chapter discusses the following resource categories:

- HTML Include (page 8-2)
- String (page 8-4)
- Dynamic Tables (page 8-8)
- Static Tables (page 8-9)
- Query (page 8-9)
- Service (page 8-12)
- Templates (page 8-23)
- Environment (page 8-32)

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

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

- **Manual editing—**Open the resource file in a text editor and edit the code manually. **This is not recommended**.
- **Component Wizard—**You can add, edit, or remove a resource file from a component using the Component Wizard. The Component Wizard provides code for predefined resources that you can use as a starting point for creating custom resources. You can also open resource files in a text editor from within the Component Wizard. Each resource type described in this section includes step-by-step instructions for using the Component Wizard to create and edit that type of resource.

See Creating a Custom Component (page 5-3) for details about using these resources.

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

HTML INCLUDE

An include is defined within <@`dynamichtml` *name*@> and <`@end@`> tags in an HTM resource file. The include is then called using the syntax <`$include` *name*`$`>.

Includes can contain Idoc Script and valid HTML code, including JavaScript, Java applets, cascading style sheets, and comments. Includes can be defined in the same file as they are called from, or they can be defined in a separate HTM file. Standard HTML includes are defined in the <*install_dir*>/shared/config/resources/std_page.htm file.

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

The Super Tag

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

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

The super tag uses the following syntax:

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

You can use the super tag to refer to a standard include or a custom include. The super tag incorporates the include that was loaded last. You can also specify multiple super tags to call an include that was loaded earlier than the last version.

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

Super Tag Example

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

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

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

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

Editing an HTML Include Resource

Use the following procedure to edit an existing HTML include resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource in the Custom Resource Definition list.
3. If the resource file contains more than one type of resource, select the Includes tab in the right pane.
4. Modify the includes in the Custom HTML Includes list.

- To edit an existing include, select the include and click **Edit**. Modify the code and click **OK**.
- To add an include to the resource file, click **Add**.
- To remove an include, select the include and click **Delete**. Click **Yes** to confirm.

STRING

A string resource defines locale-sensitive text strings that are used in error messages and on Content Server web pages and applets. Strings are resolved by the Content Server each time a web page is assembled, an applet is started, or an error message is displayed.

A string is defined in an HTM file using the following format:

```
<@stringID=Text string@>
```

A string is called from an HTM template file using the following Idoc Script format:

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

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

Standard English strings are defined in the <*install_dir*>/shared/config/resources/lang/ directory. Strings for supported languages are located in subdirectories of the /lang/ directory.

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

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

Escape Sequence	Character
&at;	@
\&lf;	line feed (ASCII 10)
\&cr;	carriage return (ASCII 13)
\&tab;	tab (ASCII 9)
\&eatws;	Eats white space until the next non-white space character.

Escape Sequence	Character
\<	< (less than)
\>	> (greater than)
\&sp;	space (ASCII 32)
\&#xxx;	ASCII character represented by decimal number *xxx*

You can specify strings for more than one language in the same resource file using the language identifier prefix, as long as the languages are all single-byte or all multi-byte. For example:

```
<@myString=Thank you@>
<@es.myString=Gracias@>
<@fr.myString=Merci@>
<@de.myString=Danke@>
```

Caution: Do not specify single-byte strings and multi-byte strings in the same resource file. You should create separate resource files for single-byte and multi-byte strings.

If you are specifying multi-byte strings in your custom string resource, you must change the character set specification on your HTML pages to the appropriate encoding. The easiest way to do this is to change the default character set from “iso-8859-1” to “UTF-8” in the std_html_head_definition_declarations include in the *<install_dir>*/shared/config/resources/std_page.htm file:

```
<@dynamichtml std_html_head_definition_declarations@>
    <$if not noHeadDefinitionDeclarations$>
    <$if PageCharset$><$charset = PageCharset$>
    <$else$><$charset = "iso-8859-1"$><$endif$>
    <$if PageTitle$><$pageTitle = eval(PageTitle)$>
    <$else$><$pageTitle = defaultPageTitle$><$endif$>
    <meta http-equiv="Content-Type" content="text/html;charset=<$charset$>">
    <meta name="GENERATOR" content="Idc Content 4.0">
    <title><$pageTitle$></title>
    <$endif$>
<@end@>
```

String Parameters

Text strings can contain variable parameters, which are specified by placing the parameter argument inside curly braces (for example, `{1}`). When a string is localized, the arguments are passed along with the string ID and the ExecutionContext that contains the locale information. The following table describes the syntax for parameterized strings:

Syntax	Meaning	Examples
{{}	Opening curly brace. (Note that only the opening curly brace needs to be expressed as a literal.)	{{}Text in braces}
{n}	Substitute the *n*th argument.	Content ID {1} not found
{ni}	Substitute the *n*th argument, formatted as an integer.	dID {1i} does not exist
{nd}	Substitute the *n*th argument, formatted as a date.	The release date is {1d}
{nD}	Substitute the *n*th argument, formatted as a date. The argument should be ODBC-formatted.	The release date is {1D}
{nt}	Substitute the *n*th argument, formatted as a date and time.	The release date is {1t}
{nT}	Substitute the *n*th argument, formatted as a date and time. The argument should be ODBC-formatted.	The release date is {1T}
{nfm}	Substitute the *n*th argument, formatted as a float with m decimal places.	The distance is {1f3} miles.
{nk}	Substitute a localized string using the *n*th argument as the string ID.	Unable to find {1k} revision of {2}

Syntax	Meaning	Examples
{nm}	Localize the *n*th argument as if it were a string-stack message. (For example, the argument could include concatenated text strings and localized string IDs.)	Indexing internal error: {1m}
{nl}	Substitute the *n*th argument as a list. The argument must be a list with commas (,) and carets (^) as the separators.	Add-ons: {1l}
{nq}	If the *n*th argument is non-null and non-zero in length, substitute the argument in quotation marks. Otherwise, substitute the string "*syUndefined*".	Content item {1q} was not successfully checked in
{n?text}	If the value of the *n*th argument is not 1, substitute the text.	{1} file{1?s} deleted
{n?text1:text2}	❖ If the value of the *n*th argument is not 1, substitute *text1*. ❖ If the value of the *n*th argument is 1, substitute *text2*.	There {1?are:is} currently {1} active search{1?es}.
{n?text1:text2:text3}	❖ If the value of the *n*th argument is not 1 or 2, substitute *text1*. ❖ If the value of the *n*th argument is 2, substitute *text2*. ❖ If the value of the *n*th argument is 1, substitute *text3*.	Contact {1?their:her:his} supervisor.
 **Note:** The (n?) function can be extended with as many substitution variables as required. The last variable in the list always corresponds to a value of 1.		

Editing a String Resource

Use the following procedure to edit an existing string resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource in the Custom Resource Definition list.
3. If the resource file contains more than one type of resource, select the Strings tab in the right pane.
4. Modify the strings in the Custom Strings list.
   - ❖ To edit an existing string, select the string and click **Edit**. Modify the string text and click **OK**.
   - ❖ To add a string to the resource file, click **Add**.
   - ❖ To remove a string, select the string and click **Delete**. Click **Yes** to confirm.

DYNAMIC TABLES

Dynamic table resources are defined in the HDA file format. See Elements in HDA files (page 3-2) for more information and an example of an HDA ResultSet table. Merge rules are required for a dynamic table resource if data from the custom resource replaces data in an existing table. Merge rules are not required if data from the custom resource is to be placed in a new table.

Editing a Dynamic Table Resource

Use the following procedure to edit an existing dynamic table resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource file in the Custom Resource Definition list.
3. Click **Launch Editor**.
4. Modify the table in the text editor.
5. Save and close the resource file.

   Changes are reflected in the right pane of the Resource Definition tab.

STATIC TABLES

Static tables, HTML includes, and strings can be present in the same HTM file. Merge rules are required for a static table resource if data from the custom resource replaces data in an existing table. Merge rules are not required if data from the custom resource is to be placed in a new table.

Editing a Static Table Resource

Use the following procedure to edit an existing static table resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource file in the Custom Resource Definition list.
3. Click **Launch Editor**.
4. Modify the table in the text editor.
5. Save and close the resource file.

   Changes are reflected in the Resource Tables list.

QUERY

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

The standard Content Server queries are defined in the QueryTable table in the <*install_dir*>/config/shared/resources/query.htm file. You will also find special-purpose queries in the indexer.htm and workflow.htm files that are stored in the <*install_dir*>/config/shared/resources/ directory. Merge rules are not required for a query resource.

A query resource is defined in an HTM file using a ResultSet table with three columns: name, queryStr, and parameters.

- The `name` column defines the name for each query. To override an existing query, use the same name for your custom query. To add a new query, use a unique query name. When naming a new query, identify the type of query by starting the name with one of the following characters:

First Character	Query Type
D	Delete
I	Insert
Q	Select
U	Update

- The `queryStr` column defines the query expression. Query expressions are in standard SQL syntax. If there are any parameter values to pass to the database, their place is held with a question mark (?) as an escape character.
- The `parameters` column defines the parameters that are passed to the query from a service. A request from a web browser calls a service, which in turn calls the query. It is the responsibility of the web browser to provide the values for the query parameters, which are standard HTTP parameters The browser can pass query parameters from the URL or from FORM elements in the web page. For example, the *QdocInfo* query requires the *dID* (revision ID) to be passed as a parameter, so the value is obtained from the service request URL.

The following example shows the standard QdocInfo query as defined in the <*install_dir*>/shared/config/resources/query.htm file. This query obtains the metadata information to display on the DOC_INFO template page, which is the page displayed when a user clicks the information icon on a search results page.

The parameter passed from the web browser URL is the dID, which is the unique identification number for the content item revision. The query expression selects the data for the primary revision from the *Revisions*, *Documents*, and *DocMeta* database tables that matches the dID, as long as the revision does not have “Deleted” status.

Figure 8-16 Standard QDocInfo query

<@table QueryTable@>

Query Definition Table

name	queryStr	parameters
QdocInfo	SELECT Revisions.*, Documents.*, DocMeta.* FROM Revisions, Documents, DocMeta WHERE Revisions.dID=? AND Revisions.dID=Documents.dID AND DocMeta.dID = Documents.dID AND Revisions.dStatus<>'DELETED' AND Documents.dIsPrimary<>0	dID int

<@end@>

```
<HTML>
<HEAD>
<META HTTP-EQUIV='Content-Type' content='text/html; charset=iso-8859-1'>
<TITLE>Query Definition Resources</TITLE>
</HEAD>
<BODY>
<@table QueryTable@>
<table border=1><caption><strong>Query Definition Table</strong></caption>
<tr>
    <td>name</td>
    <td>queryStr</td>
    <td>parameters</td>
</tr>
<tr>
    <td>QdocInfo</td>
    <td>SELECT Revisions.*, Documents.*, DocMeta.*
    FROM Revisions, Documents, DocMeta
    WHERE Revisions.dID=? AND Revisions.dID=Documents.dID AND DocMeta.dID = Documents.dID AND
    Revisions.dStatus<>'DELETED' AND Documents.dIsPrimary<>0</td>
    <td>dID int</td>
</tr>
</table>
<@end@>
</BODY>
</HTML>
```

Editing a Query Resource

Use the following procedure to edit a query resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource in the Custom Resource Definition list.
3. If there is more than one table in the resource, select the query table to edit from the Table Name list.
4. Modify the selected query table.
   - ❖ To add a query to the table, click **Add**.
   - ❖ To edit an existing query, select the query and click **Edit**. Modify the query expression and/or parameters and click **OK**.
   - ❖ To remove a query, select the query and click **Delete**. Click **Yes** to confirm.

SERVICE

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

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

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

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

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

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

The standard Content Server services are defined in the StandardServices table in the <*install_dir*>/config/shared/resources/std_services.htm file. You will also find special-purpose services in the workflow.htm file in the <*install_dir*>/config/shared/resources/ directory.

Services depend on other resource definitions to perform their functions. Any service that returns HTML requires a template to be specified. A common exception is the PING_SERVER service, which does not return a page to the browser.

Most services use a query. A common exception is the SEARCH service, which sends a request directly to the search collection. Merge rules are not required for a service resource.

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

Figure 8-17 Service definition example

<@table StandardServices@>

Scripts For Standard Services

Name	Attributes	Actions
UPDATE_DOCINFO	DocService 2 null null documents !csUnableToUpdateInfo (dDocName)	3:doSubService:UPDATE_DOCINFO_SUB:12:null

<@end@>

A service resource is defined in an HTM file using a ResultSet table with the following three columns:

- The `Name` column defines the name for each service. For client-side service requests, this is the name called in the URL. To override an existing service, use the same name for your custom service. To add a new service, use a unique service name.
- The `Attributes` column defines the following attributes for each service:

Attribute	**Description**	**Example (attributes from the DELETE_DOC service)**
Service class	Determines, in part, what actions can be performed by the service.	**`DocService`** `4 MSG_PAGE null documents !csUnableToDeleteItem(dDocName)`
Access level	Assigns a user access level to the service. This number is the sum of the following possible bit flags: READ_PRIVILEGE = 1 WRITE_PRIVILEGE = 2 DELETE_PRIVILEGE = 4 ADMIN_PRIVILEGE = 8 GLOBAL_PRIVILEGE = 16 SCRIPTABLE_SERVICE=32	`DocService` **`4`** `MSG_PAGE null documents !csUnableToDeleteItem(dDocName)`
Template page	Specifies the template that presents the results of the service. If the results of the service do not require presentation, this attribute is *null*.	`DocService 4` **`MSG_PAGE`** `null documents !csUnableToDeleteItem(dDocName)`

Attribute	Description	Example (attributes from the DELETE_DOC service)
Service type	If the service is to be executed inside another service, this attribute is *SubService*; otherwise, this attribute is *null*.	`DocService 4 MSG_PAGE **null** documents !csUnableToDeleteItem(dDocName)`
Subjects notified	Specifies the subjects (subsystems) to be notified by the service. If no subjects are notified, this attribute is *null*.	`DocService 4 MSG_PAGE null **documents** !csUnableToDeleteItem(dDocName)`
Error message	Defines the error message returned by the service if no action error message overrides it. This can be either an actual text string or a reference to a locale-sensitive string (see *Resolving Localized Strings* in *Customizing* Content Server for more information).	`DocService 4 MSG_PAGE null documents **!csUnableToDeleteItem(dDocName)**`

- The `Actions` column defines the actions for each service. An action is an operation to be performed as part of a service script. Actions can execute an SQL statement, perform a query, run code, cache the results of a query, or load an option list. Each service includes one or more actions, which specify what happens upon execution.

  The `<br>` tags in the `Actions` column are for browser display purposes only, so they are optional. However, the `</td>` tag must occur immediately after the actions, without a line break in between. An action is defined using the following format:

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

Section	Description	Example (first action from the DELETE_DOC service)
Type	Defines the type of action: QUERY_TYPE = 1 EXECUTE_TYPE = 2 CODE_TYPE = 3 OPTION_TYPE = 4 CACHE_RESULT_TYPE = 5	**5**:QdocInfo:DOC_INFO:6: !csUnableToDeleteItem(dDocName)!csRevision NoLongerExists
Name	Specifies the name of the action.	5:**QdocInfo**:DOC_INFO:6: !csUnableToDeleteItem(dDocName)!csRevision NoLongerExist
Parameters	Specifies parameters required by the action. If no parameters are required, leave this part empty (two colons will appear in a row).	5:QdocInfo:**DOC_INFO**:6: !csUnableToDeleteItem(dDocName)!csRevision NoLongerExist
Control mask	Controls the results of queries to the database. This number is the sum of the following possible bit flags: No control mask = 0 CONTROL_IGNORE_ERROR = 1 CONTROL_MUST_EXIST = 2 CONTROL_BEGIN_TRAN = 4 CONTROL_COMMIT_TRAN = 8 CONTROL_MUST_NOT_EXIST = 16	5:QdocInfo:DOC_INFO:**6**: !csUnableToDeleteItem(dDocName)!csRevision NoLongerExist
Error message	Defines the error message to be displayed by this action. This error message overrides the error message provided as an attribute of the service. This can be either an actual text string or a reference to a locale-sensitive string (see *Resolving Localized Strings* in *Customizing Content Server* for more information).	5:QdocInfo:DOC_INFO:6: **!csUnableToDeleteItem(dDocName)!csRevisionNoLongerExist**

Service Example

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

Figure 8-18 DOC_INFO service

```
<HTML>
<HEAD>
<META HTTP-EQUIV='Content-Type' content='text/html; charset=iso-8859-1'>
<TITLE>Standard Scripted Services</TITLE>
</HEAD>
<BODY>
<@table StandardServices@>
<table border=1><caption><strong>Scripts For Standard Services</strong></caption>
<tr>
    <td>Name</td><td>Attributes</td><td>Actions</td>
</tr>
<tr>
    <td>DOC_INFO</td>
    <td>DocService
        33
        DOC_INFO
        null
        null<br>
        !csUnableToGetRevInfo</td>
    <td>5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2
        3:mapNamedResultSetValues:DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null
        3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName)
        3:getDocFormats:QdocFormats:0:null
        3:getURLAbsolute::0:null
        3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null
        3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null
        3:getWorkflowInfo:WF_INFO:0:null
        3:getDocSubscriptionInfo:QisSubscribed:0:null
        5:QrevHistory:REVISION_HISTORY:0:!csUnableToGetRevHistory(dDocName)</td>
</tr>
</table>
<@end@>
</BODY>
</HTML>
```

<@table StandardServices@>

Scripts For Standard Services

Name	Attributes	Actions
DOC_INFO	DocService 33 DOC_INFO null null ! csUnableToGetRevInfo	5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2 3:mapNamedResultSetValues:DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null 3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName) 3:getDocFormats:QdocFormats:0:null 3:getURLAbsolute::0:null 3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null 3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null 3:getWorkflowInfo:WF_INFO:0:null 3:getDocSubscriptionInfo:QisSubscribed:0:null 5:QrevHistory:REVISION_HISTORY:0:! csUnableToGetRevHistory(dDocName)

<@end@>

Attributes

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

Attribute	Value	Description
Service class	DocService	This service is providing information about a content item.
Access level	33	32 = This service can be executed with the *executeService* Idoc Script function. 1 = The user requesting the service must have Read privilege on the content item.
Template page	DOC_INFO	This service uses the DOC_INFO template (*doc_info.htm* file). The results from the actions will be merged with this template and presented to the user.
Service type	null	This service is not a subservice.
Subjects notified	null	No subjects are affected by this service.
Error Message	!csUnableToGetRevInfo	If this service fails on an English Content Server system, it returns the error message string: *Unable to retrieve information about the revision*

Actions

The DOC_INFO service executes the following actions:

Action Definition	Description
Action 1 `5:QdocInfo:DOC_INFO:2:!csItemNoLongerExists2`	
5	Cached query action that retrieves information from the database using a query.
QDocInfo	This action retrieves content item information using the *QDocInfo* query in the *query.htm* file.
DOC_INFO	The result of the query is assigned to the parameter DOC_INFO and stored for later use.
2	The CONTROL_MUST_EXIST control mask specifies that the query must return a record, or the action fails.
!csItemNoLongerExists2	If this action fails on an English Content Server system, it returns the error message string: *This content item no longer exists*
Action 2 `3:mapNamedResultSetValues:DOC_INFO,dStatus,dStatus,dDocTitle,dDocTitle:0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
mapNamedResultSetValues	This action retrieves the values of *dStatus* and *dDocTitle* from the first row of the DOC_INFO ResultSet and stores them in the local data. (This increases speed and ensures that the correct values are used.)
DOC_INFO,dStatus,dStatus, dDocTitle,dDocTitle	Parameters required for the *mapNamedResultSetValues* action.
0	No control mask is specified.
null	No error message is specified.

Action Definition	Description
Action 3 `3:checkSecurity:DOC_INFO:0:!csUnableToGetRevInfo2(dDocName)`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
checkSecurity	This action retrieves the data assigned to the DOC_INFO parameter and evaluates the assigned security level to verify that the user is authorized to perform this action.
DOC_INFO	Parameter that contains the security information to be evaluated by the *checkSecurity* action.
0	No control mask is specified.
!csUnableToGetRevInfo2(dDocName)	If this action fails on an English Content Server system, it returns the error message string: *Unable to retrieve information for "{dDocName}."*
Action 4 `3:getDocFormats:QdocFormats:0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
getDocFormats	This action retrieves the file formats for the content item using the *QdocFormats* query in the *query.htm* file. A comma-separated list of the file formats is stored in the local data as *dDocFormats*.
QdocFormats	Specifies the query used to retrieve the file formats.
0	No control mask is specified.
null	No error message is specified.

Action Definition	Description
Action 5 `3:getURLAbsolute::0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
getURLAbsolute	This action resolves the URL of the content item and stores it in the local data as *DocUrl*.
blank	This action takes no parameters.
0	No control mask is specified.
null	No error message is specified.
Action 6 `3:getUserMailAddress:dDocAuthor,AuthorAddress:0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
getUserMailAddress	This action resolves the e-mail address of the content item author.
dDocAuthor,AuthorAddress	This action passes *dDocAuthor* and *AuthorAddress* as parameters.
0	No control mask is specified.
null	No error message is specified.

Action Definition	Description
Action 7 `3:getUserMailAddress:dCheckoutUser,CheckoutUserAddress:0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
getUserMailAddress	This action resolves the e-mail address of the user who has the content item checked out.
dCheckoutUser, CheckoutUserAddress	This action passes *dCheckoutUser* and *CheckoutUserAddress* as parameters.
0	No control mask is specified.
null	No error message is specified.
Action 8 `3:getWorkflowInfo:WF_INFO:0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
getWorkflowInfo	This action evaluates whether the content item is part of a workflow. If the WF_INFO ResultSet exists, then workflow information is merged into the DOC_INFO template.
WF_INFO	This action passes WF_INFO as a parameter.
0	No control mask is specified.
null	No error message is specified.

Action Definition	Description
Action 9 `3:getDocSubscriptionInfo:QisSubscribed:0:null`	
3	Java method action specifying a module that is a part of the Java class implementing the service.
getDocSubscriptionInfo	This action evaluates if the current user has subscribed to the content item: ❖ If the user is subscribed, an **Unsubscribe** button is displayed. ❖ If the user is not subscribed, a **Subscribe** button is displayed.
QisSubscribed	Specifies the query used to retrieve the subscription information.
0	No control mask is specified.
null	No error message is specified.
Action 10 `5:QrevHistory:REVISION_HISTORY:0:!csUnableToGetRevHistory(dDocName)`	
5	Cached query action that retrieves information from the database using a query.
QrevHistory	This action retrieves revision history information using the *QrevHistory* query in the *query.htm* file.
REVISION_HISTORY	The result the query is assigned to the parameter REVISION_HISTORY. The DOC_INFO template uses this parameter in a loop to present information about each revision.
0	No control mask is specified.
!csUnableToGetRevHistory (dDocName)	If this action fails on an English Content Server system, it returns the error message string: *Unable to retrieve revision history for "{dDocName}."*

Editing a Service Resource

Use the following procedure to edit a service resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource in the Custom Resource Definition list.
3. If there is more than one table in the resource, select the service table to edit from the Table Name list.
4. Modify the selected service table.
   - ❖ To add a service to the table, click **Add**.
   - ❖ To edit an existing service, select the service and click **Edit**. Modify the service attributes and/or actions and click **OK**.
   - ❖ To remove a service, select the service and click **Delete**. Click **Yes** to confirm.

TEMPLATES

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

The actual template pages (.htm files) are separate files that are referenced in the template resource file. *Template HTM* files contain the code that the Content Server uses to assemble web pages. HTML markup in a template file defines the basic layout of the page, while Idoc Script in a template file generates additional HTML code for the web page at the time of the page request. Because HTM template files contain a large amount of script that is not resolved by the Content Server until the final page is assembled, these files are not viewable web pages.

The template type of HTM file is used to define the following component files:

- ❖ **Template pages—**Standard template pages are located in the <*install_dir*>/shared/config/templates/ directory.
- ❖ **Report pages—**Standard report pages are located in the <*install_dir*>/shared/config/reports/ directory.

A *template resource (templates.hda)* is defined in the HDA file format. See Elements in HDA files (page 3-2) for more information and an example of an HDA ResultSet table. The standard templates are defined in the <*install_dir*>/shared/config/templates/templates.hda file.

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

Figure 8-19 MergeRules ResultSet for a template

```
@ResultSet MergeRules
4
fromTable
toTable
column
loadOrder
MultiCheckinTemplates
IntradocTemplates
name
1
@end
```

The standard templates.hda file defines three ResultSet tables:

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

pages in the Library. Query result pages are a special type of search results page. This table consists of six columns:

- The `name` column defines the name for each template page. This name is how the template is referenced in the Content Server CGI URLs, in code, and in the Web Layout Editor utility.

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

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

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

  where *Text1* contents appear on the first line, and *Text2* contents appear on the second line in each search results row. `<Tab>` represents a literal tab character.

  Idoc Script can be used to define the contents in the `flexdata` field. You can also change the `flexdata` of the *StandardResults* template through the Web Layout Editor, but these changes are saved in a separate file (<*install_dir*>/data/results/custom_results.hda) rather than in the *SearchResultTemplates* table in the *templates.hda* file.
- The `description` column defines a description of the template.

The following example shows a custom template resource file that points to a custom Content Management page (multicheckin_doc_man.htm) and a custom search results page (MultiCheckin_search_results.htm).

Figure 8-20 Custom template resource file

```
<?hda version="5.1.1 (build011203)" jcharset=Cp1252 encoding=iso-8859-1?>
@Properties LocalData
blDateFormat=M/d{/yy} {h:mm[:ss] {aa}[zzz]}!tAmerica/Chicago!mAM,PM
blFieldTypes=
@end
@ResultSet MultiCheckinTemplates
5
name
class
formtype
filename
description
DOC_MANAGEMENT_LINKS
DocManagement
DocManagementLinks
multicheckin_doc_man.htm
Page containing links to various document management functions
@end
@ResultSet MultiCheckin_2
6
name
formtype
filename
outfilename
flexdata
description
StandardResults
SearchResultsPage
MultiCheckin_search_results.htm
null
Text2 <$dDocTitle$> <$dInDate$>%Text1 <$dDocName$>%
apStandardResultsDesc
@end
```

Template and Report Pages

Template pages and report pages are also called “presentation” pages, because the Content Server uses them to assemble, format, and present the results of a web page request.

The standard template pages are located in the <*install_dir*>/shared/config/templates directory. The standard report pages are located in the <*install_dir*>/shared/config/reports directory.

Tech Tip: The <*install_dir*>/samples directory contains a component called Custom Reports, which shows you how to create custom reports. It also provides two custom reports that display the security groups permissions and accounts that users belong to.

Template Page Example

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

Figure 8-21 Template page example

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

<$include body_def$>

<$include std_page_begin$>

<$include std_header$>

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

<$include std_doc_man_pages$>

</table>

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

<$include std_page_end$>

</body>
</html>
```

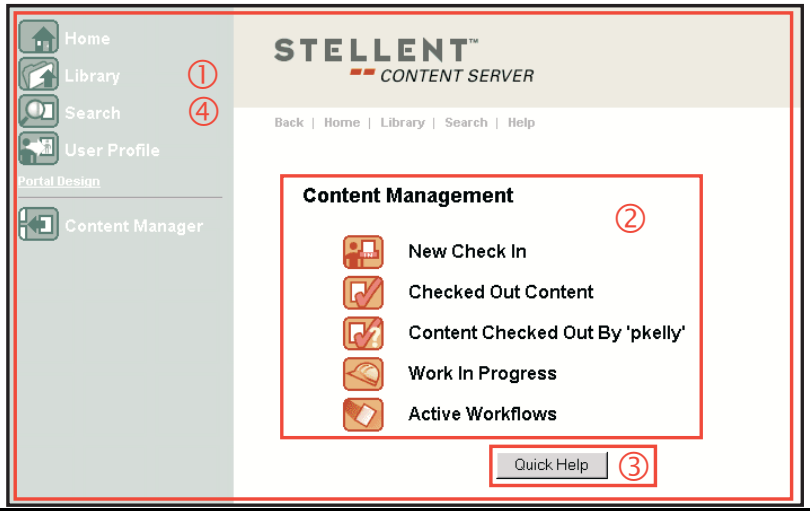
The page title is referenced here, and the code for the head section is built using the *std_html_head_declarations* include code from the *std_page.htm* resource file.

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

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

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

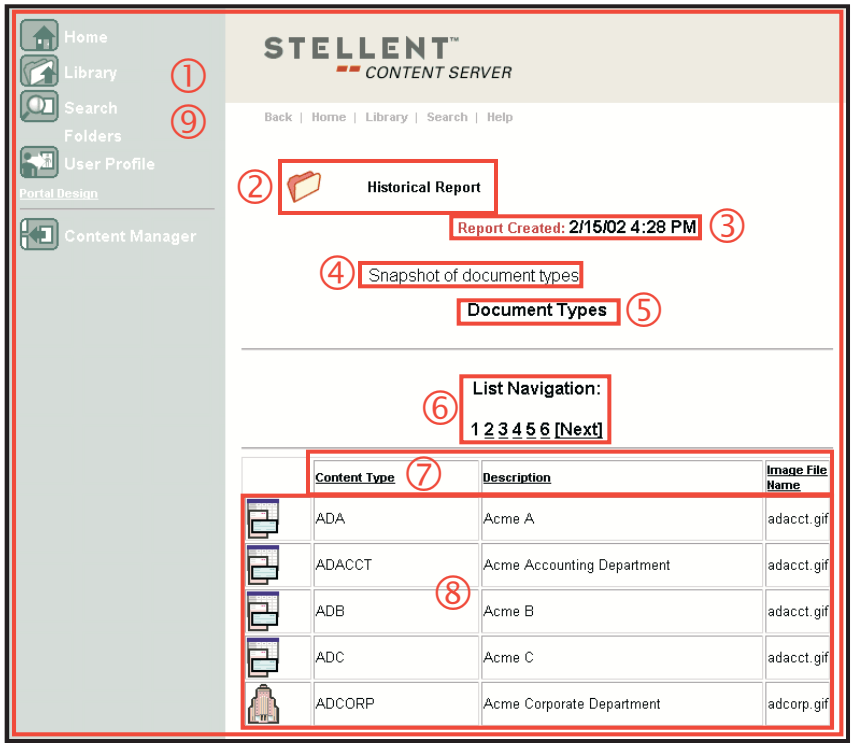
④ The code at the end of the page is built using the *std_page_end* include code from the *std_page.htm* resource file.



Report Page Example

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

Figure 8-22 Report page example



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

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

<$include body_def$>

<$include std_page_begin$>

<$include std_header$>

<!--Directory Title--->
<table border="0" cellpadding="0" cellspacing="0">
<tr>
        <td width="75"><$if PageParent$><a href="<$PageParent$>">
                <$strTrimWs(inc("open_folder_image"))$></a>
                <$endif$></td>
        <td colspan="2" width="390"><span class=title><$PageTitle$></span></td>
</tr>
</table>

<$if IsSavedQuery$>
<!---Parameters for historical reports-->
<table border="0" cellpadding="0" cellspacing="0">
<tr>
        <td width="75" height="45"> </td><!---Indent-->
        <td><strong><span class=highlightField><$lc("wwReportCreated")$></span> <$ReportCreationDate$></strong></td>
</tr>
</table>
<$endif$>

<!--Directory Header--->
<table  border="0" cellspacing="0">
<tr>
        <td width="75" height="45"> </td><!---Indent-->
        <td colspan="2" width="390"><$HeaderText$></td>
</tr>
</table>

<!---Doc types report-->
<table border=0 cellpadding=0 cellspacing=0>
<tr>
        <td width=565 align=center><h4 class=pageTitle><$lc("wwDocumentTypes")$></h4>
        </td>
</tr>

<$if IsMultiPage$>
<!---Navigation Bar-->
<tr>
        <td width=565 align="center"><$include std_page_nav_bar$></td>
</tr>
<$endif$>

<tr>
        <td>
        <table border=1 width=<$StdPageWidth$>>
                <tr>
                        <td width=12%></td>
                        <td width=29%><span class=reportColumnSmall><u><$lc("wwDocumentType")$></u>
                        </span></td>
                        <td width=49%><span class=reportColumnSmall><u><$lc("wwDescription")$></u>
                        </span></td>
                        <td width=12%><span class=reportColumnSmall><u><$lc("wwImageFileName")$></u>
                        </span></td>
                </tr>
                <$loop DocTypes$>
                <tr>
                        <!--Document types are localized to each instance, so we must use direct path to images directory.-->
                        <td width=12%><img src="<$HttpWebRoot$>images/docgifs/<$dGif$>" border=0></td>
                        <td width=29%><span class=tableEntry><$dDocType$></span></td>
                        <td width=49%><span class=tableEntry><$dDescription$></span></td>
                        <td width=12%><span class=tableEntry><$dGif$></span></td>
                </tr>
                <$endloop$>
                </table>
        </td>
</tr>
</table>

<$include std_page_end$>

</body>
</html>
```

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

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

② Displays the open folder image, links it to the parent page, and displays the page title.

③ Displays the original query date for a historical report.

④ Displays the report description.

⑤ Displays the table title.

⑥ If a historical report requires more than one page, this displays the page navigation bar.

⑦ Displays the table column headers.

⑧ Loops on the document types to create the rows of the report table.

⑨ The code at the end of the page is built using the *std_page_end* include code from the *std_page.htm* resource file.

Editing a Template Resource

Use the following procedure to edit an existing template resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource in the Custom Resource Definition list.
3. To remove a template definition table or edit a template definition manually, click **Launch Editor** in the Custom Resource Definition pane.
4. If there is more than one table in the resource, select the template table to edit from the Table Name list.
5. Modify the selected template table.
   - ❖ To add a template definition to the table, click **Add**.
   - ❖ To edit an existing template definition, select the definition and click **Edit**. Modify the parameters and click **OK**.
   - ❖ To remove a template definition, select the definition and click **Delete**. Click **Yes** to confirm.

ENVIRONMENT

An environment resource defines configuration variables, either by creating new variable values or replacing existing values. Because custom resources are loaded after the standard *config.cfg* file is loaded, the variable values defined in the custom environment resource replace the original variable values.

An environment resource is defined in a CFG file using a name/value pair format:

```
variable_name=value
```

After defining a variable value, you can reference the variable in templates and other resource files with the following Idoc Script tag:

```
<$variable_name$>
```

Environment resource files can include comment lines, which are designated with a # symbol:

```
#Set this variable to true to enable the function.
```

Environment Example

The following is an example of an environment resource file.

Figure 8-23 Environment resource file

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

# These are the nested search field definitions.

nsFld1Caption=Document Text
nsFld1Name=
nsFld1Type=FullText
nsFld1OptionKey=
#
nsFld2Caption=Text
nsFld2Name=xtext
nsFld2Type=Text
nsFld2OptionKey=
#
nsFld3Caption=Date
nsFld3Name=xdate
nsFld3Type=Date
nsFld3OptionKey=
#
nsFld4Caption=Integer
nsFld4Name=xinteger
nsFld4Type=Int
nsFld4OptionKey=
#
nsFld5Caption=Option List
nsFld5Name=xoptionlist
nsFld5Type=OptionList
nsFld5OptionKey=optionlistList
#
nsFld6Caption=Info Topic
nsFld6Name=xwfsInfoTopic
nsFld6Type=OptionList
nsFld6OptionKey=wfsInfoTopicList
```

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

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

Standard configuration variables are defined in the <*install_dir*>/config/config.cfg file. See the *Idoc Script Reference Guide* for a complete list of configuration variables.

Editing an Environment Resource

Use the following procedure to edit an existing environment resource using the Component Wizard.

1. In the Component Wizard, open the component that contains the resource to edit.
2. Select the resource file in the Custom Resource Definition list.
3. Click **Launch Editor**.
4. Modify the configuration variables in the text editor.
5. Save and close the resource file.

   Changes are reflected in the Custom Environment Parameters list.

Note: The configuration settings might not appear in the Custom Environment Parameters list in the order they actually appear in the resource file. It is recommended that you launch the text editor for easier viewing.

Chapter

9

COMPONENT INTERFACE SCREENS

This section contains screen representations and explanations of the options used to create custom components. It covers these topics:

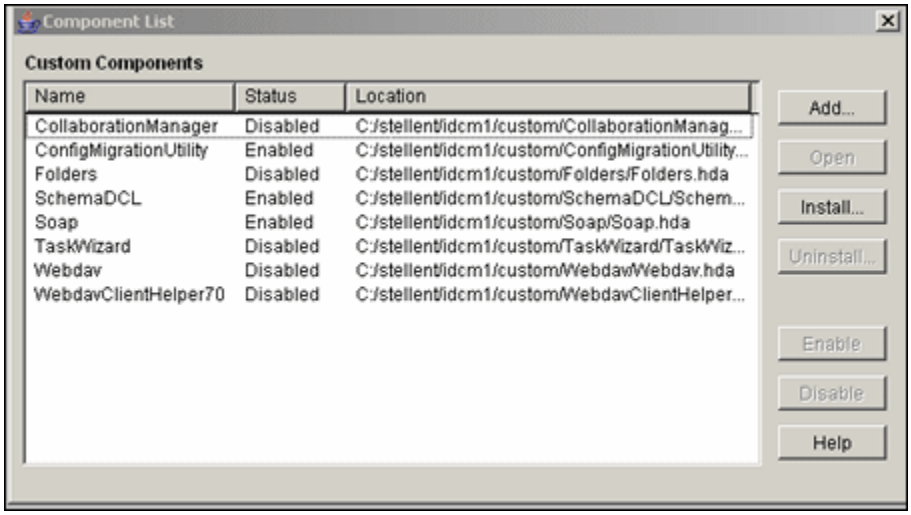
- Component List Screen (page 9-2)
- The Component Wizard Main Screen (page 9-3)
- Component Creation Screens (page 9-6)
- Build Screens (page 9-41)

To display the Component Wizard:

- **Windows:** Select **Start—Programs—Content Server—<*instance_name*>—Tools—Component Wizard**.
- **Unix:** Change to the <*install_dir*>/bin/ directory and run the *ComponentWizard* program.

The Component List Screen (page 9-2) and the The Component Wizard Main Screen (page 9-3) are displayed.

COMPONENT LIST SCREEN



The Component List appears when you first access the Component Wizard. It lists all currently installed components.

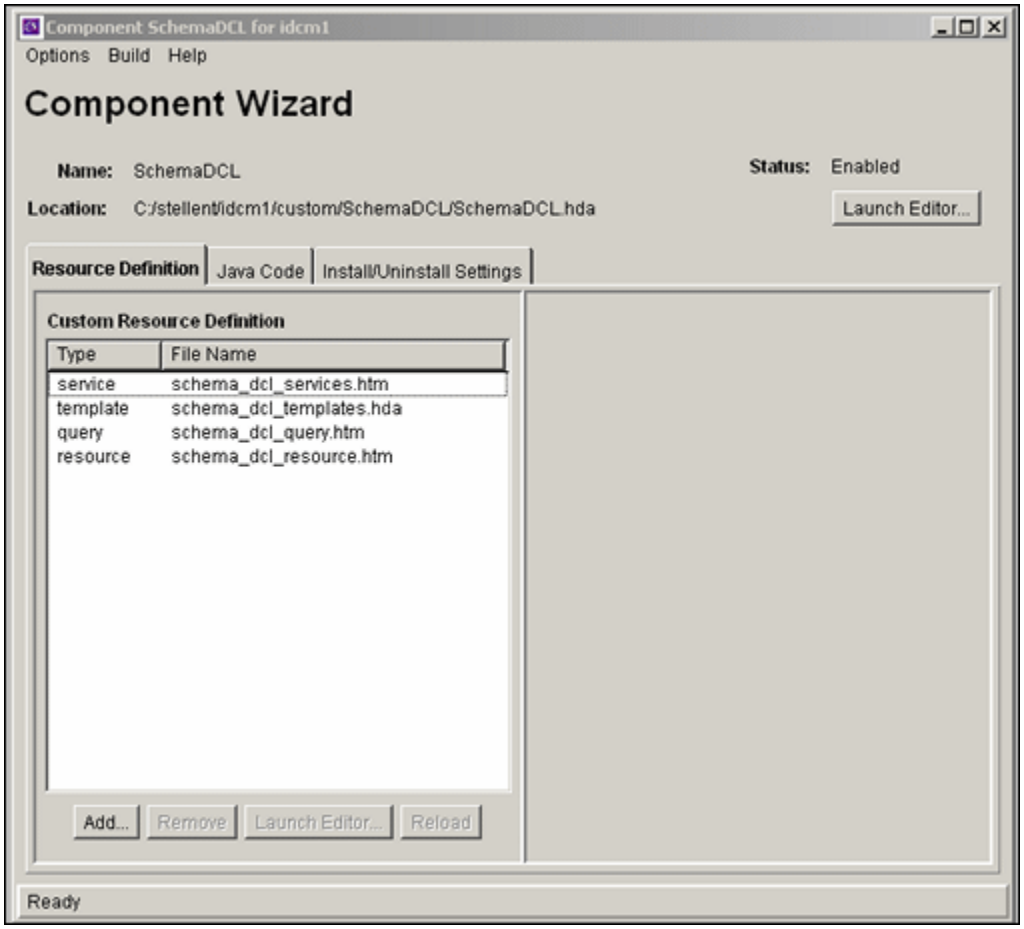
To access this screen, do one of the following:

- Start the Component Wizard.
- Select **Options—Open** from the The Component Wizard Main Screen (page 9-3).

Feature	Function
Custom Components list	Shows the name, status (enabled or disabled), and location of the component definition file for each component that has been installed in the Content Server.
Add button	Displays the Add Component Screen (page 9-6).
Open button	Opens the selected component in the Component Wizard.
Install button	Displays the Unpackage Screen (page 9-8).
Uninstall button	Removes the selected component from the Content Server. (The component files remain in the file system, but the component no longer appears on the list of components.)

Feature	Function
Enable button	Enables the selected component.
Disable button	Disables the selected component.
Help button	Displays a help page for the Component List screen.

THE COMPONENT WIZARD MAIN SCREEN



Feature	Description
Options Menu (page 9-5)	Provides options for working with components and settings.
Build Menu (page 9-5)	Used to package component files into a Zip file.
Help Menu (page 9-6)	Provides links to online documentation.
Summary fields	Show the name of the component, the location and file name of the component definition file, and the status of the component (enabled or disabled).
Launch Editor button (top right)	Displays the component definition file (“glue” file) in the default text editor.
Custom Resource Definition list	Lists the custom resources that have been defined for the component.
Add button	Displays the Add Resource screen, which is used to add a new resource file to the component.
Remove button	Removes the selected resource from the component.
Launch Editor button (Resource Definition tab)	Displays the selected resource file in the default text editor.
Reload button	Reloads the component definition file for the selected resource.
Custom resource parameters list	Shows the custom parameters for the resource selected in the Custom Resource Definition list. This pane is different for each type of resource.
Java Code tab	Displays any custom Java code that has been defined for the component.

Feature	Description
Install/Uninstall Settings tab	Displays custom installation parameters.

Options Menu

The Options menu provides options for working with components and settings.

Menu Item	Description
Add	Displays the Add Component Screen (page 9-6).
Open	Displays the Component List Screen (page 9-2).
Close	Closes the open component.
Install	Displays the Unpackage Screen (page 9-8).
Enable	Enables the component that is open in the Component Wizard.
Disable	Disables the component that is open in the Component Wizard.
Configuration	Displays the Component Configuration Screen (page 9-8).
Edit Readme File	Displays the *readme.txt* file for the open component in the default text editor. If a *readme.txt* file does not exist for the component, a blank *readme.txt* file is created.
Exit	Closes the Component Wizard.

Build Menu

The Build menu is used to package component files into a zip file.

Menu Item	Description
Build Settings	Displays the Build Settings screen, which is used to build a component Zip file.
Build	Displays the Build screen, which is used to build a component Zip file.

Help Menu

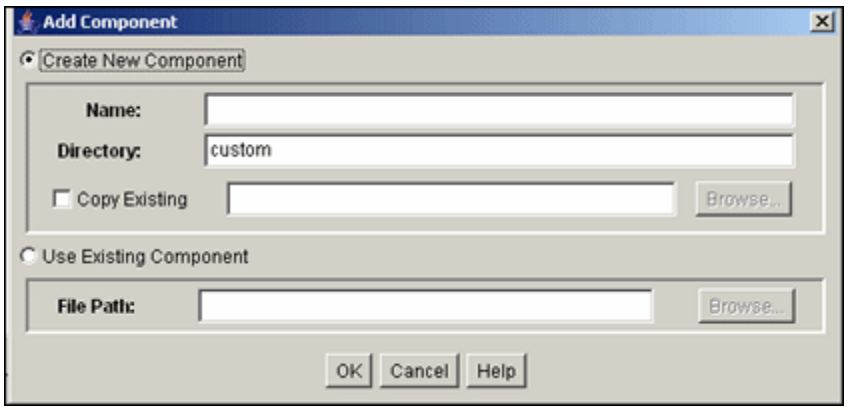
This screen provides links to online documentation.

Menu Item	Description
Contents	Displays the online help for system administrators.
About Content Server	Displays version, build, and copyright information for the Content Server.

COMPONENT CREATION SCREENS

The following screens are all used to build custom components.

Add Component Screen

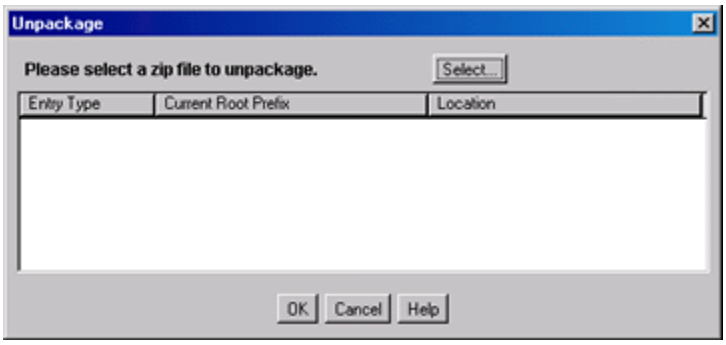


The Add Component screen is used to add a new component to the Content Server. To access this screen, do one of the following:

- Select **Options—Add** from the The Component Wizard Main Screen (page 9-3).
- Click **Add** on the Component List Screen (page 9-2).

Feature	Function
Create New Component option	Select this option to create a new custom component in the Content Server.
Name field	Assign a descriptive component name. The name cannot contain spaces.
Directory field	Enter the directory where the component definition file will be located, relative to Content Server install directory. Typically, components are located in the *custom* directory.
Copy Existing check box	**Selected** = The new component will be a copy of an existing component, including all resources and other component files. Enter the path and file name of an existing component definition file (.hda). The new component must have a unique name. **Clear** = The new component will be created without any resource files.
Browse button	Used to navigate to and select an existing component definition file.
Use Existing Component option	Select this option to add an existing component to the Content Server.
File Path field	The path and file name of the existing component.
Browse button	Used to navigate to and select an existing component definition file.
OK button	Adds the component to the Content Server.
Cancel button	Closes the screen without adding a component.
Help button	Displays a help page for the Add Component screen.

Unpackage Screen

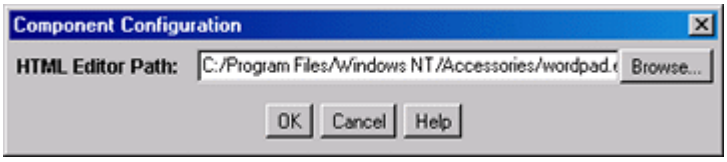


The Unpackage screen is used to install a component zip file on the Content Server. To access this screen, do one of the following:

- Select **Options—Install** from the The Component Wizard Main Screen (page 9-3).
- Click **Install** on the Component List Screen (page 9-2).

Feature	Description
Select button	Used to navigate to and select the Zip file to be unpackaged.
Entry Type column	Lists the items that are included in the component Zip file.
Current Root Prefix column	The root directory where the related component files will be installed.
Location column	The subdirectory and/or the file name of the component file.
OK button	Unpackages the component onto the Content Server.
Cancel button	Closes the screen without unpackaging the component.
Help button	Displays a help page for the Unpackage screen.

Component Configuration Screen

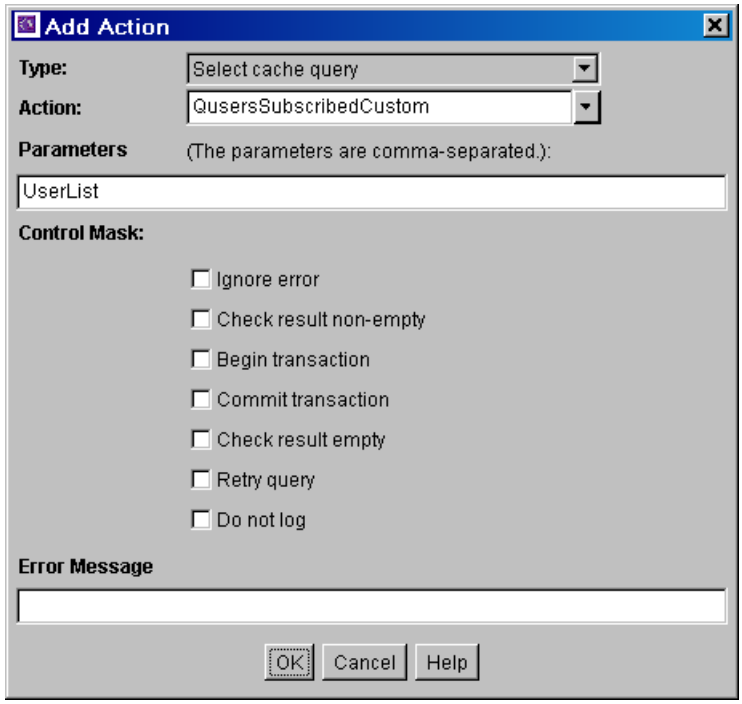


The Component Configuration screen is used to specify which program to use to edit component files from within the Component Wizard. To access this screen, select **Options—Configuration** from the Component Wizard application.

Specify a text editor (such as WordPad) rather than a graphical HTML editor (such as FrontPage). Graphical editors can insert or change HTML tags and might cause Idoc Script tags to be converted into a string of characters that will not be recognized by the Content Server.

Feature	Description
HTML Editor Path field	The path and file name of the executable file for the editing program. For example, *c:/Program Files/Windows NT/accessories/wordpad.exe*.
Browse button	Used to navigate to and select the file.
OK button	Sets the specified file as the editing program.
Cancel button	Closes the screen without changing the editing program.
Help button	Displays a help page for the Component Configuration screen.

Add/Edit Action Screen



This screen is used to specify the actions that are associated with your newly defined component service.

Feature	Description
Type list	Select the type of action. See Predefined Action Types (page 9-11).
Action list	Select an action from the list or enter a custom action. The option list shows the predefined actions that are associated with the option selected from the Type list.

Feature	Description
Parameters field	If the action takes parameters, enter the parameters as a comma-separated list. • For the Select Query and Select Cache Query action types, the first parameter is the name that the action assigns to the ResultSet returned from the query. This ResultSet can then be referenced in the template page. • For the Load Option List action type, the parameters are optional. However, if parameters are given, the first parameter is the key under which the option list is loaded, and the second parameter is the selected value for display on an HTML page.
Control Mask check boxes	The control mask controls the results of queries to the database.
Error Message field	Enter the error message to be displayed by this action. This action error message overrides the error message provided as an attribute of the service. • If the action error message is not empty, it becomes the active error message. • If the action error message is empty, the error message remains unchanged from the previous action.
OK button	Saves the action in the Actions list on the Add/Edit Service screen.
Cancel button	Closes the Add/Edit Action screen without creating a service action.
Help button	Displays this help page for the Add/Edit Action screen.

Predefined Action Types

Note: See the query.htm, workflow.htm, and indexer.htm files in the <*install_dir*>/shared/config/resources/ directory for more information on predefined queries.

The following action types can be specified for a service:

- Select Query: This action type is used to select a query and then discard it immediately.
- Execute Query: This action type is used to execute a query.
- Java Method: This action type is used to apply a method that is part of the Java class implementing the service. The following Java Method actions can be selected from the Action list:

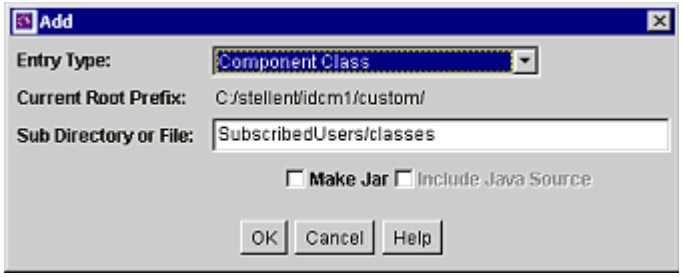
Name	Description
checkSecurity	This method is used for validating security for actions on a particular document, such as check in, check out, and delete. It checks the logged in user's security group and account permissions against the service's access level for performing the specified action. Takes zero or one parameter, which is the name of a ResultSet.
createResultSetSQL	This method executes a query with parameters taken from the Data Binder (*dataSource* and *whereClause* local data) rather than from given parameters. It also places the results in the local data using the ResultSet name found in the Data Binder *(resultName)*. Takes no parameters.
doSubService	This method executes a subservice. Takes one parameter, which is the name of a subservice.
loadDefaultInfo	This method is used for creating checkin and update pages. It first executes the loadDefaultInfo filter, and then loads environment information, content types, formats, and accounts. Takes no parameters.
loadMetaOptionsLists	This method first executes the loadMetaOptionsLists filter, and the loads all options lists referred to in the DocMetaDefinition table. Takes no parameters.

Name	Description
loadSharedTable	This method is used to make a server-cached table available for a template. Use this method instead of executing a query when the data is already cached in the server. Takes two parameters. The first parameter is the name of the table to look up in the server's cached tables. The second is the name the table will be given when it is added to the data.
loadSharedTableRow	This method is used to retrieve cached information, such as data about a specific user. The value for the key in the request data is used to find the row in the cached table. The values of the row are mapped to the local data using the names of the columns as keys. Takes two parameters. The first parameter is the name of the table to look up in the server's cached tables. The second parameter is an argument specifying a column in the database and a lookup key into the request data.
mapResultSet	This method is used to replace a Type 5 action when the service requires only a part of the first row of a ResultSet to be stored. It executes the specified query and maps the specified columns of the first row of the ResultSet to the local data. Takes at least three parameters. The first parameter is the name of a select query; the parameters that follow must appear in comma-separated pairs. The first member of the pair is the column name, and the second member is the key that is used to put the row value into local data.
refreshCache	This method performs a refresh on the specified Subjects (page 9-33). Takes one or more parameters, as a comma-separated list of subjects.

Name	Description
renameValues	This method assigns the value from one variable to another variable. Takes one or more sets of parameters that must appear in comma-separated pairs. The first member of a pair is the variable name that is looked up in the Data Binder, and the second member is the variable name that stores the found value in the local data.
setConditionVars	This method sets condition variables to true (1) or false (0). These values can be tested only in HTM template pages. They are not put into local data. Takes one or more sets of parameters that must appear in comma-separated pairs. The first member of a pair is the name of the condition variable, and the second member is the value (1 or 0).
setLocalValues	This method places name/value pairs into the local data. Takes one or more sets of parameters that must appear in comma-separated pairs. The first member of a pair is the variable name, and the second member is the value.

- Load Option List: This action type is used to load an option list stored in the system.
- Select Cache Query: This action type is used to select a query and then cache the query results.

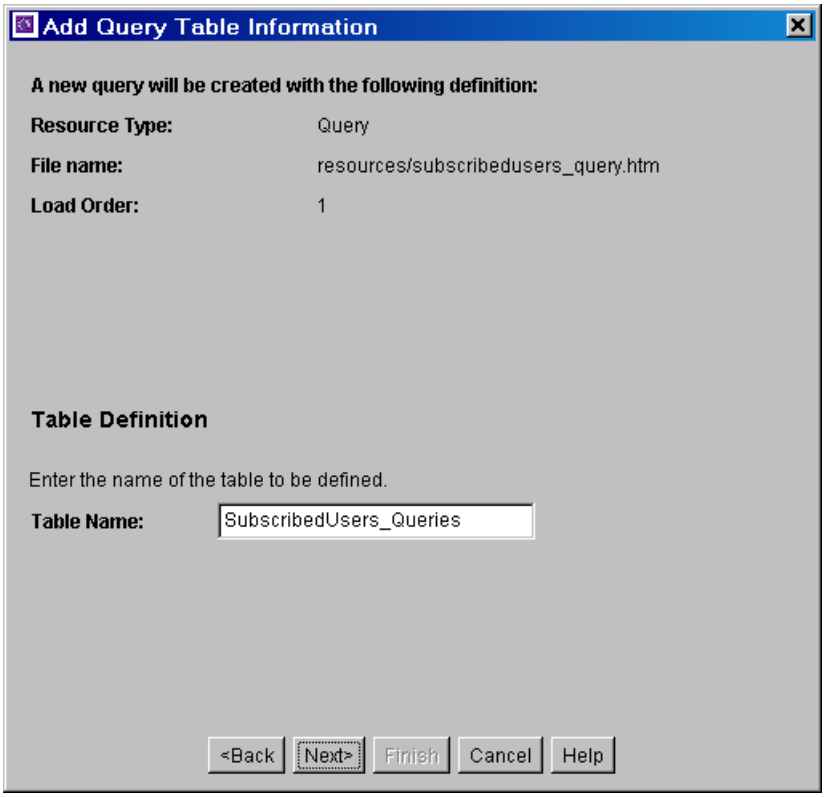
Add Screen



The Add Screen is used during the build process to specify what should be included in the component zip file.

Feature	Description
Entry Type list	Select the type of item to be included in the component Zip file. Each entry type has a default location (Current Root Prefix) that cannot be changed. The Component Class option ensures that the components and related files are placed in the 'component' directory.  **Note:** If your component must work with earlier versions of Content Server (pre-7.0), the following Entry Type options are not compatible: Component Class, Component Library, Bin, Data, Weblayout, and Resources.
Current Root Prefix field	Shows the directory where the specified files will be copied when the component Zip file is unpackaged.
Sub Directory or File field	Enter the sub-directory that contains the component files of the selected type, or enter an individual file name. If an individual file is contained in a sub-directory of the current root prefix, enter the sub-directory along with the file name. For example, *new_custom/new_component.htm*.
Make Jar check box	**Selected** = A Jar file is created and included in the manifest file. Selecting this option enables the Include Java Source option. **Clear** = A Jar file is not created.
Include Java Source check box	**Selected** = Source files are included which allows the Java source code to be shipped with the component. This option is only available if the Make Jar check box is selected. **Clear** = Source files are not included.
OK button	Adds the specified item to the component Zip file list.
Cancel button	Closes the Add screen without adding an item to the component Zip file list.
Help button	Displays this help page for the Add screen.

Add Query Table Information Screen

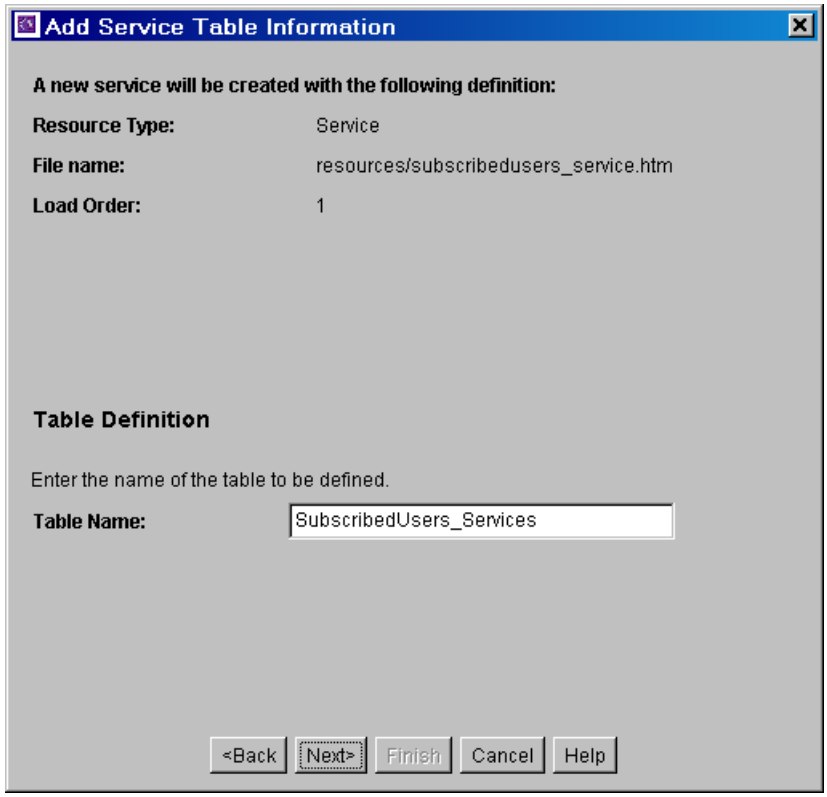


This screen is used to specify the database table to be used with a component's query.

Feature	Description
Table Name field	Enter the name of the query table that will be created for the new resource. The default is the name of the component followed by an underscore and the string *Queries*.
Back button	Displays the Add Resource screen.
Next button	Displays the Add Query screen.
Finish button	Creates the new query resource. This button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add Query Table Information screen without creating a new resource.

Feature	Description
Help button	Displays this help page for the Add Query Table Information screen.

Add Service Table Information Screen

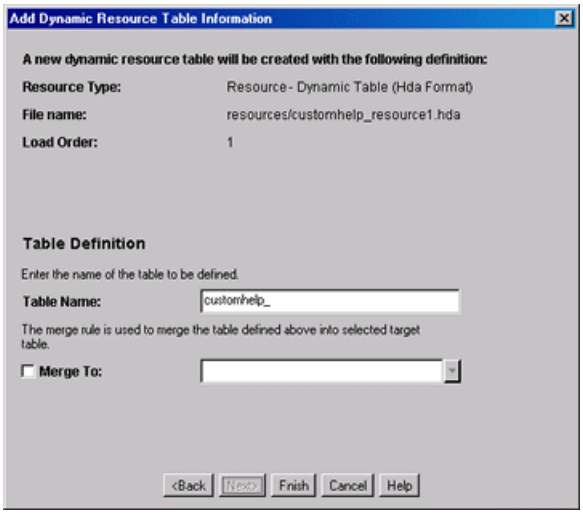


This screen is used to specify the database to be used by the service in the component.

Feature	Description
Table Name field	Enter the name of the service table that will be created for the new resource. The default is the name of the component followed by an underscore and the string *Services*.
Back button	Displays the Add Resource screen.
Next button	Displays the Add Service screen.

Feature	Description
Finish button	Creates the new service resource. This button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add Service Table Information screen without creating a new resource.
Help button	Displays this help page for the Add Service Table Information screen.

Add Dynamic Resource Table Information Screen



This screen is used to create dynamic tables to be used in a custom component.

Feature	Description
Table Name field	Enter the name of the dynamic table that will be created for the new resource. The default is the name of the component followed by an underscore.
Merge To check box and field	Used to create a merge rule for the new dynamic table. Select this check box and either select the target table from the list of Predefined Dynamic Tables (page 9-19) or enter the name of a custom table.

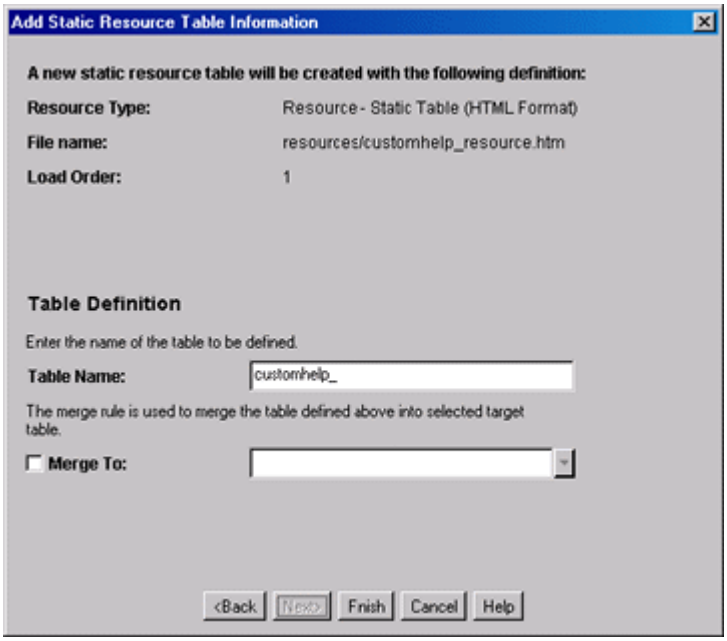
Feature	Description
Back button	Displays the Add Resource screen.
Next button	Inactive (there are no more screens for defining a dynamic resource table).
Finish button	Displays the Column Information screen. This button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add Dynamic Resource Table Information screen without creating a new resource.
Help button	Displays this help page for the Add Dynamic Resource Table Information screen.

Predefined Dynamic Tables

The following dynamic resource table is predefined in the Content Server:

Table Name	mergeColumns	Description
IgnoredFlexFields	templatename, flexareaname, fields	Used to exclude any custom metadata fields from specific template pages. Wildcards are supported for both the *templatename* and *flexareaname* columns.

Add Static Resource Table Information Screen



This screen is used to specify a static resource in your component.

Feature	Description
Table Name field	Enter the name of the static table that will be created for the new resource. The default is the name of the component followed by an underscore.
Merge To check box and field	Creates a merge rule for the new static table. Select this check box and either select the target table from the list of Predefined Static Tables (page 9-21) or enter the name of a custom table.
Back button	Displays the Add Resource screen.
Next button	Inactive (there are no more screens for defining a static resource table).
Finish button	Displays the Column Information screen. This button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add Static Resource Table Information screen.

Feature	Description
Help button	Displays this help page for the Add Static Resource Table Information screen.

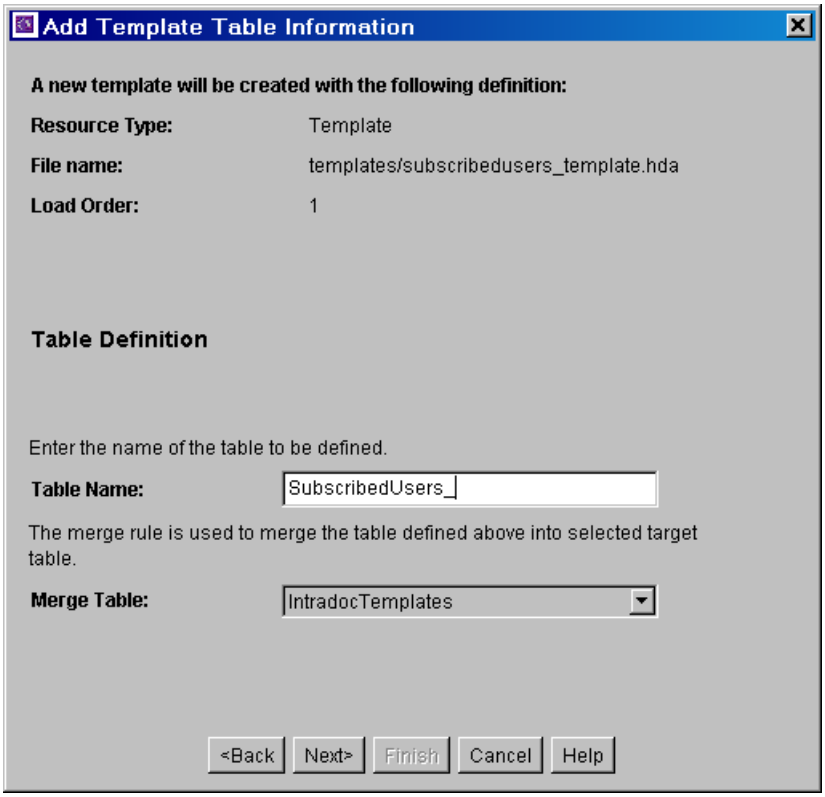
Predefined Static Tables

The following static resource tables are predefined in the Content Server:

Table Name	mergeColumns	Table Location (in *<install_dir>*/shared/config/)	Description
ColumnTranslation	column, alias	/resources/upper_clmns_map.htm	Contains uppercase database fields with their translated field names. This table is required for databases that use all uppercase (such as Oracle).
DataSources	name, dataSource, useMaxRows	/resources/std_resources.htm	Contains the queries that are executed to create reports in the Web Layout Editor.
IntradocReports	name, datasource, filename, description	/reports/reports.hda	Contains the list of report templates.
IdocScriptExtensions	name, class, loadOrder	/resources/std_resources.htm	Contains specializations of the ScriptExtensionsAdaptor. Used to create new Idoc Script functions and variables.
ServiceHandlers	serviceName, handler, searchOrder	/resources/std_resources.htm	Contains specializations of the ServiceHandler base class. Defines Java methods for handling service script Java functions.

Table Name	mergeColumns	Table Location (in <*install_dir*>/shared/config/)	Description
SubscriptionTypes	type, fields, description	/resources/std_resources.htm	Contains document subscription types. Default subscription is by document name. Document criteria subscriptions can be defined in this table.
UserMetaDefinition	umdName, umdType, umdCaption, umdIsOptionList, umdOptionListType, umdOptionListKey, umdIsAdminEdit, umdOverrideBitFlag	/resources/std_resources.htm	Contains the definitions of the auxiliary user metadata fields. Values for *umdOverrideBitFlag* should start at 16 (0x10) or higher. See the design of the *DocMetaDefinition* database table for a description of the appropriate contents of these fields.

Add Template Table Information Screen



This screen is used to specify the table that will be accessed for the template used in the component.

Feature	Description
Table Name field	Enter the name of the template table that will be created for the new resource. The default is the name of the component followed by an underscore.
Merge Table list	Creates a merge rule for the new dynamic table. Select a target table from the list of Predefined Template Tables (page 9-24).
Back button	Displays the Add Resource screen.
Next button	Depending on which Merge Table is selected, displays the Add Intradoc Template screen or Add SearchResults Template screen.

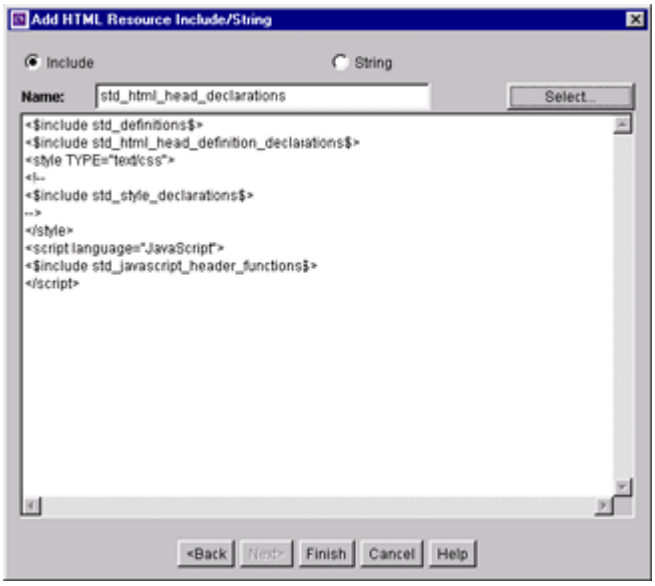
Feature	Description
Finish button	Creates the new template resource. This button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add Template Table Information screen without creating a new resource.
Help button	Displays this help page for the Add Template Table Information screen.

Predefined Template Tables

The following template tables are predefined in the Content Server:

Table Name	mergeColumns	Description
IntradocTemplates	name, class, formtype, filename, description	This is a ResultSet table that defines the templates used in the Content Server.
SearchResultTemplates	name, formtype, filename, outfilename, flexdata, description	This table is used to create result templates in memory for use with results that are returned from the search engine.

Add/Edit HTML Resource Include/String Screen

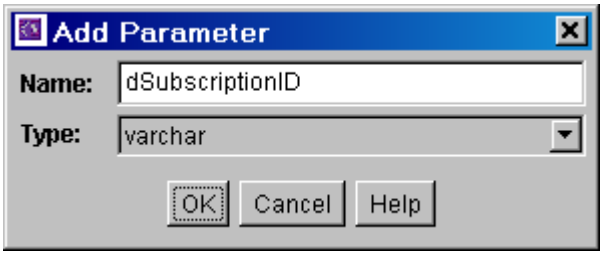


This screen is used to specify a customized HTML resource or a customized string resource in a component.

Feature	Description
Resource type options	**Include** = The resource defines an HTML include. **String** = The resource defines a locale-sensitive string.
Name field	Enter the name of the include or string that will be created, or click Select to start with a predefined include. For string names, use the following prefix conventions: **sy**`StringName`: System-level messages and errors. **cs**`StringName`: Content Server messages and log messages (this is the most common type of string). **ww**`StringName`: Strings used on Web pages. **ap**`StringName`: Strings used in applets.
Select button	Displays the Resource Selection Dialog screen, which lists the predefined includes. This button is available only when the Include option is selected.

Feature	Description
Code field	Shows the code for the include or string, which can be edited directly in this field. If a predefined include is selected, the code is automatically added to this field.
Back button	Displays the Add Resource screen.
Next button	Inactive (there are no more screens for defining an include or string).
Finish button/OK button	Saves the include or string resource and asks if you want to open the text file for editing.
Cancel button	Closes the Add/Edit HTML Resource Include/String screen without saving the include or string.
Help button	Displays this help page for the Add/Edit HTML Resource Include/String screen.

Add/Edit Parameter Screen

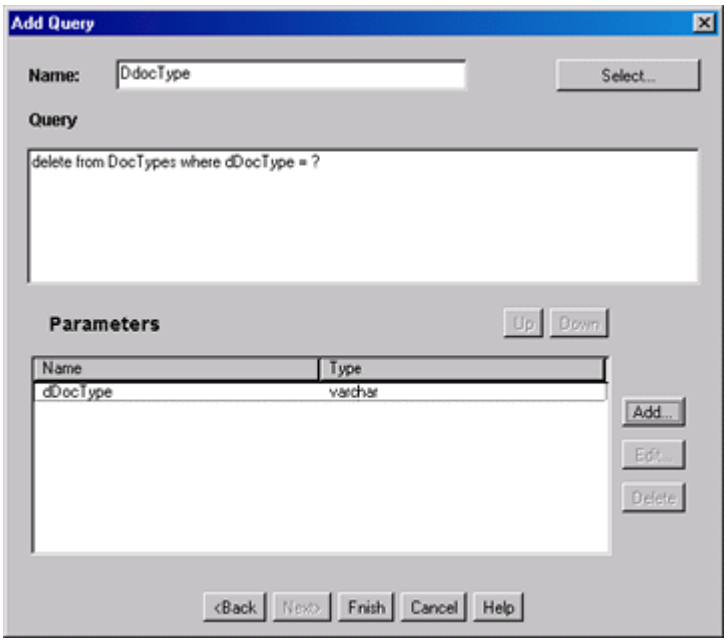


This screen is used to define the parameters that will be passed to your defined resources.

Feature	Description
Name field	Enter a name for the parameter. A parameter name cannot contain spaces.
Type field	Select the type of parameter:
OK button	Saves the parameter in the query.

Feature	Description
Cancel button	Closes the Add/Edit Parameter screen without creating or changing the parameter.
Help button	Displays this help page for the Add/Edit Parameter screen.

Add/Edit Query Screen

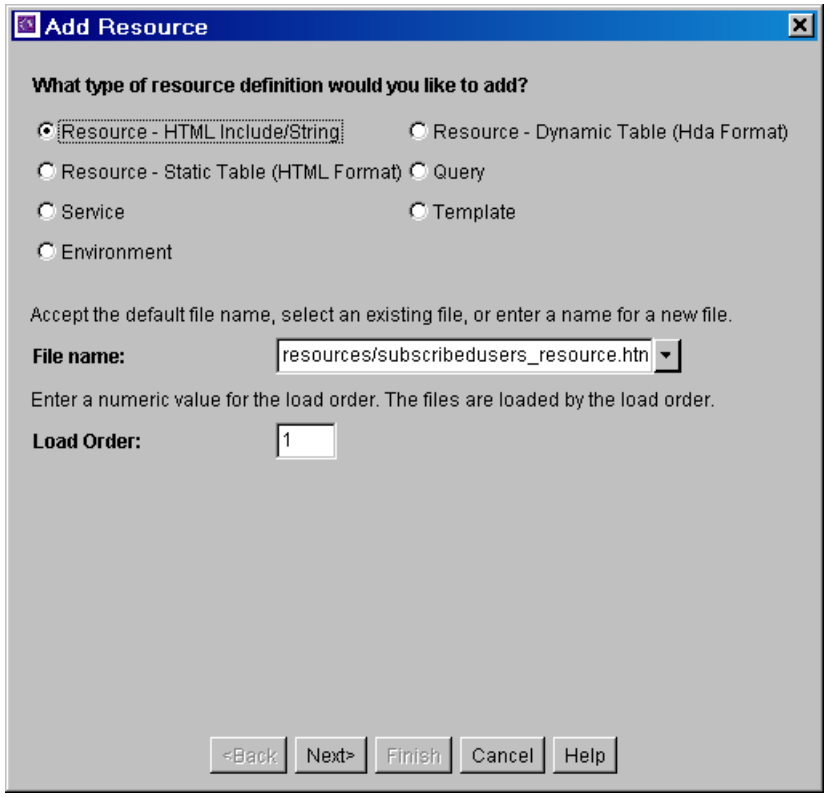


This screen is used to specify the SQL query for the query resource defined in the component.

Feature	Description
Name field	Enter the name of the query that will be created for the resource, or click Select to start with a predefined query.

Feature	Description
Select button	Displays the Resource Selection Dialog screen, which lists the predefined queries.  **Note:** See the *query.htm*, *workflow.htm*, and *indexer.htm* files in the <*install_dir*>/shared/config/resources/ directory for more information on predefined queries.
Query field	Shows the SQL query expression, which can be edited directly in this field. If an existing query is selected, the query expression is automatically added to this field.
Parameters list	Lists the name and type for each parameter defined for the query. Parameters must be listed in the order they appear in the query expression.
Up and Down buttons	Move the selected parameter up or down in the Parameters list.
Add button	Displays the Add Parameter screen.
Edit button	Displays the Edit Parameter screen for the selected parameter.
Delete button	Deletes the selected parameter from the Parameters list.
Back button	Displays the Add Query Table Information screen.
Next button	Inactive (there are no more screens for defining a query).
Finish button/OK button	Saves the query in the query resource. The Finish button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add/Edit Query screen without creating or changing the query resource.
Help button	Displays this help page for the Add/Edit Query screen.

Add Resource Screen

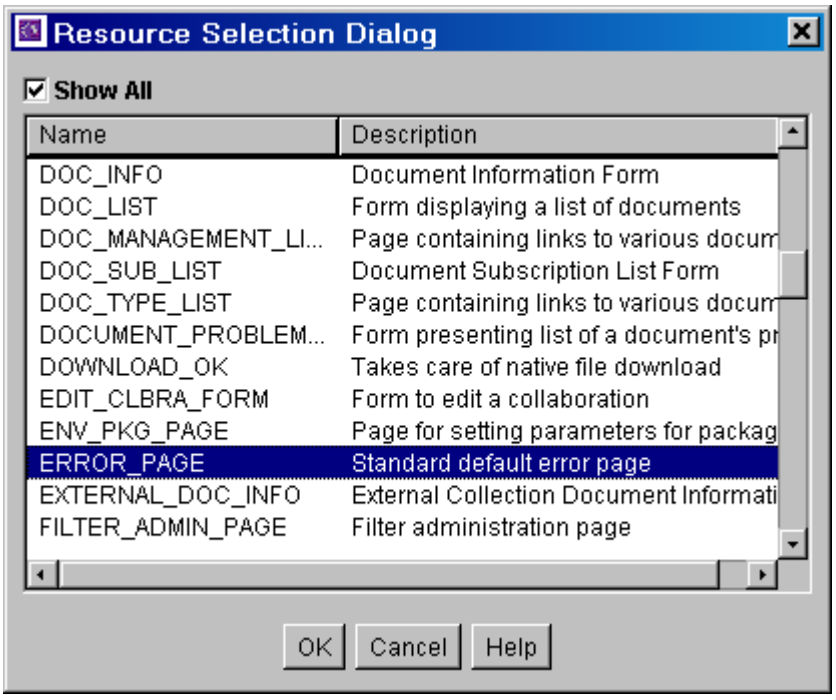


The Add Resource screen is used to select the type of resource you will use in your component. It is used to select a variety of resources.

Feature	Description
Resource Type options	Select an option from the Resource Type list.
File Name field	Select or enter the path and file name for the new resource.
Load Order field	Enter a load order number for the resource. Lower values are loaded first. Resources that have the same load order number are loaded in the order they appear in the component definition (“glue”) file.
Back button	Displays the previous page of the resource definition screens (if any).

Feature	Description
Next button	Displays the next page of the resource definition screens (if any).
Finish button	Creates the new resource. This button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add Resource screen without creating a new resource.
Help button	Displays this help page for the Add Resource screen.

Resource Selection Dialog Screen

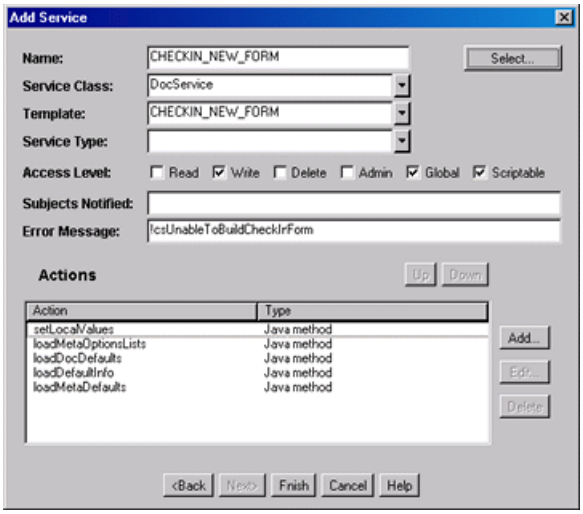


This screen is used to select an existing resource for use or to edit for your component.

Feature	Description
Show All check box	**Selected** = All predefined items are displayed. **Clear** = The most commonly used predefined items are displayed.
Name column	Lists the predefined items.

Feature	Description
Description column	Describes each predefined item.
OK button	Selects the selected option and fills in fields on the associated "Add" screen.
Cancel button	Closes the Resource Selection Dialog screen without selecting a resource item.
Help button	Displays this help page for the Resource Selection Dialog screen.

Add/Edit Service Screen



This screen is used to enter the information for the service being created by the component.

Feature	Description
Name field	Enter the name of the service that will be created, or click Select to start with a predefined service.
Select button	Displays the Resource Selection Dialog screen, which lists the predefined services.

Feature	Description
Service Class field	Select a service class from the list or enter a name for a custom service class. The service class determines what actions can be performed by the service. There are actions that all services share, while other actions are specific to the service class.
Template field	Select a template to present the results of service. If the results of the service do not require page presentation, leave this field blank. For example, the PageHandlerService, which is called from an applet, does not specify a template page.
Service Type field	If the service is to be executed inside another service, select SubService.
Access Level check boxes	Select one or more check boxes to assign a user access level to the service.
Subjects Notified field	Enter the Subjects (page 9-33) (subsystems) to be notified by the service as a comma-separated string. If a service changes one or more subjects, it must notify the affected subjects of changes. For example, the ADD_USER service adds a new user to the system and informs the system that the *userlist* subject has changed.
Error Message field	Enter the error message to be displayed by this service. This error message is returned by the service if no action error message overrides it. The error message can be a plain text string, or it can be a parameter to be looked up in the Content Server language strings (for example, *!csUnableToBuildCheckInForm*).
Actions list	Lists the name and type for each action defined for the service. Actions are used to execute an SQL statement, perform a query, run code, cache the results of a query, or load an option list. The order of the list specifies the order in which the actions are performed.
Up and Down buttons	Move the selected action up or down in the Actions list.
Add button	Displays the Add Action screen.

Feature	Description
Edit button	Displays the Edit Action screen for the action selected in the Actions list.
Delete button	Deletes the selected action from the Actions list.
Back button	Displays the Add Service Table Information screen.
Next button	Inactive (there are no more screens for defining a service).
Finish button/OK button	Saves the service in the service resource. The Finish button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add/Edit Service screen without creating or changing the service.
Help button	Displays this help page for the Add/Edit Service screen.

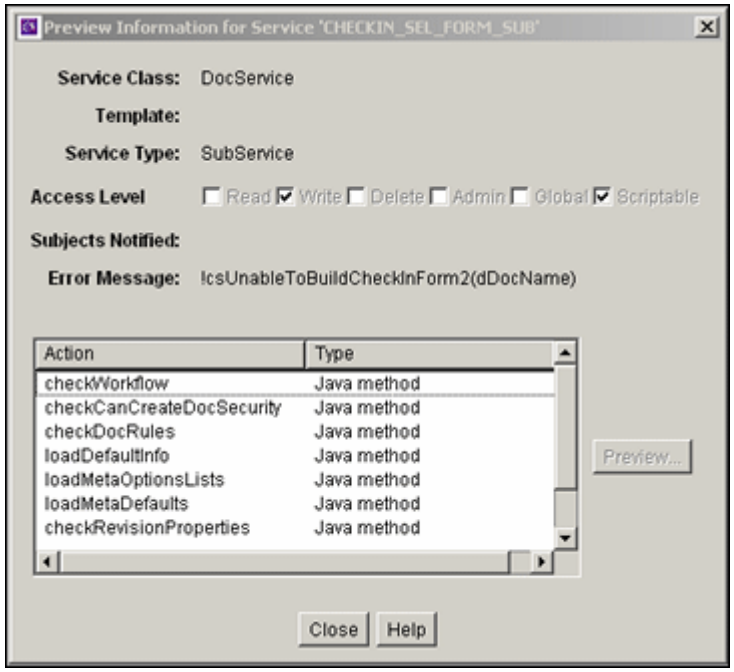
Subjects

Subjects are subsystems within the Content Server. When a service makes a change (such as add, edit, or delete) to one of the following subjects, the subject must be notified:

- accounts
- aliases
- collections
- docformats
- doctypes
- documents
- dynamicqueries
- indexerwork
- metadata
- metaoptlists
- subscriptiontypes

- templates
- userlist
- usermetaoptlists
- wfscripts
- wftemplates
- workflows

Preview Information for Service Screen



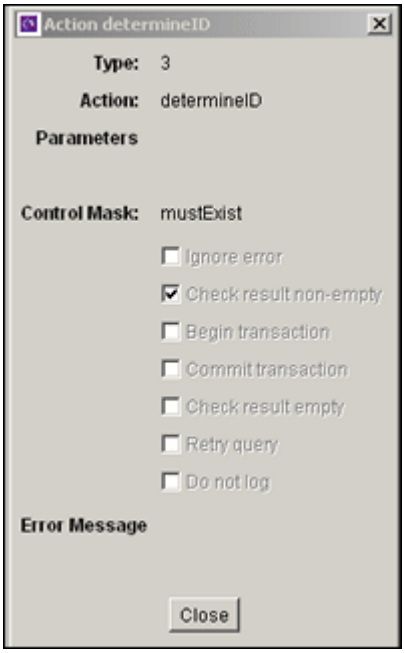
This screen is used to view details about a service before selecting it for use as a service resource. To access this screen, highlight a service on the Add/Edit Service Screen (page 9-31) and click **Preview**.

To view details about the actions used in the service, highlight an action and click **Preview.** The Preview Action Information Screen (page 9-35) is displayed.

When you finish viewing service information, click **Close**.

For complete information about services and actions, see the *Services Reference Guide*.

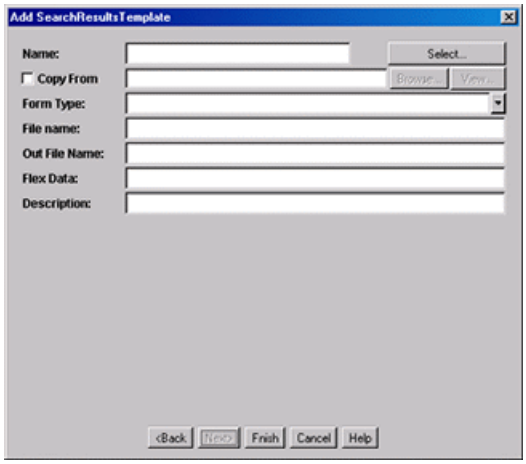
Preview Action Information Screen



This screen is used to view the details of service actions. To access this screen, highlight an action on the Preview Information for Service Screen (page 9-34) and click **Preview**. When done viewing action details, click **Close**.

For complete information about services and actions, see the *Services Reference Guide*.

Add/Edit SearchResults Template Screen

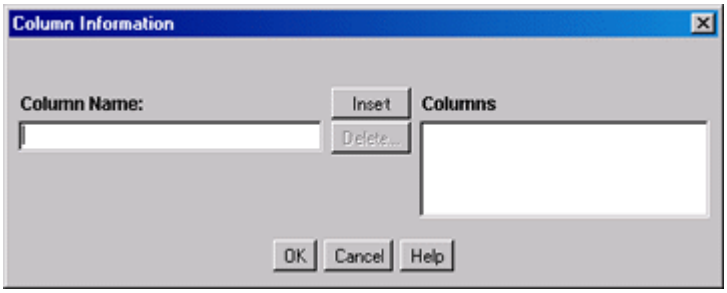


This screen is used to find a template to use for your component.

Feature	Description
Name field	Enter the name of the template that will be created for the resource, or click Select to start with a predefined template.
Select button	Displays the Resource Selection Dialog screen, which lists the predefined *StandardResults* template.
Copy From check box and field	**Selected =** The new template resource will be a copy of an existing template. Enter the complete path and file name of the existing template file (.htm). **Clear =** A new template resource file will be created.
Browse button	Used to navigate to and select the desired template file.
View button	Displays the template file in a read-only text window.
Form Type field	Select the template form type, which is the specific type of functionality the page is trying to achieve.
File name field	The file name of the template resource. This can be either an absolute path or a relative path, relative to the location of the *component_template.hda* resource file.
Out File Name field	For future use. Leave this field blank.
Flex Data field	Defines the metadata to be displayed for each row on the search results page.
Description field	Enter a description of the template file.
Back button	Displays the Add Template Table Information screen.
Next button	Inactive (there are no more screens for defining a template).
Finish button/OK button	Saves the template file in the template resource. The Finish button is unavailable if the minimum specifications have not been defined for the resource.

Feature	Description
Cancel button	Closes the Add/Edit SearchResults Template screen without creating or changing the template resource.
Help button	Displays this help page for the Add/Edit SearchResults Template screen.

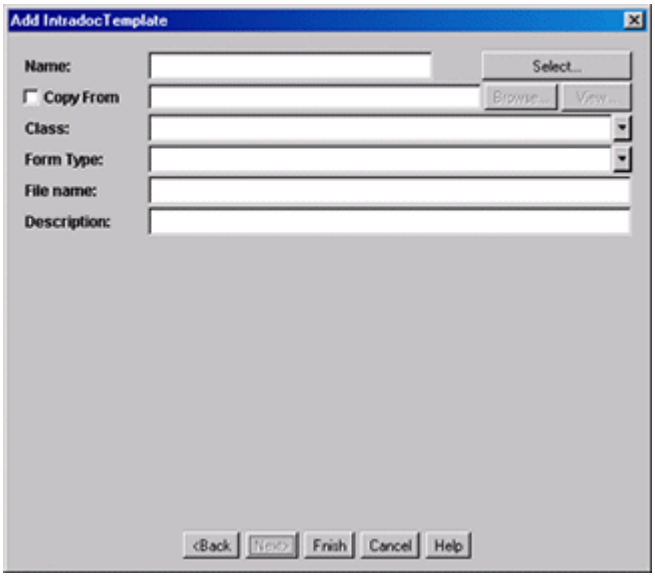
Column Information Screen



This screen appears only when you create a new table. To edit the table, you will need to open the file in a text editor.

Feature	Description
Column Name field	Enter a column name to be added to the bottom of the Columns list.
Insert button	Adds the Column Name entry to the bottom of the Columns list.
Delete button	Deletes the column selected in the Columns list.
Columns list	Lists the columns that are defined for the table.
OK button	Saves the column list.
Cancel button	Closes the Column Information screen without saving the column list.
Help button	Displays this help page for the Column Information screen.

Add/Edit Intradoc Template Screen

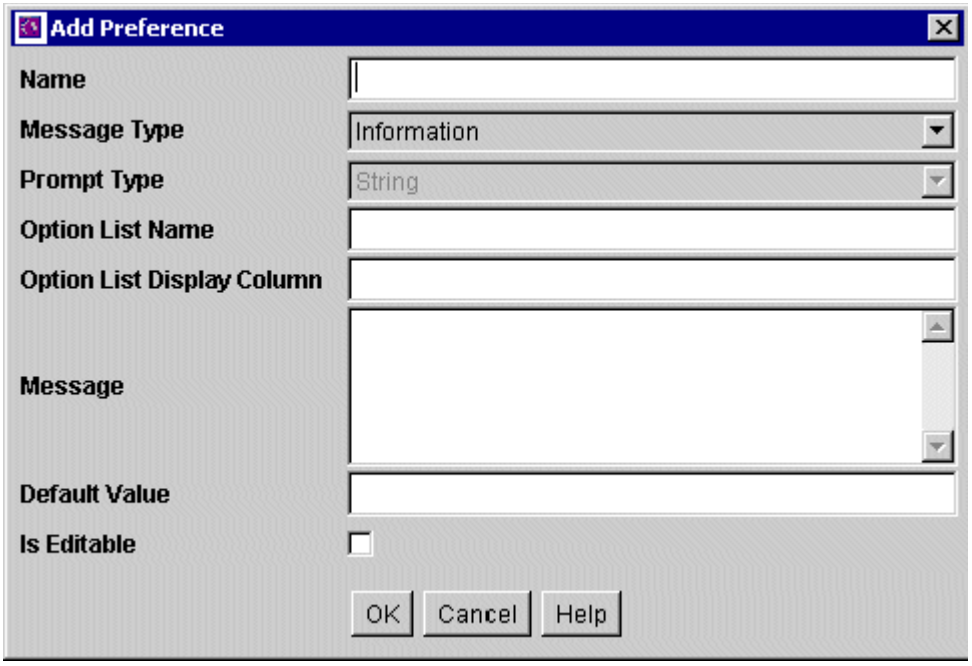


This screen is used to begin building a template for your component.

Feature	Description
Name field	Enter the name of the template that will be created for the resource, or click Select to start with a predefined template. The unique template name is how the template is referenced in the Content Server CGI URLs and in code. When merging custom template file entries into the Templates table, the Name is used as the merge key.
Select button	Displays the Resource Selection Dialog screen, which lists the predefined templates.
Copy From check box and field	**Selected** = The new template resource will be a copy of an existing template. Enter the complete path and file name of the existing template file (.htm). **Clear** = A new template resource file will be created.
Browse button	Used to navigate to and select the desired template file.
View button	Displays the template file in a read-only text window.

Feature	Description
Class field	Select the template class type, which is the general category of the template. The template class is not used for standard Content Server functionality, but it can be used in a component to create functions specific to a particular class of templates.
Form Type field	Select the template form type, which is the specific type of functionality the page is trying to achieve.
File name field	The file name of the template resource. This can be either an absolute path or a relative path, relative to the location of the *component_template.hda* resource file.
Description field	Enter a description of the template file.
Back button	Displays the Add Template Table Information screen.
Next button	Inactive (there are no more screens for defining a template).
Finish button/OK button	Saves the template file in the template resource. The Finish button is unavailable if the minimum specifications have not been defined for the resource.
Cancel button	Closes the Add/Edit Intradoc Template screen without creating or changing the template resource.
Help button	Displays this help page for the Add/Edit Intradoc Template screen.

Add/Edit Preference Screen



This screen is used to specify custom installation parameters.

Feature	Description
Name field	Name of the custom installation parameter.
Message Type field	Select a message type.
Prompt Type field	Select the prompt type. This field is only enabled if Prompt is selected as the Message Type.
Option List Name field	Enter the result set name from the Content Server. This field is enabled when Option List is selected in the Prompt Type field.
Option List Display Column field	Enter the field name from the result set specified in the Option List Name field. This field is used for building a choice list. This field is enabled when Option List is selected in the Prompt Type field.

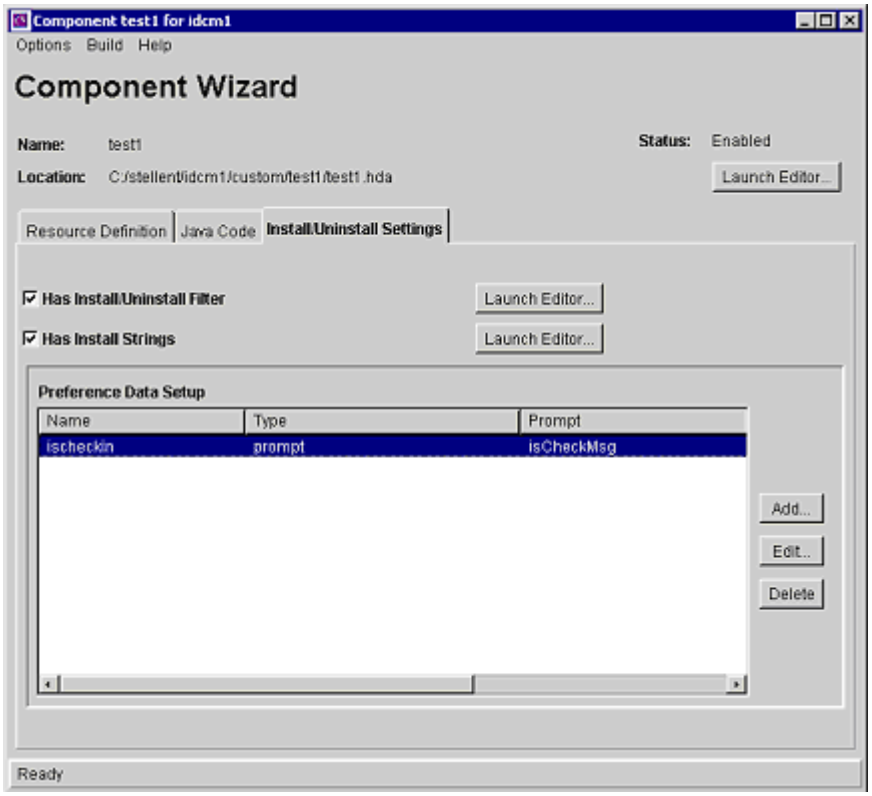
Feature	Description
Message field	Enter the prompt or message text or, preferably, enter the key associated with the prompt or message (created using the code template file corresponding to the Has Install Strings check box on the Install/Uninstall Settings tab). Entering the key references the installation strings file to obtain the actual text (which can be edited for localization requirements).
Default Value field	Enter the default value for the prompt.
Is Editable check box	**Selected** = Makes the installation parameter configurable after the component is installed. **Clear** = The installation parameter is not reconfigurable.
OK button	Adds the installation parameter to the component.
Cancel button	Closes the screen without adding the installation parameter to the component.
Help button	Displays a help page for the Add Preference screen.

BUILD SCREENS

The screens depicted in this section are used to package and build a custom component:

- Main Build Screen (page 9-44)
-

Install/Uninstall Settings Tab

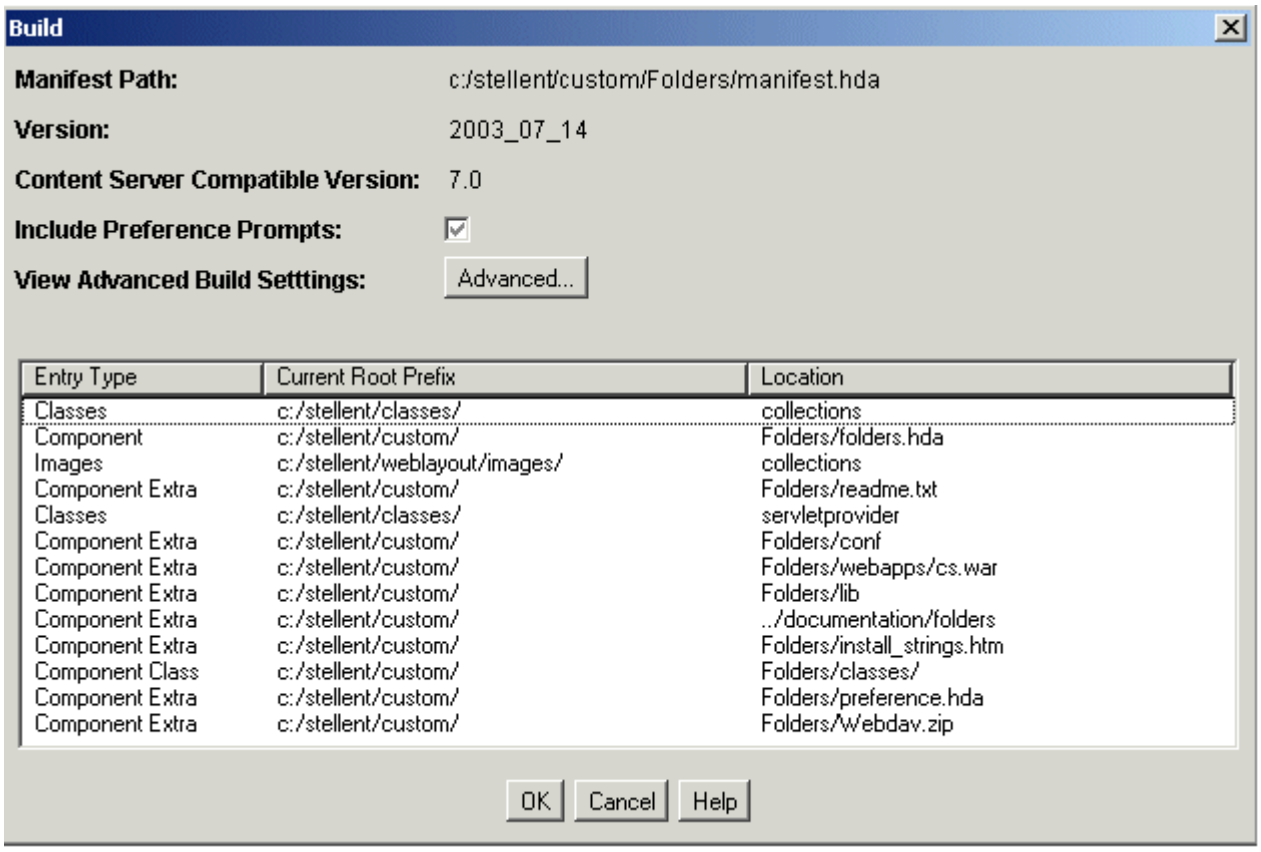


The Install/Uninstall Settings tab is used to create customized installation components that can include preference data parameters. These parameters can be user prompts and messages. The user prompts and messages created for specific components are displayed to users only during the installation process.

Feature	Description
Has Install/Uninstall Filter check box	**Selected** = Includes additional custom installation or uninstall filters in the component resource definition file. Checking this option also creates the template java source if the file doesn't already exist. The <component_name>InstallFilter.java file is created in the <component_name>/classes/<component_name> directory. **Clear** = Additional custom installation procedures are not included.

Feature	Description
Launch Editor button	Displays a code template file in the default text editor. Edit the template Java source to define custom initialization or uninstall procedures for the component (such as creating meta fields, executing service scripts, etc.)
Has Install Strings check box	**Selected** = Includes prompts or messages during the component installation process. These prompts or messages are stored in an installation strings file and can be edited for localization requirements. Checking this option also creates the install_strings.htm file if the file doesn't already exist in the <component_name> directory. **Clear** = Prompts or messages are not included.
Launch Editor button	Displays a code template file in the default text editor. Edit the template to define prompts or messages for the component.
Preference Data Setup list	Shows the name, type and prompt fields of the custom installation parameters. If one or more custom installation parameters are defined and included, the preference.hda file will be created in the component directory.
Add button	Displays the Add Preference screen, which is used to define the settings for custom installation parameters.
Edit button	Displays the Edit Preference screen, which is used to edit the settings for custom installation parameters.
Delete button	Removes the selected parameter from the component.

Main Build Screen

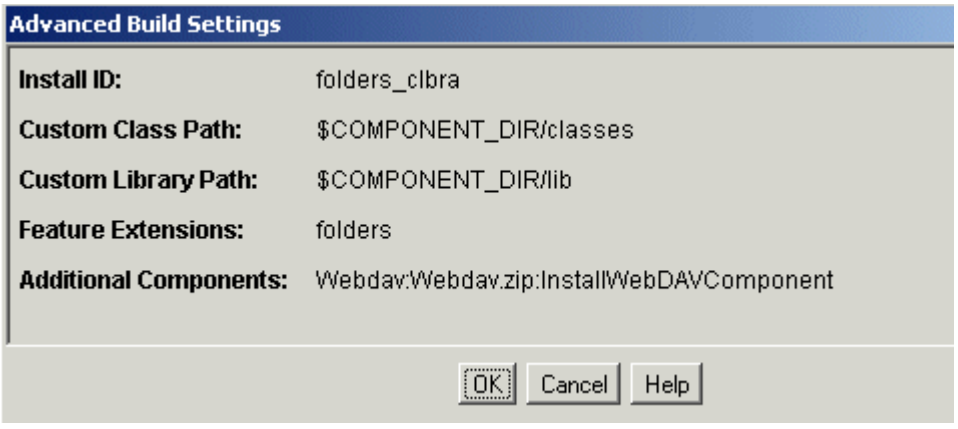


This screen is used during the build process, when creating a zip file of a custom component. It shows the files that will be included in the zip file.

Feature	Description
Manifest Path	Shows the path and file name of the *manifest.hda* file, which contains the instructions for how to unpackage the component zip file.
Version field	Supports component versioning. By default, the date is listed with a build number in parenthesis, but this value can be overridden. It is used for reference purposes only, and it is not validated.

Feature	Description
Include Preference Prompts check box	**Selected** = The parameter option settings (preference data) that were established using the Install/Uninstall Settings tab are included in the component manifest file. By selecting this, the preference.hda file, which holds preference data settings, will be included. **Clear** = The preference data is not included in the component manifest file.
View Advanced Build Settings / Advanced button	Displays the Advanced Build Settings Review Screen (page 9-45) which lists the field values configured using the Advanced Build Settings Screen (page 9-49).
Entry Type column	Lists the items that will be included in the component Zip file.
Current Root Prefix column	The root directory where the component files are located.
Location column	The sub-directory and/or and the file name of the component file.
OK button	Builds the component Zip file.
Cancel button	Closes the screen without building the component Zip file.
Help button	Displays the help page for this screen.

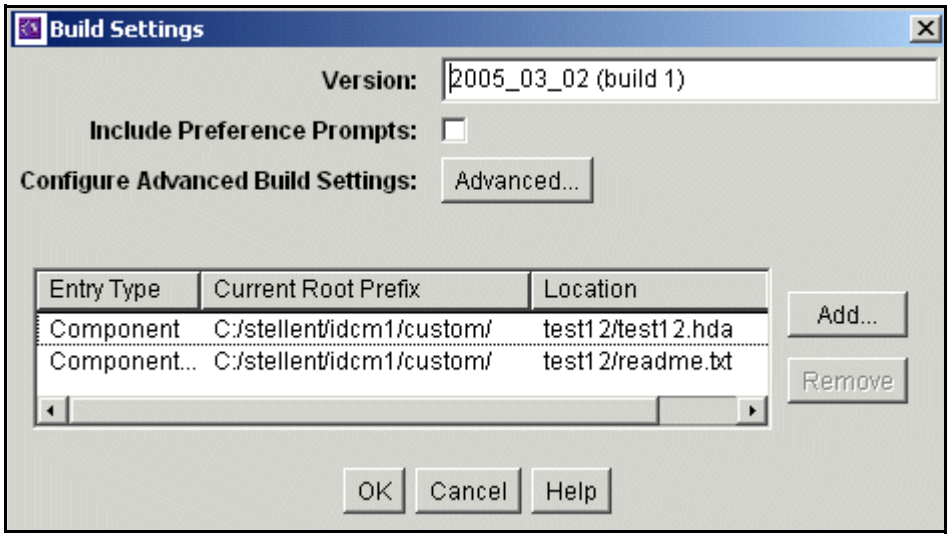
Advanced Build Settings Review Screen



This screen is accessed by clicking the **OK** button on the Advanced Build Settings Screen (page 9-49). It shows those options which have been specified.

Feature	Description
Install ID field	Stores the user's preference settings that are specified during the component installation.
Custom Class Path field	Lists the class paths of additional classes or jar files that are included in the classpath that is required for the component to function.
Custom Library Path field	Lists the paths of library or other executable files required for the component.
Feature Extensions field	Lists the specific types of features or functions that the component provides.
Additional Components field	Lists the additional components that need to be installed along with the component.
OK button	Accepts the field values and closes the screen.
Cancel button	Closes the screen.
Help button	Displays the help page for this screen.

Build Settings Screen

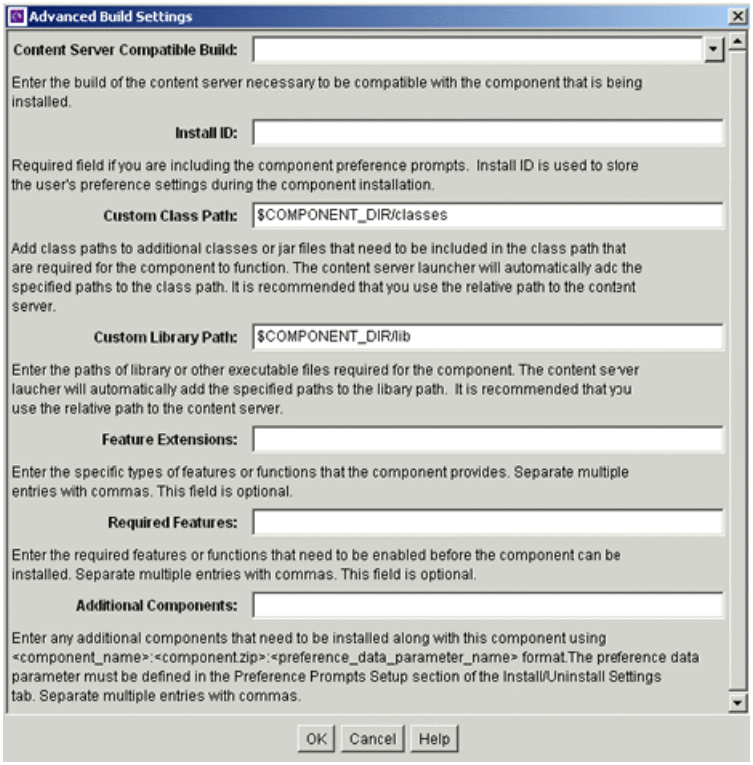


The Build Settings screen defines installation settings and what files to include in the component Zip file. The list of files included in the component Zip file is saved in the component build file (manifest.hda) and the installation settings are saved in the component definition file (<component_name>.hda). See Additional Component Wizard Tasks (page 5-15) for an example of usage.

Feature	Description
Version field	Supports component versioning. By default, the date is listed with a build number in parenthesis, but this value can be overridden. It is used for reference purposes only, and it is not validated.
Include Preference Prompts check box	**Selected** = The parameter option settings (preference data) that were established using the Install/Uninstall Settings tab are included in the component manifest file. By selecting this, the preference.hda file, which holds preference data settings, will be included. **Clear** = The preference data is not included in the component manifest file.
Configure Advanced Build Settings / Advanced button	Displays the Advanced Build Settings Screen (page 9-49) which is used to enter values for the advanced build settings fields. The configured values are viewed using the Advanced Build Settings Review Screen (page 9-45).

Feature	Description
Entry Type column	Lists the items that will be included in the component Zip file.
Current Root Prefix column	The root directory where the component files are located.
Location column	The sub-directory and/or and the file name of the component file.
Add button	Displays the Add Screen (page 9-14).
Remove button	Removes the selected item from the list.
OK button	Saves the build settings.
Cancel button	Closes the screen without changing the build settings.
Help button	Displays the help page for this screen.

Advanced Build Settings Screen



This screen is used to specify additional build settings for the component zip file. It is accessed by selecting the Advanced button on the Main Build Screen (page 9-44).

Feature	Description
Content Server Compatible Build field	The minimal build number of the Content Server, for which the component must be compatible. Use this field if the component is not compatible with other Content Server versions.The build number is displayed on the application About screens and on the Configuration Information page. The component will not be installed for a Content Server build number that is older than the value specified in this field. If a build number is not specified, then the component will be installed without checking the Content Server build number. For versions of Content Server older than 7.5.1, the build number was not as readily available. It was aligned with the product version as noted below: Display Value — Build number 6.2 — 6.2 7.0 — 7.0.1.145 7.1 — 7.0.1.149 7.1.1 — 7.0.3.158 7.5 — 7.1.2.166 7.5.1 — 7.1.2.169 The build number is composed of four, decimal-separated numbers. If you specify a build number, during validation, if a version is missing any values, these values are padded with zeros. This means that 7.0 becomes 7.0.0.0. For example, if a component has the compatible build number set to 7.0, and it is installed against a 7.5.1 Content Server, the server would check that 7.1.2.169 > 7.0.0.0, and load the component without complaint.

Feature	Description
Install ID field	Required if preference data is defined using the Install/Uninstall Settings tab. This field value is used during the component installation process to access the preference data stored in configuration files. Two configuration files hold the preference data: config.cfg (contains the parameters that can be reconfigured after installation) and install.cfg (contains the preference data definitions and prompt answers).
Custom Class Path field	Add class paths to additional classes or jar files that need to be included in the classpath that are required for the component to function. It is recommended that you use the relative path to the Content Server and place any component-related class files or jar files in the <component_name>/classes directory.
Custom Library Path field	Enter the paths of library or other executable files required for the component. It is recommended that you use the relative path to the Content Server and place any component-related library or executable files in the <component_name>/classes directory.
Feature Extensions field	Enter the specific types of features or functions that the component provides. Separate multiple entries with commas. This field is optional (not required).

Feature	Description
Additional Components field	This field allows installation components to install individual add-on components or to generate a grouping of multiple components into a single installation package. Enter any additional components that need to be installed along with this component using the following format: *<component_name>:<component.zip>:<preference_data_parameter_name>* The *<preference_data_parameter_name>* is optional. If a parameter name is not specified, the component will be installed by default. The preference data parameter must be defined in the Preference Data Setup section of the Install/Uninstall Settings tab. You must include a colon (:) after *<component.zip>* even if you are not including a *<preference_data_parameter_name>*. If excluding *<preference_data_parameter_name>*, the format is: *<component_name>:component.zip>:* Separate multiple entries with commas, as in the following: *<component_name>:<component.zip>:,<component_name>:<component.zip>:*
OK button	Returns to the previous screen.
Cancel button	Closes the screen without building the component Zip file.
Help button	Displays the help page for this screen.

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 (page B-1)
- W3C® Software Notice and License (page B-2)
- Zlib License (page B-3)
- General BSD License (page B-4)
- General MIT License (page B-5)
- Unicode License (page B-5)
- Miscellaneous Attributions (page B-7)

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

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.

Portions Copyright © 1997 Soft Horizons. All rights reserved.

Portions Copyright © 1995-1999 LizardTech, Inc. All rights reserved.

Red Hat is a registered trademark of Red Hat, Inc.

Sun is a registered trademark, and Sun ONE, Solaris, iPlanet and Java are trademarks of Sun Microsystems, Inc.

Sybase is a registered trademark of Sybase, Inc.

UNIX is a registered trademark of The Open Group.

Verity is a registered trademark of Autonomy Corporation plc

Index

A

access level attribute, 8-13
action format, 8-14
actions
    control mask, 8-15
    error message, 8-15
    name, 8-15
    parameters, 8-15
    predefined, 9-11
    service, 9-10
    service resource, 8-18
    type, 8-15
Actions column
    service ResultSet, 8-14
Add Action screen, 9-10
Add Component screen, 9-6
Add Dynamic Resource Table Information screen, 9-18
Add HTML Resource Include/String screen, 9-25
Add Intradoc Template screen, 9-38
Add Parameter screen, 9-26
Add Query screen, 9-27
Add Query Table Information screen, 9-16
Add Resource Definition screen, 9-29
Add screen
    Component Wizard, 9-14
Add SearchResults Template screen, 9-35
Add Service screen, 9-31
Add Service Table Information screen, 9-17
Add Static Resource Table Information screen, 9-20
Add Template Table Information screen, 9-23
adding
    existing components, 5-21
Admin Server
    enabling and disabling components, 6-3
    uploading components, 6-4
Asian language, 3-2
attributes
    service resources, 8-17

B

Build menu, 9-5
Build screen, 9-44
build settings, 5-15
build settings, component, 9-6
building, 9-44, 9-47
    component Zip file, 5-15
    components, 9-44, 9-47

C

CFG file, 8-32
ClassAliases ResultSet, 7-8
Column Information screen, 9-37
ColumnTranslation table, 9-21
component
    creating with Component Wizard, 5-3
Component Configuration screen, 9-8
component definition file, 3-6
component directories, 3-1, 4-1
component files, 3-1, 4-1
    overview, 2-3
Component List screen, 9-2
Component Manager, 2-2, 2-4, 6-1, 9-1
    overview, 2-2
    using, 6-3
Component Manager page, 6-1
component manager tasks, 6-3
component overview, 2-1
Component Wizard, 2-1, 2-4, 4-2, 5-15, 6-1, 9-1
    Add Action screen, 9-10
    Add Dynamic Resource Table Information screen, 9-18
    Add HTML Resource Include/String screen, 9-25
    Add Intradoc Template screen, 9-38
    Add Parameter screen, 9-26
    Add Query screen, 9-27
    Add Query Table Information screen, 9-16
    Add Resource Definition screen, 9-29
    Add screen, 9-14

Add SearchResults Template screen, 9-35
Add Service screen, 9-31
Add Service Table Information screen, 9-17
Add Static Resource Table Information screen, 9-20
Add Template Table Information screen, 9-23
adding existing components, 5-21
Build menu, 9-5
Build screen, 9-44
Build Settings screen, 9-47
Column Information screen, 9-37
configuring default HTML editor, 5-20
creating a dynamic table, 8-8
creating a new component, 5-1
creating component example, 5-3
creating HTML includes, 5-11
creating static tables, 5-14
Edit Action screen, 9-10
Edit HTML Resource Include/String screen, 9-25
Edit Intradoc Template screen, 9-38
Edit Parameter screen, 9-26
Edit Query screen, 9-27
Edit SearchResults Template screen, 9-35
Edit Service screen, 9-31
editing dynamic tables, 8-8
editing environment resources, 8-34
editing HTML includes, 8-3, 8-8
editing service resources, 8-23
editing static tables, 8-9
editing template resources, 8-32
editing the Readme file., 5-3
enabling and disabling components, 5-18
Help menu, 9-6
Java code tab, 5-2
opening components, 5-19
Options menu, 9-5
overview, 2-1
removing components, 5-19
Resource Selection Dialog screens, 9-30
unpackaging components, 5-20
using, 5-1
working with resources, 5-2, 8-2
component Zip file, 3-11
building, 5-15
components, 6-1, 9-1
adding existing, 5-21
build settings, 9-6
building, 9-44, 9-47
creating, 4-1, 5-1
disabling, 5-18, 6-3
enabling, 5-18, 6-3
naming conventions, 4-4
opening, 5-19
removing, 5-19
unpackaging, 5-20
uploading, 6-4
working with files, 4-2
components HDA file, 3-11, 7-1
Components ResultSet, 3-4, 7-2
components.hda file, 3-11, 7-1
configuration files, 8-32
configuring
default HTML editor, 5-20
content server
development instance, 4-2
control mask, 8-15
conventions
naming, 4-4
creating
component, 4-1
dynamic tables, 8-8
HTML includes, 8-8
new component, 5-1
static table, 5-14
custom components
understanding, 3-1, 4-1
Custom Reports component, 8-27
custom resource files, 3-11
custom resources, 5-2

D

DataBinder, 3-8
DataSources table, 9-21
default HTML editor, configuring, 5-20
development
instance, 4-2
directories
naming conventions, 4-4
organization, 4-3
disabling
component, 5-18, 6-3
double-byte characters, 3-2
dynamic resource tables, 9-18, 9-19, 9-37
dynamic table resource, 8-8
dynamic tables
creating, 8-8
editing, 8-8

E

Edit Action screen, 9-10
Edit HTML Resource Include/String screen, 9-25
Edit Intradoc Template screen, 9-38
Edit Parameter screen, 9-26
Edit Query screen, 9-27
Edit SearchResults Template screen, 9-35

Edit Service screen, 9-31
editing
dynamic table, 8-8
environment resource, 8-34
HTML includes, 8-3, 8-8
Readme file, 5-3
service resource, 8-23
static table, 8-9
template resource, 8-32
enabling
components, 5-18, 6-3
environment, 3-8, 3-9
environment resources, 8-32
editing, 8-34
example, 8-33
error message, 8-15
error message attribute, 8-14
example
creating component, 5-3
examples
ClassAliases ResultSet, 7-8
component definition file, 3-6, 7-3
components HDA file, 7-1
environment resource, 8-33
Filters ResultSet, 7-8
glue file, 3-6, 7-3
HDA file, 3-2
HTML includes, 8-2
LocalData section, 3-3
MergeRules ResultSet, 7-7, 8-24
Properties section, 3-3
report page, 8-29
ResourceDefinition ResultSet, 7-5
ResultSet section, 3-4
service actions, 8-18
service attributes, 8-17
service resource, 8-13, 8-16
super tag, 8-3
template page, 8-28
existing component, adding, 5-21

F

file types, 2-4
files
component definition, 3-6
component Zip, 3-11
components HDA, 3-11, 7-1
configuration, 8-32
custom resource, 3-11
environment, 8-32
glue, 3-6
HDA, 3-1
manifest, 3-9
naming conventions, 4-4
organization, 4-3
Readme, 5-3
search_results.htm, 8-25
working with in components, 4-2
Filters ResultSet, 7-8
format
action, 8-14

G

glue file, 3-6

H

HDA files, 3-1
components HDA, 7-1
example, 3-2
ResultSet section, 3-3
Help menu, 9-6
HTML editor, configuring default, 5-20
HTML includes, 8-2
creating, 8-8
editing, 8-3, 8-8
example, 8-2
standard, 8-2, 8-4
super tag, 8-2, 8-2

I

IdocScriptExtensions table, 9-21
IgnoredFlexFields table, 9-19
include resources
adding, 9-25
editing, 9-25
Intradoc template, 9-38
IntradocReports ResultSet, 3-5
IntradocReports table, 9-21
IntradocTemplates ResultSet, 3-5
IntradocTemplates table, 9-24

J

Java code, 5-2

L

LocalData, 3-8, 3-9
LocalData section, 3-3

M

manifest file, 3-9
manifest path, 9-44
Manifest ResultSet, 3-10
manifest.hda, 9-44
manifest.hda file, 3-10
MergeRules ResultSet, 7-6
    template resource, 8-24
    toTable column, 7-7

N

name/value pair, 8-32
naming conventions, 4-4
new component
    creating, 5-1
non-active ResultSets, 3-8

O

opening
    component, 5-19
organization
    component files, 4-3
Overview
    Conventions, 1-3

P

packaging, 9-44, 9-47
pages
    report, 8-23, 8-26
    template, 8-23, 8-26
parameters
    action, 8-15
    adding, 9-26, 9-27
    editing, 9-26, 9-27
predefined dynamic tables, 9-19
predefined resources, 9-30
predefined ResultSets, 3-4
predefined service actions, 9-11
predefined static tables, 9-21
predefined template tables, 9-24

Q

query resources, 8-9

R

Readme file, 5-3
removing
    component, 5-19
report pages, 8-23, 8-26
    example, 8-29
reports
    custom, 8-27
resource categories, 8-1
Resource Selection Dialog screens, 9-30
ResourceDefinition ResultSet, 3-5, 7-5
    columns, 7-5
resources, 8-1
    component, 9-29
    custom, 3-11, 5-2
    dynamic table, 8-8
    environment, 8-32
    predefined, 9-30
    query, 8-9
    service, 8-12
    static table, 8-9
    string, 8-4
ResultSet section, 3-3
ResultSets, 3-9
    ClassAliases, 7-8
    Components, 3-4, 7-2
    Filters, 7-8
    IntradocReports, 3-5
    IntradocTemplates, 3-5
    Manifest, 3-10
    MergeRules, 7-6
    non-active, 3-8
    predefined, 3-4
    ResourceDefinition, 3-5, 7-5
    SearchResultTemplates, 3-5

S

search_results.htm file, 8-25
SearchResults template, 9-35
SearchResultTemplates ResultSet, 3-5
SearchResultTemplates table, 9-24
sections
    LocalData, 3-3
    ResultSet, 3-3
service attributes
    access level, 8-13
    error message, 8-14
    service class, 8-13
    service type, 8-14
    subjects notified, 8-14
    template page, 8-13

service class attribute, 8-13
service definition table, 8-13
service resource
    attributes, 8-17
service resources, 8-12, 9-10, 9-17
    action, 9-10
    actions, 8-18
    adding, 9-31
    editing, 8-23, 9-31
    example, 8-13, 8-16
    subjects, 9-33
service ResultSet
    Actions column, 8-14
service type attribute, 8-14
ServiceHandlers table, 9-21
services
    actions, 8-14
special characters
    in strings, 8-4
standard report pages, 8-26
standard template pages, 8-26
StandardResults template, 8-25
static resource tables, 9-20, 9-37
static tables, 8-9, 9-21
    creating, 5-14
    editing, 8-9
string resources
    adding, 9-25
    editing, 9-25
strings
    resource files, 8-4
    special characters, 8-4
    structure, 8-4
structure
    files and directories, 4-3
subjects, 9-33
subjects notified attribute, 8-14
SubscriptionTypes table, 9-22
super tag, 8-2
syntax
    service action, 8-14

T

template page attribute, 8-13
template pages, 8-23, 8-26
    example, 8-28
template resources, 8-23, 9-23
    adding, 9-35, 9-38
    editing, 8-32, 9-35, 9-38
    MergeRules ResultSet, 8-24
template tables, 9-24
text editor, 4-2
toTable column, 7-7
tutorial
    creating component, 5-3

U

understanding
    custom components, 3-1, 4-1
Unpackage screen, 9-8
unpackaging a component, 5-20
uploading a component, 6-4
UserMetaDefinition table, 9-22
using
    Component Manager, 6-3
    Component Wizard, 5-1

V

variables
    configuration, 8-32
    environment, 8-32

W

Web Layout Editor, 8-25
working with
    components, 6-1, 9-1
working with component files, 4-2
working with Java code, 5-2
working with resources, 5-2

Z

Zip file, 3-11, 5-15