One Design
One Server
One User Experience

Actuate BIRT Java Components Developer Guide

Contains information proprietary to:
Actuate Corporation, 2207 Bridgepointe Parkway, San Mateo, CA 94404

www.actuate.com
www.birt-exchange.com

Document No. 110812-2-771302 August 24, 2011

Contents

About *Actuate BIRT Java Components Developer Guide*

Actuate BIRT Java Components Developer Guide is a guide to designing, deploying and accessing custom reporting web applications using Actuate Java Component.

Actuate BIRT Java Components Developer Guide includes the following chapters:

- *About Actuate BIRT Java Components Developer Guide.* This chapter provides an overview of this guide.
- *Part 1. Customizing an Actuate Java Component.* This part describes how to use Java Component and how to customize its appearance and layout.
- *Chapter 1. Introducing Actuate Java Components.* This chapter introduces Actuate Java Component web applications and explains how Java Components work.
- *Chapter 2. Deploying Actuate BIRT reports using an Actuate Java Component.* This chapter explains how to publish and support BIRT reports and features using Java Components.
- *Chapter 3. Creating a custom Java Component web application.* This chapter explains how to work with Java Component JSP files to design custom reporting web applications.
- *Part 2. Actuate Java Component Reference.* This part describes the code components that make up Java Component, such as URIs, JavaScript files, servlets, tags, beans, and security facilities.
- *Chapter 4. Actuate Java Component configuration.* This chapter describes the Java Component configuration files and how to use them.
- *Chapter 5. Actuate Java Component URIs.* This chapter describes the Java Component JSPs and URL parameters.
- *Chapter 6. Actuate Java Component JavaScript.* This chapter describes the Java Component JavaScript files.

- *Chapter 7. Actuate Java Component servlets.* This chapter describes the Java Component Java servlets.
- *Chapter 8. Actuate Java Component JavaBeans.* This chapter lists the Java Component JavaBeans.
- *Chapter 9. Using Actuate Java Component security.* This chapter introduces the iPortal Security Extension (IPSE) and explains how to use it.
- *Chapter 10. Customizing Java Component online help.* This chapter describes how to customize the Java Component online help files.

Part One

Customizing an Actuate Java Component

Chapter

1

Introducing Actuate Java Components

This chapter contains the following topics:

- About Actuate Java Components
- About Actuate Java Component architecture

About Actuate Java Components

Actuate Java Component is a web application that supports accessing and working with report information using a web browser. Web developers and designers use Actuate Java Component's industry-standard technology to design custom e.reporting web applications to meet business information delivery requirements.

Actuate Java Component technology is platform-independent and customizable. By separating user interface design from content generation, Java Components ensures that reporting web application development tasks can proceed simultaneously and independently. You deploy Actuate Java Component on a web or application server. Java Component accesses documents in a file system repository. Actuate Java Component technology is also scalable.

When deployed, the context root is name of the web archive (.war) or engineering archive (.ear) file without the file extension. For example, if your web archive (.war) file were named DeploymentKit.war, the URL to access the application is:

```
http://<web server>:<port>/DeploymentKit/
```

The context root for Java Component is the root directory of the web archive (.war) file when it is extracted.

Actuate Java Component technology includes the following features:

- JavaServer Pages (JSPs) support creating HTML or XML pages that combine static web page templates with dynamic content.
- Simple Object Access Protocol (SOAP) standards provide plain text transmission of XML using HTTP.
- Report designs and documents are stored on a file system.
- Secure HTTP (HTTPS) supports secure information transfer on the web.
- JSR 168 compliant portlets provide access to reports through portal servers that support the JSR 168 standard.

Licensing Java Components

Java Components have a temporary license by default. To fully license the Java Component you have purchased, you must move the license file received from actuate into the <context root>\WEB-INF directory of the web archive (.war) file.

How to license Java Component

1. Rename the Java Component license file that Actuate sent you to ajclicense.xml.

2. Create a temporary directory, such as C:\Temp\jc on a Microsoft Windows server or /temp/jc on a UNIX server. If you use an existing directory, ensure that this directory is empty.
3. Extract the contents of the Java Component WAR file into a temporary directory.
   - On a Windows server, open a command window and type the following commands, replacing the E: DVD drive letter with the path of your Java Component WAR file:

     ```
     cd C:\Temp\jc
     copy E:\ActuateJavaComponent.war
     jar -xf ActuateJavaComponent.war
     ```

     The Java Component files appear in the temporary directory. Leave the command window open.
   - On a LINUX or UNIX server, type the following commands, replacing the DVD drive name with the path of your Java Component WAR file:

     ```
     cd /temp/jc
     cp /dev/dsk/cd/ActuateJavaComponent.war  .
     jar -xf ActuateJavaComponent.war
     ```

     The Actuate Java Component files appear in the temporary directory.
4. Copy the ajclicense.xml file into the extracted <context root>\WEB-INF directory.
5. Type the following command:

   ```
   jar -cf ..\DeploymentKit.war *
   ```

   This command creates DeploymentKit.war in the parent directory. This new Java Component WAR file contains the license.
6. Deploy the DeploymentKit.war file to the application server or servlet engine as an application.
7. Restart the application server or servlet engine.

Setting up Actuate Java Component

To deploy a report to the web, you need:

- An Actuate Java Component installation.
- An application server or JSP or servlet engine such as Actuate embedded servlet engine or IBM WebSphere.
- One or more Actuate designer tools.
- Permission to read, write, and modify operating system directories as necessary. For example, the directory Java uses to hold temporary files is

defined by the java.io.tmpdir property and is by default the value of the TMP system variable in the Windows environment and /var/tmp in the UNIX and LINUX environments. Read and write permission must be provided to the application server running Information Console for this directory.

For more information about installing Java Component, see *Installing an Actuate Java Component*.

Customizing Java components for installation

When you deploy Java Components on an application server, you can use a customized Java Component application. To do this, you need to extract the contents of the Actuate Java Components WAR or EAR file and customize the files directly. After you customize the system, recreate a WAR or EAR file using the Java jar utility and redeploy it to your application server. The customizations can include any modifications of JavaScript, Java Server Pages (JSP) and other web pages, and skins. Later chapters in this book provide detailed information about customizing JavaScript and JSPs.

When Actuate Java Component is deployed, you cannot further customize skins, add pages, or make any other changes that affect the Actuate Java Component file structure without extracting the contents of the WAR or EAR file, modifying the contents, and re-deploying it.

Clustered Actuate Java Component instances can use a third-party application to balance the load among the application servers. Actuate Java Component supports third-party load balancing, as illustrated in Figure 1-1, to ensure high availability and to distribute tasks for efficient processing.

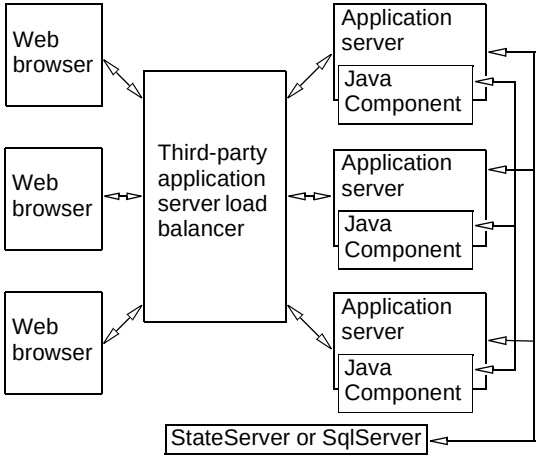

Figure 1-1 Load-balancing architecture for Java Component

About using a cluster of application servers

If the application servers running Java Component support session state management, you can configure Actuate Java Component and the application servers to share and maintain a web browsing session state across a cluster of Java Component instances.

How to customize and deploy Actuate Java Component

To customize Actuate Java Component and deploy it to application servers in a clustered environment, use the following general procedure.

1. Extract the contents of the Actuate Java Component WAR file into a temporary directory.
2. Customize the Actuate Java Component JavaScript, skins, and web pages as desired.
3. Save all files and archive Actuate Java Components as a new WAR or EAR file using the Java jar utility.
4. Deploy the WAR or EAR file to each machine in your cluster.

About Actuate Java Component architecture

This section describes the general operation, authentication, and structure of Java Component as a web application.

The Actuate Java Component architecture is illustrated in Figure 1-2.

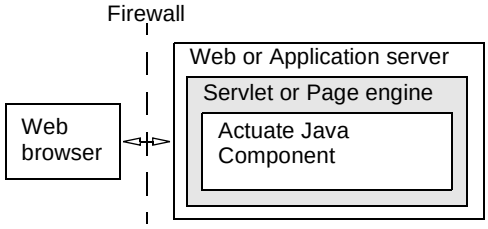

Figure 1-2 Actuate Java Component architecture overview

A user submits a request by choosing a link that specifies an Actuate Java Component URI. As shown in Figure 1-2, the web or application server passes the URI to the servlet or page engine, which invokes Actuate Java Component and interprets the URI. The web server returns the results to the web browser. Then, the web browser displays the results for the user.

Actuate Java Component manages requests as part of a JSP engine within a web or application server. See your web or application server documentation for more information on managing the engine.

Using proxy servers with Actuate Java Component

When setting up a proxy server with Actuate Java Component, there are steps you must take if your internal application server port is protected by a firewall. In this situation, when the proxy server changes the URL to point to the new context's port, that port is unavailable due to the firewall. The usual solution is to configure a reverse proxy, but if you are using multiple proxies and a reverse proxy is not practical for your installation, Actuate Java Component can perform the redirection.

To redirect a page without using a reverse proxy, Actuate Java Component forwards the URL to redirect to the processRedirect.jsp page and updates the browser's location bar accordingly. This action processes on the client. The browser takes the current URL location and updates the rest of the URI using the redirected URL. You must also set the ENABLE_CLIENT_SIDE_REDIRECT configuration parameter to true and modify the redirect attributes in the <context root>/WEB-INF/struts-config.xml file. The necessary modifications are included in the file. You just need to comment out the lines that have the redirect attribute set to true and uncomment the lines that forward to the processRedirect.jsp page.

For example, the following code is the struts-config.xml entry for the login action. By default the forward statement for success points to getfolderitems.do with the redirect attribute set to true. This code instructs the application server to send a redirect with the getfolderitems.do URL when the user logs in.

```
<!-- Process a user login -->
<action
   path="/login"
   name="loginForm"
   scope="request"
   input="/iportal/activePortal/private/login.jsp"
   type="com.actuate.activeportal.actions.AcLoginAction"
   validate="false">
   <forward name="loginform"
      path="/iportal/activePortal/private/login.jsp" />
   <!--
      <forward name="success"
         path="/iportal/activePortal/private/common
         /processredirect.jsp?redirectPath=/getfolderitems.do" />
   -->
   <forward name="success" path="/getfolderitems.do"
      redirect="true" />
   <forward name="landing" path="/landing.jsp"
      redirect="false" />
</action>
```

From behind a firewall and proxy, this redirect will fail because the redirect sent by the application server points to the application server port instead of the firewall and proxy port. For this redirect method to operate behind a firewall, you need to comment out the line that has redirect="true" and uncomment the line

that points to processRedirect.jsp. The following code shows the updated entry in struts-config.xml:

```
<!-- Process a user login -->
<action
   path="/login"
   name="loginForm"
   scope="request"
   input="/iportal/activePortal/private/login.jsp"
   type="com.actuate.activeportal.actions.AcLoginAction"
   validate="false">
   <forward name="loginform"
      path="/iportal/activePortal/private/login.jsp" />
   <forward name="success"
      path="/iportal/activePortal/private/common
      /processredirect.jsp?redirectPath=/getfolderitems.do" />
   <!--
      <forward name="success" path="/getfolderitems.do"
         redirect="true" />
   -->
   <forward name="landing" path="/landing.jsp"
      redirect="false" />
</action>
```

This change needs to be made for all the actions in struts-config.xml that send a redirect to the browser.

About Actuate Java Component pages

Actuate Java Component uses JSPs to generate web pages dynamically before sending them to a web browser. These JSPs use custom tags, custom classes, and JavaScript to generate dynamic web page content. The JavaScript, classes, and tags provide access to other pages, JavaBeans, and Java classes. For example, application logic in Actuate Java Component can reside on the web server in a JavaBean.

Web browsers can request a JSP with parameters as a web resource. The first time a web browser requests a page, the page is compiled into a servlet. Servlets are Java programs that run as part of a network service such as a web server. Once a page is compiled, the web server can fulfill subsequent requests quickly, provided that the page source is unchanged since the last request.

The filesfolders JSPs support accessing repository files and folders. These JSPs reside in <context root>\iportal\activePortal\private\filesfolders.

The submit request JSPs support submitting new jobs. The submit request JSPs reside in <context root>\iportal\activePortal\private\newrequest. For specific information about running jobs using Actuate Java Component, see *Using Actuate BIRT Java Components*.

The viewing JSPs support the following functionality, according to report type:

- Searching report data
- Using a table of contents to navigate through a report
- Paginating or not paginating a report
- Fetching reports in supported formats

For specific information about viewing reports using Actuate Java Component, see *Using Actuate BIRT Java Components.*

Use the default pages, customize the pages, or create entirely new pages to deploy your reporting web application.

Working with Actuate Java Component URIs

Actuate Java Component Uniform Resource Identifiers (URIs) convey user requests to an application server. URIs access functionality including generating reports, managing repository contents, and viewing reports.

About Actuate Java Component URIs

Actuate Java Component URIs consist of the context root and port of the web server where you install and deploy the JSPs or servlets. Actuate Java Component URIs have the following syntax:

```
http://<web server>:<port>/<context root>
   /<path><page>.<type>[?<parameter=value>{&<parameter=value>}]
```

where

- <web server> is the name of the machine running the application server or servlet engine. You can use localhost as a trusted application's machine name if your local machine is running the server.
- <port> is the port on which you access the application server or servlet engine.
- <context root> is the context root for accessing the Actuate Java Component pages, which by default is the name of the WAR or EAR file.
- <path> is the directory containing the page to invoke.
- <page> is the name of the page or method.
- <type> is jsp or do.
- <parameter=value> specifies the required parameters and values for the page.

For example, to view the document list page, Actuate Java Component uses a URI with the following format:

```
http://<web server>:<port>/ActuateJavaComponent
   /getfolderitems.do?doframe=true&userid=anonymous
```

where

- ActuateJavaComponent/getfolderitems.do is the JSP that provides file browsing for Java Component.
- doframe=true is a reserved parameter that displays the documents page in a frame next to other frames for the banner and file explorer tree.
- userid=anonymous indicates that the default anonymous user is being used and security is not enabled. This is the default security setting for Actuate Java Components. For information about customizing security, see Chapter 9, "Using Actuate Java Component security."

Using a special character in a URI

Actuate Java Component URIs use encoding for characters that a browser can misinterpret. You use hexadecimal encoding in these circumstances to avoid misinterpretation. Use the encoding only when the possibility of misinterpreting a character exists. Always encode characters that have a specific meaning in a URI when you use them in other ways. Table 1-1 describes the available character substitutions. An ampersand introduces a parameter in a URI, so you must encode an ampersand that appears in a value string. For example, use:

```
&company=AT%26T
```

instead of:

```
&company=AT&T
```

Table 1-1 Encoding sequences for use in URIs

Character	Encoded substitution
ampersand (&)	%26
asterisk (*)	%2a
at (@)	%40
backslash (\)	%5c
colon (:)	%3a
comma (,)	%2c
dollar sign ($)	%24
double quote (")	%22
equal (=)	%3d
exclamation (!)	%21
greater than (>)	%3e
less than (<)	%3c

(continues)

Table 1-1 Encoding sequences for use in URIs (continued)

Character	Encoded substitution
number sign (#)	%23
percent (%)	%25
period (.)	%2e
plus (+)	%2b
question mark (?)	%3f
semicolon (;)	%3b
slash (/)	%2f
space ( )	%20
underscore (_)	%5f

If you customize Actuate Java Component by writing code that creates URI parameters, encode the entire parameter value string with the encode() method. The encode( ) method is included in encoder.js, which is provided in the Actuate Java Component <context root>/js directory. The following example encodes the folder name /Training/Sub Folder before executing the getFolderItems action:

```
<%-- Import the StaticFuncs class. --%>
<%@ page import="com.actuate.reportcast.utils.*" %>
<%
   String url =
   "http://localhost:8080/ActuateJavaComponent/getfolderitems.do
   ?folder=" + StaticFuncs.encode("/Training/Sub Folder");
   response.sendRedirect(url);
%>
```

The encode( ) method converts the folder parameter value from:

```
/Training/Sub Folder
```

to:

```
%2fTraining%2fSub%20Folder
```

About UTF-8 encoding

UTF-8 encoding is also the default encoding that web browsers support. All Java Component communication also uses UTF-8 encoding. For 8-bit (single byte) characters, UTF-8 content appears the same as ANSI content. If, however, extended characters are used (typically for languages that require large character sets), UTF-8 encodes these characters with two or more bytes.

Chapter

2

Deploying Actuate BIRT reports using an Actuate Java Component

This chapter contains the following topics:

- Publishing a BIRT report design to the Actuate Java Component
- Using fonts
- Using BIRT encryption
- Deploying custom emitters

Publishing a BIRT report design to the Actuate Java Component

Actuate Java Components generate BIRT reports using BIRT report design (.rptdesign) files and their associated resource files. Actuate Java Components access BIRT report design and associated resource files from configurable locations on a file system.

The default location designated for BIRT report design files is the repository folder in the context root directory structure, as illustrated in Figure 2-1.

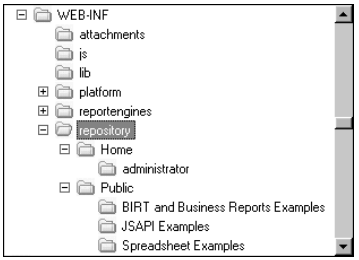

Figure 2-1 Actuate Java Component folder structure

To configure the repository location for publishing BIRTdesigns and documents, change the value of the STANDALONE_REPOSITORY_PATH parameter in the Actuate Java Component's web.xml file. The web.xml file is in the following location:

```
<context root>/WEB-INF
```

The following code sets STANDALONE_REPOSITORY_PATH to the <context root>/WEB-INF/repository subfolder:

```
<context-param>
   <param-name>STANDALONE_REPOSITORY_PATH</param-name>
   <param-value>WEB-INF/repository</param-value>
</context-param>
```

BIRT_RESOURCE_PATH specifies the path to the shared resources for Actuate BIRT Java Components, including libraries, templates, properties, and Java archive (.jar) files for BIRT report designs. The default value is <context root> /WEB-INF/repository.

How to publish a BIRT report design to an Actuate Java Component

This procedure uses the default location of the Actuate Java Component repository.

1. Navigate to the application server's directory for deployed web applications. For example, Apache Tomcat stores web applications in <Apache Tomcat root directory>/Tomcat 6.0/webapps.

2 In the web application directory, manually copy the BIRT report design to a directory in the following location:

```
<context root>/WEB-INF/repository
```

The installation provides default home and public directories, as shown in Figure 2-1. All user directories are created in the repository/home directory.

3 To make a report design available to all users, place the file in a directory within:

```
<context root>/WEB-INF/repository/Public
```

4 To make a report design available to an individual user only, place the file in a directory within:

```
<context root>/WEB-INF/repository/Home/<user name>
```

5 Run the Actuate Java Component to access the report design.

Publishing a BIRT resource to an Actuate Java Component

You configure the repository for publishing a BIRT resource using the BIRT_RESOURCE_PATH parameter in an Actuate Java Component's web.xml file. The web.xml file is in the following location:

```
<context root>/WEB-INF
```

The following code sets BIRT_RESOURCE_PATH to the <context root>/resources subfolder:

```
<context-param>
   <param-name>BIRT_RESOURCE_PATH</param-name>
   <param-value>resources</param-value>
</context-param>
```

BIRT_RESOURCE_PATH specifies the path to the shared resources for Actuate BIRT Java Components, including libraries, templates, properties, and Java archive (.jar) files for BIRT report designs. The default value is <context root>/resources.

If the BIRT report explicitly includes a resource such as a JAR file, library, CSS, a Flash (.swf) file, images, or JavaScript in the report design, then the resources need to be copied under the BIRT_RESOURCE_PATH folder to the correct relative path.

For example, if the images for your report are in the /images folder in your report design project, when you deploy the report, you copy the images to the <context root>/resources/images folder.

In cases when an Actate BIRT report uses Java classes directly from JAR files, copy your JAR files to:

```
<context root>/scriptlib
```

How to publish a BIRT resource to an Actuate Java Component

1. Copy the resource file to the resource directory, defined in web.xml.
2. To test the resource, run the Actuate Java Component to execute and view a report that uses the resource.

Installing a custom JDBC driver in an Actuate Java Component

When you use an Actuate Java Component and an Actuate BIRT report uses a custom JDBC driver, you must install the JDBC driver in the following location:

```
<context root>/WEB-INF/platform/plugins/
   org.eclipse.birt.report.data.oda.jdbc_<VERSION>/drivers
```

Installing custom ODA drivers and custom plug-ins in an Actuate Java Component

All custom ODA drivers and custom plug-ins need to be installed in the following folder:

```
<context root>/WEB-INF/platform/plugins
```

Accessing BIRT report design and BIRT resources paths in custom ODA plug-ins

ODA providers often need to obtain information about a resource path defined in ODA consumer applications. For example, if you develop an ODA flat file data source, you can implement an option to look up the data files in a path relative to a resource folder managed by its consumer. Such resource identifiers are needed at both design-time and run-time drivers. ODA consumer applications are able to specify the following items as described in the next two sections:

- The run-time resource identifiers to pass o the ODA run-time driver in an application context map
- The design-time resource identifiers in a DataSourceDesign, as defined in an ODA design session model

Accessing resource identifiers in run-time ODA driver

For run time, the BIRT ODA run-time consumer passes its resource location information in a org.eclipse.datatools.connectivity.oda.util.ResourceIdentifiers instance in the appContext map. ODA run-time drivers can get the instance in

any one of the setAppContext methods, such as IDriver.setAppContext. You can use resource identifiers to perform the following tasks:

- To get the BIRT resource folder URI, call getApplResourceBaseURI( ) method.
- To get the instance from the appContext map, pass the map key ResourceIdentifiers.ODA_APP_CONTEXT_KEY_CONSUMER_RESOURCE_IDS, defined by the class as a method argument.
- To get the URI of the associated report design file folder, call getDesignResourceBaseURI( ) method. The URI is application dependent and it can be absolute or relative. If your application maintains relative URLs, call the getDesignResourceURILocator.resolve( ) method to get the absolute URI.

The code snippet on Listing 2-1 shows how to access the resource identifiers through the application context.

Listing 2-1 Accessing resource identifiers at run time

```
URI resourcePath = null;
URI absolutePath = null;

Object obj = this.appContext.get(
   ResourceIdentifiers.ODA_APP_CONTEXT_KEY_CONSUMER_RESOURCE_IDS
   );
if ( obj != null )
{
   ResourceIdentifiers identifier = (ResourceIdentifiers)obj;
   if ( identifier.getDesignResourceBaseURI( ) != null )
   {
      resourcePath = identifier.getDesignResourceBaseURI( );

      if ( ! resourcePath.isAbsolute( ) )
         absolutePath =
            identifier.getDesignResourceURILocator( ).resolve(
            resourcePath );
      else
         absolutePath = resourcePath;
   }
}
```

Accessing resource identifiers in design ODA driver

The resource identifiers are available to the custom ODA designer UI driver. The designer driver provides the user interface for a custom data source and data set.

Typically, to implement a custom behavior, the data source UI driver extends:

```
org.eclipse.datatools.connectivity.oda.design.ui.wizards.
   DataSourceWizardPage
```

The DataSourceWizardPage class has an inherited method getHostResourceIdentifiers( ) that provides access to the resource and report paths. The extended DataSourceWizardPage just needs to call the base method to get the ResourceIdentifiers for its paths information.

Similarly, if the custom driver implements a custom data source editor page, it extends:

```
org.eclipse.datatools.connectivity.oda.design.ui.wizards.
   DataSourceEditorPage
```

The DataSourceEditorPage class has an inherited method getHostResourceIdentifiers( ). The extended class needs to call the base class method to get the ResourceIdentifiers object for the two resource and report paths base URIs.

Related primary methods in the org.eclipse.datatools.connectivity.oda.design. ResourceIdentifiers are:

- URI getDesignResourceBaseURI( );
- URI getApplResourceBaseURI( );

Using fonts

Java Components supports rendering BIRT reports in different formats such as PDF, Microsoft Word, Postscript, and PowerPoint. The conversion processes use the fonts installed on your system to display the report characters by default.

BIRT uses a flexible mechanism that supports configuring font usage and substitution. This mechanism uses font configuration files for different purposes that control different parts of the rendering process. The configuration files can configure the fonts used in specific operating systems, in specific formats, in specific locales, or combinations of these parameters, as described in the next section.

The plug-in folder, org.eclipse.birt.report.engine.fonts, contains the font configuration files. Table 2-1 shows the location of this folder in the supported BIRT environments.

Table 2-1 Locations of the font configuration file plug-in folder

Environment	Font configuration file folder location
Actuate Java Components	$ActuateJavaComponents/WEB-INF/platform/plugins
BIRT Report Designer	$Actuate11/BRD/eclipse/plugins
BIRT Report Designer Professional	$Actuate11/BRDPro/eclipse/plugins

Understanding font configuration file levels and priorities

BIRT reports use five different types of font configuration files. The font configuration file naming convention includes information about the rendering format, the system platform, and the system locale, as shown in the following template:

```
fontsConfig_<Format>_<Platform>_<Locale>.xml
```

The platform name is defined by the Java System property, os.name. The following code shows how to check the os.name property for the proper value in your configuration:

```
System.getProperty("os.name");
```

Table 2-2 lists the supported values for the three properties that form the font configuration file name. The platform property in this table shows the values that Sun Microsystems uses for the os.name property.

Table 2-2 Font configuration file name properties

Format	Platform	Locale
pdf	Windows_Vista	en
ppt	Windows_2003	fr
html	Windows_XP	de
postscript	Windows_2000	it
doc	SunOS	ja
	AIX	ko
	HP-UX	zh
	Linux	zh_Hans
		zh_Hant
		fr_FR
		de_DE
		it_IT
		ja_JP
		ko_KR
		zh_Hans_CN
		zh_Hant_TW
		en_GB

(continues)

Table 2-2 Font configuration file name properties (continued)

Format	Platform	Locale
doc *(continued)*	Linux *(continued)*	en_US
		en_CA

BIRT supports the following levels of font configuration files, with increasing priority:

- For all rendering formats
  These files have no format specifier in their names. These configuration files are divided into three sub-levels:
  - The default configuration file:

    ```
    fontsConfig.xml
    ```

  - Configuration files for a specific platform, for example:

    ```
    fontsConfig_Windows_XP.xml
    ```

  - Configuration files for a specific platform and locale, for example:

    ```
    fontsConfig_Windows_XP_zh.xml
    fontsConfig_Windows_XP_zh_CN.xml
    ```

- For certain formats only
  These files include the format specifier in their names. These configuration files are divided into three sub-levels:
  - The default configuration file for a format, for example:

    ```
    fontsConfig_pdf.xml
    ```

  - Configuration files for a format for a specific platform:

    ```
    fontsConfig_pdf_Windows_XP.xml
    ```

Understanding how BIRT accesses a font

The PDF layout engine renders the PDF, Postscript, and PowerPoint formats. The engine tries to use the font specified at design time to render. The PDF layout engine searches for the font files first in the fonts folder of the plug-in, org.eclipse.birt.report.engine.fonts. If the fonts are not in this folder, the engine searches for the font in the system-defined font folder. Change the default load order by using the settings in the font configuration file.

When the required font for a character is not available in the search path or is incorrectly installed, the engine uses the fonts defined in the UNICODE block for that character. If the UNICODE definition also fails, the engine replaces the character with a question mark (?) to denote a missing character. The font used for the ? character is the default font, Times-Roman.

The engine maps the generic family fonts to a PDF embedded Type1 font, as shown in the following list:

- cursive maps to Times-Roman
- fantasy maps to Times-Roman
- monospace maps to Courier
- sans-serif maps to Helvetica
- serif maps to Times-Roman

Understanding the font configuration file structure

The font configuration file, fontsConfig.xml, consists of three major sections, <font-aliases>, <composite-font>, and <font-paths> sections.

<font-aliases> section

In <font-aliases> section, you can:

- Define a mapping from a generic family to a font family. For example, the following defines a mapping from generic family "serif" to Type1 font family "Times-Roman":

  ```
  <mapping name="serif" font-family="Times-Roman"/>
  ```

- Define a mapping from a font family to another font family. This is useful if you want to use a font for PDF rendering that differs from the font used in design-time. For example, the following shows how to replace "simsun" with "Arial Unicode MS":

  ```
  <mapping name="simsun" font-family="Arial Unicode MS"/>
  ```

Previous versions of the BIRT Report Designers use the XML element <font-mapping> instead of <font-aliases>. In the current release, a <font-mapping> element works in the same way as the new <font-aliases> element. When a font configuration file uses both <font-mapping> and <font-aliases>, the engine merges the different mappings from the two sections. If the same entries exist in both sections, the settings in <font-aliases> override those in <font-mapping>.

<composite-font> section

The <composite-font> section defines a composite font. A composite font is a font consisting of many physical fonts used for different characters. The composite fonts are defined by <block> entries. Each <block> entry defines a mapping from a UNICODE range to a font family name, which means the font family is applied for the UNICODE characters in that range. You cannot change the block name or range or index as it is defined by the UNICODE standard. The only item you can

change in the block element is the font family name. To find information about all the possible blocks, go to http://www.unicode.org/charts/index.html.

A composite font named all-fonts is applied as a default font. When a character is not defined in the desired font, the font defined in all-fonts is used.

For example, to define a new font for currency symbols, you change font-family in the following <block> entry to the Times Roman font-family:

```
<composite-font>
…
<block name="Currency Symbols" range-start="20a0" range-end="20cf"
   index="58" font-family="Times Roman" />
…
</composite-font>
```

In cases when the Times Roman font does not support all the currency symbols, you can define the substitution character by character using the <character> tag, as shown in the following example:

```
<composite-font>
    …
      <character value="?" font-family="Angsana New"/>
      <character value="\u0068" font-family="Times Roman"/>
    …
</composite-font>
```

Note that characters are represented by the attribute, value, which can be presented two ways, the character itself or its UNICODE code.

To find information about all the currency symbols, go to http://www.unicode.org/charts/symbols.html.

<font-paths> section

If the section <font-paths> is set in fontsConfig.xml, the engine ignores the system-defined font folder, and loads the font files specified in the section, <font-paths>. You can add a single font path or multiple paths, ranging from one font path to a whole font folder, as shown in the following example:

```
<path path="c:/windows/fonts"/>
<path path="/usr/X11R6/lib/X11/fonts/TTF/arial.ttf"/>
```

If this section is set, the PDF layout engine will only load the font files in these paths and ignore the system-defined font folder. If you want to use the system font folder as well, you must include it in this section.

On some systems, the PDF layout engine does not recognize the system-defined font folder. If you encounter this issue, add the font path to the <font-paths> section.

Using BIRT encryption

BIRT provides an extension framework to support users registering their own encryption strategy with BIRT. The model implements the JCE (Java™ Cryptography Extension). The Java encryption extension framework provides multiple popular encryption algorithms, so the user can just specify the algorithm and key to have a high security level encryption. The default encryption extension plug-in supports customizing the encryption implementation by copying the BIRT default plug-in, and giving it different key and algorithm settings.

JCE provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects.

A conventional encryption scheme has the following five major parts:

- Plaintext, the text to which an algorithm is applied.
- Encryption algorithm, the mathematical operations to conduct substitutions on and transformations to the plaintext. A block cipher is an algorithm that operates on plaintext in groups of bits, called blocks.
- Secret key, the input for the algorithm that dictates the encrypted outcome.
- Ciphertext, the encrypted or scrambled content produced by applying the algorithm to the plaintext using the secret key.
- Decryption algorithm, the encryption algorithm in reverse, using the ciphertext and the secret key to derive the plaintext content.

About the BIRT default encryption plug-in

BIRT's default encryption algorithm is implemented as a plug-in named:

`com.actuate.birt.model.defaultsecurity_11.0.1`

Table 2-3 shows the location of this plug-in folder in the supported BIRT environments.

Table 2-3 Locations of the default encryption plug-in folder

Environment	Font configuration file folder location
Actuate Java Components	$ActuateJavaComponents/WEB-INF/platform/plugins
BIRT Report Designer	$Actuate11/BRD/eclipse/plugins
BIRT Report Designer Professional	$Actuate11/BRDPro/eclipse/plugins

Deploying encryption plug-ins to Actuate Java Components

If you use Java Components, you deploy all new encryption plug-ins to the Java Components plug-in folder. The BIRT report engine decrypts the encrypted report data during report generation. To do the decryption, it must have access to all encryption plug-ins. The report engine loads all encryption plug-ins at start up. When the engine runs a BIRT report, it reads the encryptionID property from the report design file and uses the corresponding encryption plug-in to decrypt the encrypted property. Every time you create reports using a new encryption plug-in, make sure you deploy the plug-in to Java Components installation, otherwise the report execution will fail.

How to deploy a new encryption plug-in instance to Actuate Java Components

1. Extract the Java Components WAR or EAR file into temporary directory.
2. Copy:

   ```
   $ACTUATE_HOME/BRDPro/eclipse/plugins
      /com.actuate.birt.model.defaultsecurity_11.0.1_rsa
   ```

   to:

   ```
   <context root>/WEB-INF/platform/plugins
   ```

3. Copy your report design to:

   ```
   <context root>/WEB-INF/repository/home/<UserHomeFolder>
   ```

4. Recompress your Java Components WAR file using the Java jar utility and redeploy it to the application server or servlet engine as an application.
5. Restart the application service where the Java Components are deployed, to load the new encryption plug-in.
6. Run your report again. The engine uses the new encryption plug-in to decrypt the password.

About the components of the BIRT default encryption plug-in

The BIRT default encryption plug-in consists of the following main modules:

- acdefaultsecurity.jar
- encryption.properties file
- META-INF/MANIFEST.MF
- plugin.xml

About acdefaultsecurity.jar

This JAR file contains the encryption classes. The default encryption plug-in also provides key generator classes that can create different encryption keys.

About encryption.properties

This file specifies the encryption settings. BIRT loads the encryption type, encryption algorithm, and encryption keys from the encryption.properties file to do the encryption. The file contains pre-generated default keys for each of the supported algorithms.

You define the following properties in the encryption.properties file:

- Encryption type
  Type of algorithm. Specify one of the two values, symmetric encryption or public encryption. The default type is symmetric encryption.
- Encryption algorithm
  The name of the algorithm. You must specify the correct encryption type for each algorithm. For the symmetric encryption type, BIRT supports DES and DESede. For public encryption type, BIRT supports RSA.
- Encryption mode
  In cryptography, a block cipher algorithm operates on blocks of fixed length, which are typically 64 or 128 bits. Because messages can be of any length, and because encrypting the same plaintext with the same key always produces the same output, block ciphers support several modes of operation to provide confidentiality for messages of arbitrary length. Table 2-4 shows all supported modes.

Table 2-4 Supported encryption modes

Mode	Description
None	No mode
CBC	Cipher Block Chaining Mode, as defined in the National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) PUB 81, "DES Modes of Operation," U.S. Department of Commerce, Dec 1980
CFB	Cipher Feedback Mode, as defined in FIPS PUB 81
ECB	Electronic Codebook Mode, as defined in FIPS PUB 81
OFB	Output Feedback Mode, as defined in FIPS PUB 81
PCBC	Propagating Cipher Block Chaining, as defined by Kerberos V4

- Encryption padding
  Because a block cipher works on units of a fixed size, but messages come in a variety of lengths, some modes, for example CBC, require that the final block be padded before encryption. Several padding schemes exist. The supported paddings are shown in Table 2-5. All padding settings are applicable to all algorithms.

Table 2-5 Supported encryption paddings

Mode	Description
NoPadding	No padding.
OAEP	Optimal Asymmetric Encryption Padding (OAEP) is a padding scheme that is often used with RSA encryption.
PKCS5Padding	The padding scheme described in RSA Laboratories, "PKCS #5: Password-Based Encryption Standard," version 1.5, November 1993. This encryption padding is the default.
SSL3Padding	The padding scheme defined in the SSL Protocol Version 3.0, November 18, 1996, section 5.2.3.2.

- Encryption keys
  Actuate provides pre-generated keys for all algorithms.

Listing 2-1 shows the default contents of encryption.properties.

Listing 2-1 Default encryption.properties

```
#message symmetric encryption , public encryption.
type=symmetric encryption

#private encryption: DES(default), DESede
#public encryption:  RSA
algorithm=DES

# NONE , CBC , CFB , ECB( default ) , OFB , PCBC
mode=ECB

# NoPadding , OAEP , PKCS5Padding( default ) , SSL3Padding
padding=PKCS5Padding

#For key , support default key value for algorithm
#For DESede ,DES we only need to support private key
#private key value of DESede algorithm : 20b0020…
#private key value of DES algorithm: 527c2…
#for RSA algorithm, there is a key pair. You should support
   private-public key pair
```

```
#private key value of RSA algorithm: 30820…

#public key value of RSA algorithm: 30819…
#private key
symmetric-key=527c23…

#public key
public-key=
```

About META-INF/MANIFEST.MF

META-INF/MANIFEST.MF is a text file that is included inside a JAR file to specify metadata about the file. Java's default ClassLoader reads the attributes defined in MANIFEST.MF and appends the specified dependencies to its internal classpath.

The encryption plug-in ID is the value of the Bundle-SymbolicName property in the manifest file for the encryption plug-in. You need to change this property when you deploy multiple instances of the default encryption plug-in, as described later in this chapter.

Listing 2-2 shows the contents of the default MANIFEST.MF.

Listing 2-2 Default MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Actuate Default Security Plug-in
Bundle-SymbolicName:
   com.actuate.birt.model.defaultsecurity;singleton:=true
Bundle-Version: 11.0.1.<version>
Require-Bundle: org.eclipse.birt.report.model,
 org.eclipse.core.runtime
Export-Package: com.actuate.birt.model.defaultsecurity.api
Bundle-ClassPath: acdefaultsecurity.jar
Bundle-Vendor: Actuate Corporation
Eclipse-LazyStart: true
Bundle-Activator:
   com.actuate.birt.model.defaultsecurity.properties.
   SecurityPlugin
```

About plugin.xml

plugin.xml is the plug-in descriptor file. This file describes the plug-in to the Eclipse platform. The platform reads this file and uses the information to populate and update, as necessary, the registry of information that configures the whole platform.

The <plugin> tag defines the root element of the plug-in descriptor file. The <extension> element within the <plugin> element specifies the Eclipse extension

point that this plug-in uses, org.eclipse.birt.report.model.encryptionHelper. This extension point requires a sub-element, <encryptionHelper>. This element uses the following attributes:

- class

  The qualified name of the class that implements the interface IEncryptionHelper. The default class name is com.actuate.birt.model.defaultsecurity.api.DefaultEncryptionHelper.

- extensionName

  The unique internal name of the extension. The default extension name is jce.

- isDefault

  Field indicating whether this encryption extension is the default for all encryptable properties. This property is valid only in a BIRT Report Designer environment. When an encryption plug-in sets the value of this attribute to true, the BIRT Report Designer uses this encryption method as the default to encrypt data. There is no default encryption mode in Java Components.

  The encryption model that BIRT uses supports implementing and using several encryption algorithms. The default encryption plug-in is set as default using this isDefault attribute. If you implement several encryptionHelpers, set this attribute to true for only one of the implementations. If you implement multiple encryption algorithms and set isDefault to true to more than one instance, BIRT treats the first loaded encryption plug-in as the default algorithm.

Listing 2-3 shows the contents of the default encryption plug-in's plugin.xml.

Listing 2-3 Default plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
      id="encryption"
      name="default encryption helper"
      point="org.eclipse.birt.report.model.encryptionHelper">
      <encryptionHelper
         class="com.actuate.birt.model.defaultsecurity.api
         .DefaultEncryptionHelper"
         extensionName="jce" isDefault="true" />
   </extension>
```

Deploying multiple encryption plug-ins

In some cases, you need to use an encryption mechanism other than the Data Source Explorer default in your report application. For example, some applications need to create an encryption mechanism using the RSA algorithm

that the default encryption plug-in supports. In this case, you must create an additional encryption plug-in instance. For use within a BIRT Report Designer, you can set this plug-in as the default encryption mechanism. If you change the default encryption mechanism, you must take care when you work with old report designs. For example, if you change an existing password field in the designer, the designer re-encrypts the password with the current default encryption algorithm regardless of the original algorithm that the field used.

How to create a new instance of the default encryption plug-in

1. Make a copy of the default encryption plug-in.
   1. Copy the folder:

      ```
      $ACTUATE_HOME/BRDPro/eclipse/plugins
         /com.actuate.birt.model.defaultsecurity_11.0.1
      ```

   2. Paste the copied folder in the same folder:

      ```
      $ACTUATE_HOME/BRDPro/eclipse/plugins
      ```

   3. Rename:

      ```
      $ACTUATE_HOME/BRDPro/eclipse/plugins/Copy of
         com.actuate.birt.model.defaultsecurity_11.0.1
      ```

      to a new name, such as:

      ```
      $ACTUATE_HOME/BRDPro/eclipse/plugins
         /com.actuate.birt.model.defaultsecurity_11.0.1_rsa
      ```

2. Modify the new plug-in's manifest file.
   1. Open:

      ```
      $ACTUATE_HOME/BRDPro/eclipse/plugins
         /com.actuate.birt.model.defaultsecurity_11.0.1_rsa
         /META-INF/MANIFEST.MF
      ```

   2. Change:

      ```
      Bundle-SymbolicName:
         com.actuate.birt.model.defaultsecurity
      ```

      to:

      ```
      Bundle-SymbolicName:
         com.actuate.birt.model.defaultsecurity.rsa
      ```

      MANIFEST.MF now looks similar to the one in Listing 2-4.

      Listing 2-4 Modified MANIFEST.MF for the new encryption plug-in

      ```
      Manifest-Version: 1.0
      Bundle-ManifestVersion: 2
      ```

      (continues)

```
Bundle-Name: Actuate Default Security Plug-in
Bundle-SymbolicName: com.actuate.birt.model.
     defaultsecurity.rsa;singleton:=true
Bundle-Version: 11.0.1.<version>
Require-Bundle: org.eclipse.birt.report.model,
 org.eclipse.core.runtime
Export-Package: com.actuate.birt.model.defaultsecurity.api
Bundle-ClassPath: acdefaultsecurity.jar
Bundle-Vendor: Actuate Corporation
Eclipse-LazyStart: true
Bundle-Activator: com.actuate.birt.model.defaultsecurity.
     properties.SecurityPlugin
```

3. Save and close MANIFEST.MF.

3 Modify the new plug-in's descriptor file to make it the default encryption plug-in.

1. Open:

```
$ACTUATE_HOME/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_11.0.1_rsa
   /plugin.xml
```

2. Change:

```
extensionName="jce"
```

to:

```
extensionName="rsa"
```

plugin.xml now looks similar to the one in Listing 2-5.

Listing 2-5 Modified plugin.xml for the new encryption plug-in

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
<extension id="encryption"
   name="default encryption helper"
   point="org.eclipse.birt.report.model.encryptionHelper">
   <encryptionHelper class="com.actuate.birt.model.
      defaultsecurity.api.DefaultEncryptionHelper"
      extensionName="rsa" isDefault="true" />
   </extension>
</plugin>
```

3. Save and close plugin.xml.

4 Modify the original plug-in's descriptor file, so that it is no longer the default encryption plug-in.

1. Open:

   ```
   $ACTUATE_HOME/BRDPro/eclipse/plugins
      /com.actuate.birt.model.defaultsecurity_11.0.1/plugin.xml
   ```

2. Change:

   ```
   isDefault="true"
   ```

   to:

   ```
   isDefault="false"
   ```

3. Save and close plugin.xml.

5. Set the encryption type in the new plug-in to RSA.

   1. Open:

      ```
      $ACTUATE_HOME/BRDPro/eclipse/plugins
         /com.actuate.birt.model.defaultsecurity_11.0.1_rsa
         /encryption.properties
      ```

   2. Change the encryption type to public encryption:

      ```
      type=public encryption
      ```

   3. Change the algorithm type to RSA:

      ```
      algorithm=RSA
      ```

   4. Copy the pre-generated private and public keys for RSA to the symmetric-key and public-key properties. encryption.properties now looks similar to the one in Listing 2-6.

      Listing 2-6 Modified encryption.properties file for the new encryption plug-in

      ```
      #message symmetric encryption , public encryption
         type=public encryption
      #private encryption: DES(default), DESede
      #public encryption:  RSA
         algorithm=RSA
      # NONE , CBC , CFB , ECB( default ) , OFB , PCBC
         mode=ECB
      #NoPadding , OAEP , PKCS5Padding( default ) , SSL3Padding
      padding=PKCS5Padding
      #For key , support default key value for algorithm
      #For DESede ,DES we only need to support private key
      #private key value of DESede algorithm : 20b0020e918..
      #private key value of DES algorithm: 527c23ea...
      #for RSA algorithm , there is key pair. you should support
      #private-public key pair
      ```

      (continues)

```
#private key value of RSA algorithm: 308202760201003....
#public key value of RSA algorithm: 30819f300d0....
#private key
symmetric-key=308202760....
#public key
public-key=30819f300d0.....
```

5 Save and close encryption.properties.

6 To test the new default RSA encryption, open a BIRT Report Designer and create a new report design. Create a data source and type the password.

7 View the XML source of the report design file. Locate the data source definition code. The encryptionID is rsa, as shown in Listing 2-7.

Listing 2-7 Data source definition, showing the encryption ID

```
<data-sources>
   <oda-data-source extensionID="org.eclipse.birt.report.
        data.oda.jdbc" name="Data Source" id="6">
      <text-property name="displayName"></text-property>
      <property name="odaDriverClass">
         com.mysql.jdbc.Driver
      </property>
      <property name="odaURL">
         jdbc:mysql://192.168.218.225:3306/classicmodels
      </property>
      <property name="odaUser">root</property>
      <encrypted-property name="odaPassword" encryptionID="rsa">
         36582dc88.....
      </encrypted-property>
   </oda-data-source>
</data-sources>
```

8 Create a data set and a simple report design. Preview the report to validate that BIRT connects successfully to the database server using the encrypted password. Before trying to connect to the data source the report engine decrypts the password stored in the report design using the default RSA encryption. The engine sends the decrypted value to the database server.

Generating encryption keys

The default encryption plug-in provides classes that can be used to generate different encryption keys. The classes' names are SymmetricKeyGenerator and PublicKeyPairGenerator. SymmetricKeyGenerator generates private keys, which are also known as symmetric keys. PublicKeyPairGenerator generates public keys. Both classes require acdefaultsecurity.jar in the classpath.

Both classes take two parameters, the encryption algorithm and the output file, where the generated encrypted key is written. The encryption algorithm is a

required parameter. The output file is an optional parameter. If you do not provide the second parameter, the output file is named key.properties and is saved in the current folder. The encryption algorithm values are shown in Table 2-6.

Table 2-6 Key generation classes and parameters

Class name	Encryption algorithm parameter
com.actute.birt.model.defaultsecurity.api.keygenerator.SymmetricKeyGenerator	des
com.actute.birt.model.defaultsecurity.api.keygenerator.SymmetricKeyGenerator	desede
com.actute.birt.model.defaultsecurity.api.keygenerator.PublicKeyPairGenerator	rsa

How to generate a symmetric encryption key

Run the main function of SymmetricKeyGenerator.

1. To navigate to the default security folder, open a command prompt window and type:

```
cd C:\Program Files\Actuate11\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_11.0.1
```

2. To generate the key, as shown in Figure 2-2, type:

```
java -cp acdefaultsecurity.jar
   com.actuate.birt.model.defaultsecurity.api.keygenerator.
   SymmetricKeyGenerator des
```

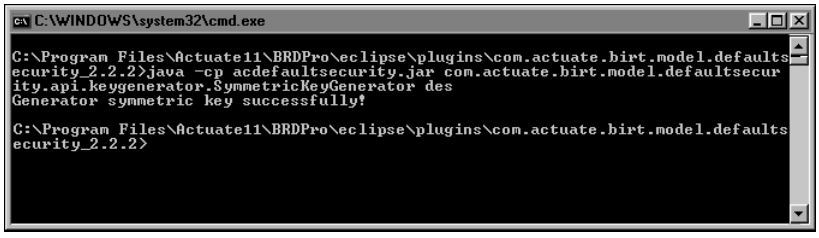

Figure 2-2 Symmetric key generation

3. The key is generated and saved in the file, key.properties. The content of the file looks like the following:

```
#Key Generator
#Wed Nov 18 16:17:06 PST 2008
symmetric-key=73c76d5…
```

4. Copy the key from the generated key file to encryption.properties file.

How to generate a public key with RSA encryption

Run the main function of PublicPairGenerator.

1. To navigate to the default security folder, open a command prompt window and type:

```
cd C:\Program Files\Actuate11\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_11.0.1
```

2. In the command prompt window, type:

```
java -cp acdefaultsecurity.jar
   com.actuate.birt.model.defaultsecurity.api.keygenerator.
   PublicPairGenerator rsa
```

   The class generates a pair of keys saved in the key.properties file such as the following example:

```
#Key Generator
#Wed Nov 18 15:58:31 PST 2008
public-key=30819f300.....
symmetric-key=3082027502010......
```

3. Copy the key from the generated key file to the encryption.properties file.

Deploying custom emitters

Actuate supports using custom emitters to export BIRT reports to custom formats. The custom emitters in BIRT are implemented as plug-ins and packaged as JAR files. To make them available to Actuate Java Components, copy the emitters to <context-root>/WEB-INF/platform/plugins folder. Every time you deploy a custom emitter, you need to restart the product or the product service. This ensures the emitter JAR file is added to the classpath and the product can discover the new rendering format.

The following products support custom emitters:

- Actuate BIRT Studio
- Actuate BIRT Report Designer
- Actuate BIRT Report Designer Professional
- Actuate Java Components:
  - Actuate BIRT Viewer Component
  - Actuate BIRT Interactive Viewer Component
  - Actuate BIRT Studio Component
  - Actuate BIRT Deployment Kit

Rendering in custom formats

After deploying the custom emitter you can see the new rendering formats displayed along with built-in emitters in the following places:

- Preview report in Web Viewer in BIRT Report Designer and BIRT Report Designer Professional.
- Export Content dialog of Actuate BIRT Viewer and Actuate BIRT Interactive Viewer.

The following examples show the deployment and usage of a custom CSV emitter. The emitter allows rendering a report as a comma separated file. The custom format type is MyCSV and the JAR file name is org.eclipse.birt.report.engine.emitter.csv.jar.

How to deploy and use a custom emitter in BIRT Report Designers

The assumption in this example is that the Actuate BIRT designers are installed in C:\Program Files\Actuate11 folder on Windows.

1. Copy org.eclipse.birt.report.engine.emitter.csv.jar to:

   ```
   C:\Program Files\Actuate11\MyClasses\eclipse\plugins
   ```

2. Open a BIRT report in BIRT Report Designer or BIRT Report Designer Professional.
3. Preview the report in Web Viewer.
4. The new MYCSV format appears in the list of formats as shown in Figure 2-3.

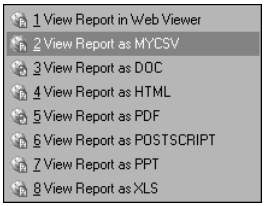

Figure 2-3 List of available formats in Web Viewer

5. Select the MYCSV option. A file download dialog box appears as shown on Figure 2-4. Select Save to save the file. The default file name is iv.mycsv. You have an option to rename the file when saving it. The report content is exported to the new format.

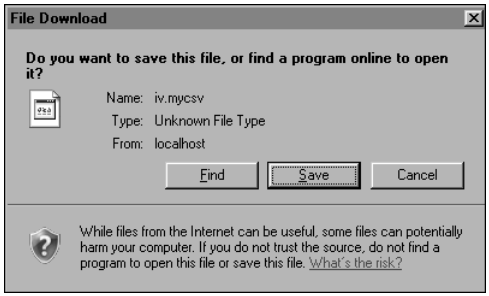

Figure 2-4 Open/Save exported content

How to deploy and use a custom emitter in Actuate Java Components

The assumption in this example is that the Java Components are deployed to Apache Tomcat 6.0, and are installed in C:\Program Files\Apache Software Foundation\Tomcat 6.0 folder on Windows.

1 Copy org.eclipse.birt.report.engine.emitter.csv.jar to:

```
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps
   \ActuateJavaComponent\WEB-INF\platform\plugins
```

2 Restart Apache Tomcat from Start➤Settings➤Control Panel➤Administrative Tools➤Services as shown in Figure 2-5.

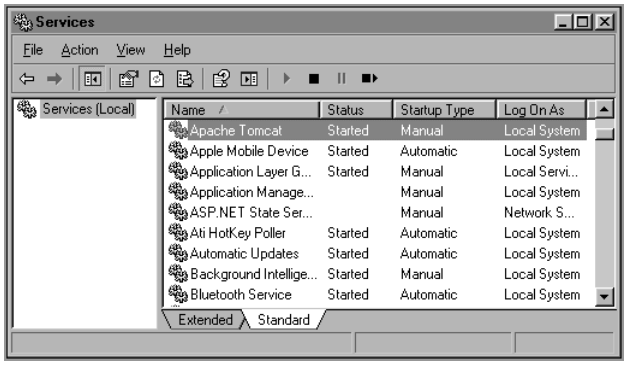

Figure 2-5 Restarting the Apache Tomcat Service

3 Open a BIRT report in Actuate BIRT Viewer or Interactive Viewer.

4 Select Export Content from the viewer menu.

5 The new MyCSV format shows up in the Export Formats, as shown in Figure 2-6.

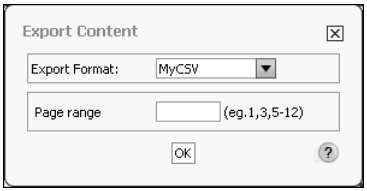

Figure 2-6 Export Content in Actuate BIRT Viewers

6 Choose OK. A file download dialog box appears as shown on Figure 2-4. Select Save to save the file.

Chapter

3

Creating a custom Java Component web application

This chapter contains the following topics:

- Java Component web application structure and contents
- Configuring a custom Java Component web application
- Customizing a Java Component web application
- Modifying global style elements

Java Component web application structure and contents

Java Component generates web pages using a set of default JSPs then sends the web pages to a web browser. Actuate Java Component JSPs use cascading style sheets, JavaScript, and custom tags to generate dynamic web page content. The JavaScript and tags provide access to other JSPs, JavaBeans, and Java classes.

The Java Component web application organizes these interoperating components into a Model-View-Controller (MVC) architecture. To operate a web application, the MVC components perform the following functions:

- Model contains the logic for sending requests to and processing responses from the repository. This component is the data model for Java Component.
- View contains the pages that display data prepared by actions. This component is the presentation portion of Java Component.
- Controller contains the servlets that implement actions. This component is the program control logic for Java Component and manages actions initiated from the browser.

The controller maps actions, designated by URLs with the .do extension, to an actionServlet. The actionServlet is configured with action paths specified in <WAR file root>\WEB-INF\struts-config.xml.

Typically, an action path leads to a JSP with parameters as a web resource. Actuate Java Component file and directory names are case-sensitive. The first time you use a JSP, your web server compiles it into a servlet. Servlets are compiled Java programs or JSPs that run as part of a network service such as a web server. After compiling a JSP into a servlet, a web server can fulfill subsequent requests quickly, provided that the JSP source does not change between requests.

Users make requests to view the contents of a repository, run and view reports, and so on. Each JSP processes any URL parameters by passing them to JSP tags.

You specify the user's file system repository location. To specify the locale and time zone to which to connect, use parameter values in an Actuate Java Component request within a URL or by specifying the desired values in the login form. For example, the following URL specifies the en_US locale for U.S. English, and the Pacific standard time for the timezone parameter:

```
http://localhost:8080/ContextRoot/login.do
   ?locale=en_US&timezone=PST
```

Understanding Java Component directory structure

The Java Server Pages (JSPs) that implement Actuate Java Component URIs are grouped by function into directories under the context root. The context root is the web directory in which an Actuate Java Component web application resides, which is the web archive (.war) file's name. When the web archive (.war) file is extracted, the context root for Java Component is the root directory of the web archive (.war) file. The Java Component context root name in the web or application server's configuration file is the name of the web archive (.war) file as set by the Java jar utility. Figure 3-1 shows the Java Component directory structure.

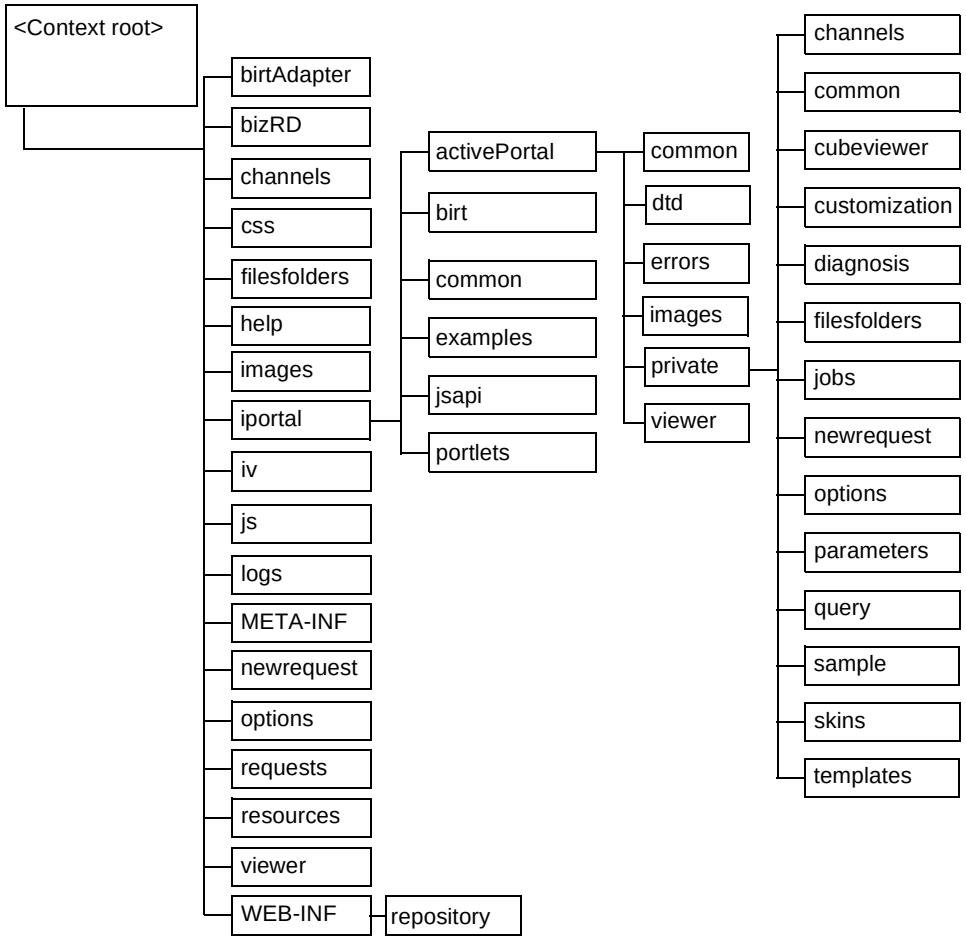

Figure 3-1 Actuate Java Component directory structure

Actuate Java Component URIs convey user requests to an application server.

Pages that support folder navigation and document viewing reside in the <context root>\iportal\activePortal directory. Within this directory, pages that support report viewing reside in the viewer directory, pages that serve as templates for other pages reside in the templates directory, and so on. Some directory names exist directly under the iportal directory and also under the <context root>\iportal\activeportal\private subdirectory. Customize the JSPs under the private subdirectory. Table 3-1 lists and describes the general context root directories.

Table 3-1 <Context root> directories

Directory	Contents
This directory	ajclanding.jsp, the default page for accessing all Actuate Java Component functionality, and supporting material.
birtAdapter	Pages that support BIRT Viewer.
bizRD	Pages that support BIRT Studio.
channels	Support for channels.
css	Actuate Java Component cascading style sheet (.css) files.
downloads	Downloaded files.
filesfolders	Pages that support working with files and folders.
help	Help files.
images	Images for Actuate Java Component web pages, such as buttons, icons, lines, and bullets.
iportal	The Java Component application.
iv	The Interactive Viewer application.
js	JavaScript files that control specific web page elements such as search, toolbar, and table of contents.
META-INF	The Java Component manifest file.
newrequest	Pages that support requests.
options	Options-specific pages, such as channels, notification, and options update pages.
requests	Pages in this directory provide backward compatibility for custom web applications referencing these pages by URL. Use the action paths and the private\jobs directory for new customization projects.
resources	Support for localization and backward compatibility.
viewer	Pages that support report viewing.
WEB-INF	Files that manage session information such as current user login, roles, and volume.

Table 3-2 lists and describes the iportal directories.

Table 3-2 <Context root>/iportal directories

Directory	Contents
activePortal	Pages that support login and authentication and directories for the remaining pages for folder navigation and document usage
birt	Libraries that support BIRT reports, BIRT Studio, and Interactive Viewer and pages that support BIRT reports
common	Common elements included in all reporting web pages, such as banner and side menu elements
examples	Java Servlet examples
jsapi	JavaScript pages to support the JavaScript API demonstration page
portlets	Actuate JSR-168 portlets

Table 3-3 lists and describes the <context root>\iportal\activePortal directories.

Table 3-3 <Context root>/iportal/activePortal directories

Directory	Contents
This directory	Pages that support login and authentication and directories for the remaining folder and document pages for the Java Component application.
common	Common elements included in all reporting web pages, such as banner and side menu elements.
dtd	Document type definitions.
errors	Error pages.
images	Images for reporting web pages, such as buttons, icons, lines, and arrows.
private	Most Java Component folders and documents web pages. Users cannot directly access pages in this directory using URLs. These pages are customizable.
private\ channels	Pages that support channels. Channels have no relevancy in the Deployment Kit.
private\ common	Common elements included in all reporting web pages, such as banner and side menu elements.
private\ cubeviewer	Pages that support viewing Actuate Analytics Option cubes. The cube viewer has no relevancy in the Deployment Kit.

(continues)

Table 3-3 <Context root>/iportal/activePortal directories (continued)

Directory	Contents
private\ customization	Pages that support customization of skins.
private\ diagnosis	Self-diagnostic utility page.
private\ filesfolders	Pages that support working with files and folders.
private\jobs	Pages that support requests such as completed requests, successful submission, and details pages by redirecting.
private\ newrequest	Pages that support new requests, such as parameter processing, scheduling, and job status pages.
private\options	Options-specific pages, such as channels, notification, and options update pages.
private\ parameters	Pages that support table parameters.
private\query	Pages that support Actuate Query functionality. Queries have no relevancy in the Deployment Kit.
private\sample	Example custom requester page.
private\skins	Skins definitions.
private\ templates	Jakarta Struts template pages that simplify customization by handling common web page structure and functionality for many pages.
viewer	Pages that support report viewing. The viewer has no relevancy in the Deployment Kit. The BIRT Viewer is a separate application and is not in the viewer directory.

Actuate recommends that you group Java Component applications in the home directory of an Actuate distribution to make them easier to locate. Place the context root in whatever location your application requires. To ensure that the JSP engine locates your Java Component application's context root, always use the jar utility to generate the web archive (.war) file after licensing or customization.

Building a custom Java Component context root

An Actuate Java Component web application resides in a context root. You specify the Java Component context root by naming the WAR file. For example, if your web archive (.war) file were named ActuateJavaComponent.war and you deployed it on an Apache Tomcat web server, the URL to access the application is:

```
http://<web server>:<port>/ActuateJavaComponent/
```

Apply a similar process to setup other application servers and servlet engines. By configuring the context root, the application server will route requests from the user's browser for Java Component web content to the JSPs in the context root.

You can create several Actuate Java Component context roots. Each context root can contain a web reporting application that uses a different design. For example, you can create different web reporting applications for particular language groups or departments.

How to create a new context root

In the following example, you create a custom reporting web application for MyCorp's Marketing Communications group. You want your Marketing Communications users to use the following URI prefix to access their custom application:

```
http://MyCorp:8900/marcom
```

For example, to access their application's login page they would choose a web page hyperlink with the following URI:

```
http://MyCorp:8900/marcom/login.do
```

1. Extract the contents of the Java Component WAR or EAR file into a temporary directory.
   - On a Windows server, open a command window and type the following commands, replacing the E: DVD drive letter with the path of your Java Component WAR file:

     ```
     cd C:\Temp\jc
     copy E:\ActuateJavaComponent.war
     jar -xf ActuateJavaComponent.war
     ```

     The Java Component files appear in the temporary directory. Leave the command window open.
   - On a LINUX or UNIX server, type the following commands, replacing the DVD drive name with the path of your Java Component WAR file:

     ```
     cd /temp/jc
     cp /dev/dsk/cd/ActuateJavaComponent.war  .
     jar -xf ActuateJavaComponent.war
     ```

     The Actuate Java Component files appear in the temporary directory.
2. Use the jar utility to create a marcom.war file. Type the following command:

   ```
   jar -cf ../marcom.war *
   ```

   This command creates marcomt.war in the parent directory. This new Java Component WAR file now has the context root marcom.
3. Deploy the marcom.war file to the application server or servlet engine on the MyCorp host as an application. Set the service port to 8900.

4 Restart your application server or JSP engine. For example, to restart Apache Tomcat on a Windows XP system, perform the following steps:

1 From the Windows Start menu, choose All Programs➤Administrative Tools➤Services.

2 On Services, select Apache Tomcat service.

3 From the menu, choose Action➤Restart.

4 Close Services.

After you stop and restart the server, your Marketing Communications users can access the Java Component web application called marcom. The application looks like the default Actuate Java Component application because you have not customized its appearance.

Modifying existing content or creating new content

You can modify the content of an existing page or create new pages to link to your custom web application. Typically, a web page has a simple JSP that specifies the template to use and another JSP to use as the content element. For example, the following code specifies that the content element uses the JSP code in <context root>\iportal\activePortal\private\newrequest\newrequestpage.jsp:

```
<template:put name="content" content="/iportal/activePortal
   /private/newrequest/newrequestpage.jsp" />
```

The content JSP contains the code that creates the page-specific content and functionality. This JSP contains code that places page-specific text, graphics, links, and other functionality on the page. You can use HTML code, JSP code, JSP built-in tags, Jakarta Struts tags, Actuate servlets, Actuate custom tags, Actuate JavaBeans, CSS, and JavaScript methods to obtain data and present information on the page. For information about how to use these features, see "Customizing a Java Component web application," later in this chapter.

The default Actuate Java Component pages use HTML tables to provide formatting for each page. The tables are often nested. Individual files include other files that define elements, such as the <TABLE> declaration. As you modify the pages to suit your needs, verify that the Actuate Java Component pages for tasks, such as logging in, listing folders and files, and viewing and requesting reports appear correctly in your web browser.

When using relative hyperlinks in your HTML code, ensure that any files to which you refer are available to Actuate Java Component. Java Component resolves relative hyperlinks from the context root. For example, in the standard Java Component installation, the following code refers to an images directory at the same level as the Java Component context root directory:

```
<A HREF="../images/myimage.gif">
```

All Actuate Java Component requests require action paths to have certain names. Similarly, the action paths require JSP files to have certain names and to reside in a particular directory under the context root. Do not rename the default files provided with Java Component without making the corresponding change to struts-config.xml. If you do not change the file name consistently in all places, Java Component cannot locate your custom files.

Activating a new web application

To activate the changes you make in the Java Component configuration files, content pages, or by creating a new context root, you must restart the web server that runs Java Component.

How to restart a web service on a Windows XP system

1. From the Windows Start menu, choose All Programs➛Administrative Tools➛Services.
2. On Services, select Application Server or servlet container service.
3. From the menu, choose Action➛Restart.
4. Close Services.

Configuring a custom Java Component web application

Java Component's configuration determines many of its essential methods. Configuring your web application customizes how it operates internally, as well as having an effect on the user's experience.

Customize specific pages and operations using the Actuate Java Component web pages, as described in "Customizing a Java Component web application," later in this chapter.

Perform cosmetic customization tasks using the Actuate Java Component style sheets, as described in "Modifying global style elements," later in this chapter.

Customizing Java Component configuration

You set configuration parameters for the Java Component application to tune performance and to control service and application execution.

You configure the Java Component application by changing configuration file contents, such as web.xml. To understand the common configuration files and how each of their entries affect Java Component, see Chapter 4, "Actuate Java Component configuration."

The following section describes the customization procedure using the text editor.

How to customize Java Component configuration parameters

Use the following procedure to customize configuration parameters for Java Component. In this procedure, it assumed that web.xml is the configuration file.

1. Extract the contents of the Actuate Java Component WAR or EAR file into a temporary directory.
2. Make a backup copy of web.xml.
3. Using a text editor that supports UTF-8 encoding, edit web.xml to change parameter values. Parameter definitions use the following format:

   ```
   <param-name><keyword></param-name>
   <param-value><value></param-value>
   ```

   where

   - <keyword> is the name of the parameter.
   - <value> is the parameter value.

   Do not enclose the keyword and value within quotes, and use no spaces between <param-name>, the keyword or value, and </param-name>. For example, the definition for the default locale parameter is:

   ```
   <param-name>DEFAULT_LOCALE</param-name>
   <param-value>en_US</param-value>
   ```

4. Save web.xml.
5. Recompress your Java Components WAR file using the Java jar utility and redeploy it to the application server or servlet engine as an application.
6. Restart the application server or servlet engine that runs Java Component.

How to set a default Java Component locale and time zone

The default locale and timezone for Java Components are set when you install it. To change the default settings, you modify the values of the DEFAULT_LOCALE and DEFAULT_TIMEZONE configuration parameters.

1. Extract the contents of the Actuate Java component WAR or EAR file into a temporary directory.
2. Using a UTF-8 compliant code editor, open the web.xml configuration file.
3. Navigate to the lines that define DEFAULT_LOCALE, similar to the following code:

   ```
   <param-name>DEFAULT_LOCALE</param-name>
   <param-value>en_US</param-value>
   ```

   Change the current locale id, en_US in the above example, to the desired locale id in param-value. Valid locale id strings are listed in <context root>\WEB-INF\localemap.xml.

4 Navigate to the lines that define DEFAULT_TIMEZONE, similar to the following code:

```
<param-name>DEFAULT_TIMEZONE</param-name>
<param-value>America/Los_Angeles</param-value>
```

Change the current time zone id, Pacific Standard Time in the above example, to the desired default time-zone in param-value. Valid time zone id strings are listed in <context root>\WEB-INF\TimeZones.xml.

5 Save web.xml.

6 Recompress your Actuate Java Component WAR or EARfile using the Java jar utility and redeploy it to the application server or servlet engine as an application.

7 Restart the application server or servlet engine that runs Java Component.

Customizing requester pages

When a user chooses to run a report, a requester page appears. Using the requester page, a user chooses values for the report's parameters, if there are any. The user can also select execution options, such as the desired output format. You can create or modify requester pages for your custom web application. The following list provides a summary of the techniques for customizing requester pages:

- Create a new JSP form for the user to specify the desired report parameter values and then use these values to construct the appropriate Actuate Java Component URI to execute the report.

  Actuate recommends this approach. It supports full control of the requester page design while using existing functionality for the execution of the report.

- Modify the existing requester page files.

  This approach is best for minor changes, such as hiding one of the many execution options that the page supports.

- Create a new requester page and an Action class to provide processing of the page and execution of the report.

  This approach provides full control of the requester page design and the processing of the report. Creating an Action class requires an understanding of Java and Jakarta Struts.

Customizing a Java Component web application

Actuate Java Component supports customization of the landing page, <context root>\landing.jsp, and the appearance of the pages in My Documents, BIRT Studio, and the Interactive Viewer for BIRT reports and business reports.

You use knowledge of the following standard languages and frameworks to customize a Java Component web application manually:

- Cascading style sheet (.css) files
  CSS files define fonts, colors, and other visual design attributes of a Java Component web application. For information about modifying style sheets, see "Modifying global style elements," later in this chapter.
- Hypertext markup language (HTML)
  HTML handles links and the presentation of text and graphics in web pages. Java Component incorporates HTML code in its JavaServer pages.
- Jakarta Struts Framework
  Jakarta Struts Framework is an open source framework for building web applications. Based on standard technologies, Struts enables the Java Component Model-View-Controller design. For more information about Struts, access the following URL:

  `http://jakarta.apache.org/struts`
- Java
  Java Component uses Java classes to provide functionality. You can create your own Java classes for your custom web application. For more information on the Java Component Java classes, see Chapter 8, "Actuate Java Component JavaBeans."
- JavaScript
  JavaScript is an interpreted object-oriented language that facilitates embedding executable content in web pages. It provides strong tools for interacting with web browsers.
- JavaServer Pages
  The JavaServer Pages (JSP) extension of the Java Servlet API facilitates the separation of page design from business logic. JSPs are a platform-independent solution. Java Component web pages are defined primarily by JSPs. For more information about the Actuate JavaServer Pages, see Chapter 5, "Actuate Java Component URIs."

Actuate recommends that you use the skin manager to customize as much as possible and then handle any remaining customization tasks manually.

Viewing modifications to a custom web application

After making changes to your Java Component web application, you need to view the changes. Caching in the browser or your application server can interfere with seeing the changes you have made. After changing a Java Component application, complete these general tasks in order:

- Save any files involved in the change.

- Refresh the browser page.
- If you do not see changes you made in a JSP or XML file, complete the following tasks in order:
  - Shut down the JSP engine.
  - Clear the JSP engine's cache or work directory to ensure that the JSP engine picks up your changes.
  - Restart the JSP engine.
- If you do not see changes you made in a cascading style sheet file or a JavaScript file, clear the web browser's cache, then refresh the page.

Your changes appear in the web browser.

Locating existing pages and linking in new pages

Actuate Java Component controls web page navigation with Jakarta Struts action paths. An action path is a uniform resource identifier (URI) called directly by Java Component or by a user to access the Java Component functionality. <context root>\WEB-INF\struts-config.xml contains the action path specifications.

An action path can specify a JSP to use to gather input. The action path uses the results of an Action class to determine the next action path to perform or the next JSP to display. Typically, an action path forwards the user to one action path or JSP if the execution succeeds and a different action path or JSP if the execution causes an error. In the following code sample, if the AcGetFolderItemsAction JavaBean returns success, the next JSP to display is <context root>\iportal \activePortal\private\filesfolders\filefolderlist.jsp:

```
<!-- Process getfolderitems  -->
<action
   attribute="fileListActionForm"
   name="fileListActionForm"
   path="/getfolderitems"
   scope="request"
   type="com.actuate.activeportal.actions.AcGetFolderItemsAction"
   validate="false">
   <forward name="success"
      path="/iportal/activePortal/private/filesfolders
      /filefolderlist.jsp" />
</action>
```

In the preceding example, the path for an error result uses the definition in the global forwards section of struts-config.xml as a default value:

```
<forward name="error"
   path="/iportal/activePortal/private/common/errors
   /errorpage.jsp"/>
```

If the JavaBean returns another result, such as viewroi, you can include a forward for that result, as shown in the following example:

```
<forward name="viewroi"
   path="/iportal/activePortal/viewer/viewframeset.jsp"
   redirect="true" />
```

To locate an existing page, navigate to that page and examine the URI in the address field of your browser. If the URI contains a JSP name, go to that file. If the URI contains an action path, search struts-config.xml for that action path without the .do extension, or look up the action path in Chapter 5, "Actuate Java Component URIs."

To add a new web page to Java Component, you change the navigation in struts-config.xml so that all navigation for your web application remains in a single location. You can change an existing input page or forward page specification in an action path to your new page, or you can create a new action path that forwards to your page. If you create a new action path, you can change another action path to forward to your new path or you can modify or create links on web pages to specify your new action path. The following action path always navigates to welcome.jsp when another action path, link, or URL invokes it:

```
<!-- Process welcome  -->
<action path="/welcome"
   forward="/iportal/activePortal/private/welcome.jsp"
   name="welcome">
</action>\
```

For more information on action paths and Jakarta Struts, access the following URL:

```
http://jakarta.apache.org/struts
```

Obtaining information about the user and the session

Typically, new Actuate Java Component web pages need access to session information. Your application server and Java Component store information about the session that you can use in your web pages. You can obtain the serverURL, volume, and other information from your application server, as shown in the following example. The volume parameter returns the name of the machine that hosts the application server and the serverURL parameter returns an empty string.

```
String volume = request.getParameter("volume");
String serverURL = request.getParameter("serverurl");
String userId = request.getParameter("userid");
String password = request.getParameter("password");
String roxReport = request.getParameter("report");
```

You also can obtain the context root path from your application server, as shown in the following code:

```
String contextRoot = request.getContextPath();
```

Actuate Java Component stores a wide variety of information about the session in UserInfoBean. To access UserInfoBean, place the following line of code near the top of your JSP:

```
<jsp:useBean id="UserInfoBean"
   class="com.actuate.activeportal.beans.UserInfoBean"
   scope="session"/>
```

After this line, you can access information in the JavaBean by the appropriate get method. The most important method for new pages is the getIportalid() method. This method retrieves the user's authentication ID with the server. This ID is based on the user name only.

To write generic code, you need to determine whether your application is running. Java Component includes a utility class, iPortalRepository, that provides this information. To access this class in your JSP, place the following code at the head of your JSP:

```
<%@ page
   import="com.actuate.iportal.session.iPortalRepository"
%>
```

You can then use code similar to the following line to check the repository type:

```
boolean isEnterprise =
   iPortalRepository.REPOSITORY_ENCYCLOPEDIA.equalsIgnoreCase(
   UserInfoBean.getRepositoryType());
```

You can then use the authentication ID and the repository type to access the server with JSP custom Actuate tags and calls to Java Component beans, as shown in the following examples:

```
String authenticationID = UserInfoBean.getIportalid();
String folderPath = UserInfoBean.getCurrentfolder();
jobDetailURL += StaticFuncs.encode(UserInfoBean.getUserid());
com.actuate.reportcast.utils.AcLocale acLocale =
   UserInfoBean.getAcLocale();
TimeZone timeZone = UserInfoBean.getTimezone();
boolean isEnterprise =
   iPortalRepository.REPOSITORY_ENCYCLOPEDIA.equalsIgnoreCase(
   UserInfoBean.getRepositoryType());
String serverURL =
   ( isEnterprise | UserInfoBean.getServerurl() | "" );
String userVolume =
   ( isEnterprise | UserInfoBean.getVolume() | "" );
```

Customizing accessible files and page structure using templates

Actuate Java Component uses Jakarta Struts templates to simplify JSP code and customization. Java Component templates handle overall page organization, access to Jakarta Struts custom tag libraries, and access to common CSS and JavaScript files. The login page and landing page do not use a template. Table 3-4 describes the Java Component templates.

Table 3-4 Actuate Java Component Struts templates

Template	Method
simpletemplate.jsp	Used for errors, confirmations, and other simple pages
querytemplate.jsp	Used by most Actuate Query pages
template.jsp	Used by all other pages except the login page

Each Actuate Java Component skin has its own version of these templates in <context root>\iportal\activePortal\private\skins\<skin name>\templates. The set of templates in <context root>\iportal\activePortal\templates sets up several JavaBeans and then accesses the template of the same name for the user's selected skin. Typically, customization only involves templates in <context root>\iportal\activePortal\private\skins\<skin name>\templates.

Specifying a template and template elements

To use a template and template elements, a page uses the Jakarta Struts custom template tags, described in Table 3-5.

Table 3-5 Struts template tags

Template tag	Method
template:insert	Specifies the template to use
template:put	Specifies the text or file to use for a template element such as the name, banner, side menu, or content elements

The Actuate Java Component template element that is displayed depends on the skin. A skin is assigned by changing the DEFAULT_WORKGROUP_SKIN parameter in web.xml. Table 3-6 lists the DEFAULT_WORKGROUP_SKIN values that correspond to the templates for each of the default skins.

Table 3-6 Valid DEFAULT_WORKGROUP_SKIN values

Skin	Value
Classic	classic
Tabbed	tabbed
Tree View	treeview

The custom template tags define the JSPs to use for the template and the custom elements that the template specifies to build the user interface. For example, the template:insert tag in the following code specifies the use of template.jsp. The first template:put tag accesses the localized string for the title of the page. The remaining template:put tags specify that the template use banner, side menu, and content elements using the files specified in each tag.

Table 3-7 lists the Java Component templates and the pages that use them.

Table 3-7 Templates for JSPs

Template	JSPs in iportal\activePortal\private
querytemplate.jsp	query\create.jsp query\execute.jsp
simpletemplate.jsp	common\errors\errorpage.jsp customization\fileupload.jsp newrequest\newrequest2.jsp query\confirmation.jsp query\fileexists.jsp query\runconfirmation.jsp
template.jsp	customization\skinedit.jsp customization\skinmanager.jsp filesfolders\deletefilestatus.jsp filesfolders\createfolder.jsp filesfolders\filedetail.jsp filesfolders\filefolderlist.jsp filesfolders\search\filefolderlist.jsp newrequest\newrequest.jsp newrequest\submitjobstatus.jsp options\options.jsp

Changing a template

Make changes to all pages that use a particular template by changing only the template. Add or remove lines in the template that make cascading style sheets, JavaScript files, and other resources accessible to all pages that use the template. Customize the overall structure of all pages that use a template by moving,

resizing, or removing the HTML, JSP, and Jakarta Struts code describing the layout of the web pages that use the template.

For example, the innerTable of <context root>\iportal\activePortal\private \skins\treeview\templates\template.jsp specifies various HTML commands and embedded Jakarta Struts tags that populate the content frame. The inner banner with the breadcrumb is in the top row. The second row contains the content page.

```
<table class="innerTable" border="0" cellspacing="0"
   cellpadding="0">
<% if (!"false".equalsIgnoreCase(showBreadCrumb)) { %>
<tr>
   <td class="allBreadcrumbs">
   <jsp:include page="<%= breadcrumb %>" flush="true" >
   <jsp:param name="fromDashboard" value="<%= fromDashboard %>" />
   <jsp:param name="showBanner"  value="<%= showBanner %>" />
   <jsp:param name="showSideBar"  value="<%= showSideBar %>" />
   <jsp:param name="showBreadCrumb" value="<%= showBreadCrumb %>"
   />
   </jsp:include>
   </td>
</tr>
<% } %>
<tr>
   <td class="fileFolderListContent" valign="top">
   <div>
   <template:get name="content" flush="true"/>
   </div>
   </td>
</tr>
</table>
```

The breadcrumb, or navigation trail, is a link or set of links. On a document page, the breadcrumb displays the repository and any folders and pages you access. Use any of these items as a link to return to that level. For a jobs or channels page, the breadcrumb supports direct access to a document page.

To implement the expandable tree, a frameset in <context root>\iportal \activePortal\private\skins\treeview\templates\template.jsp specifies the sidebar and content frames using HTML and embedded Jakarta Struts tags that define the content.

```
<FRAMESET cols="20%,80%" border="1"
   onload="if (typeof(bodyOnload) != 'undefined') bodyOnload();">
   <FRAME src="<html:rewrite page="<%= sidebar %>"/>"
      name="<%=htmlSideFrameName%>"
      id="<%=htmlSideFrameId%>"
```

```
      scrolling="auto"
   />
   <FRAME src="<html:rewrite href="<%= contentURL %>" />"
      name="main"
      scrolling="auto"
   />
</FRAMESET>
```

Modifying global style elements

Although JSPs can use HTML to set colors, fonts, and other stylistic elements directly, the JSPs also use cascading style sheets (CSS), templates, and shared images to control the global styles of a Java Component web application. To modify the appearance of the entire Java Component web application, change global style elements.

Global styles can change more than the appearance of Actuate Java Component. For example, to view search results with HKSCS characters in an English locale, change the .searchresultlink style's font from Arial to MingLiU_HKSCS. This style change only affects the search results.

Understanding style definition files

Additional style definitions for each provided skin come from <context root>\iportal\activePortal\private\skins\<skin name>\css\skinstyles.css. Add more styles to this file if you want the style definitions to take effect for only a particular skin. Java Component's JSP typically link these styles in the following order:

- <context root>\css\allstyles.css

```
<LINK href="<html:rewrite page="/css/allstyles.css"/>"
   type="text/css" rel="stylesheet">
```

- <context root>\iportal\activePortal\private\skins\<skin name>\css\skinstyles.css

```
<LINK
   href="<ap:skinResource resource="/css/skinstyles.css" />"
   type="text/css" rel="stylesheet" >
```

- Style specifications from the customization web pages

```
<STYLE>
   <bean:write
      name="UserInfoBean" property="skinConfig.cssCode" />
</STYLE>
```

If a style is defined in more than one of these files, the JSP engine uses the definition in the last file that contains the style. Thus the settings you specify in the customization web pages override any other CSS files.

allstyles.css contains additional style definitions for the Actuate Java Component application. Modify allstyles.css to change any style definitions that are not handled within the customization web pages or the <context root>\iportal \activePortal\private\skins\<skinname>\css\skinstyles.css file. Changes to a style in allstyles.css affects all Java Component skins except the parameters page unless the customization web pages or a skin's skinstyles.css file override it. To customize the parameter component, modify the style definitions in the <context root>\css\parameter.css file.

How to test and modify styles depending on the browser type

1. Near the top of your JSP, link in the allstyles.css style sheet.

```
<LINK href="<html:rewrite page="/css/allstyles.css"/>"
   type="text/css" rel="stylesheet" >
```

2. After this line, link in the style sheet located in the current skin's css directory.

```
<LINK
   href="<ap:skinResource resource="/css/skinstyles.css" />"
   type="text/css" rel="stylesheet" >
```

3. Use the Jakarta Struts bean:write custom tag to generate and include the style definitions for all styles defined through skin customization pages in Actuate Java Component.

```
<STYLE>
   <bean:write
      name="UserInfoBean" property="skinConfig.cssCode" />
</STYLE>
```

4. If the skin customization styles contain any settings that do not work in a specific browser, you can override them individually.

Specifying colors and fonts

Specify fonts and colors for styles in the customization web pages or in the cascading style sheets. Specify the color in any of the following ways:

- Using a color name such as navy, yellow, or teal, as shown in the following example:

```
color: Yellow;
```

- Using hexadecimal notation to set the amount of red, green, and blue to use in the color.

```
#FFFF00
```

- Using decimal notation to set the amount of red, green, and blue to use in the color. In the customization web pages, fill in the value for red, green, and blue in the corresponding fields. In a CSS file, use a call to the rgb() method, as shown in the following example:

  ```
  color: rgb(156, 207, 255);
  ```

How to change the font style of a single item

To change Actuate Java Component pages to display the user, system name, and volume in 12-point italic Comic Sans MS font:

1. Extract the contents of the Actuate Java Component WAR or EAR file into a temporary directory.
2. In a text editor, open <context root>\css\allstyles.css.
3. Locate the following string:

   ```
   bannerTextArea
   ```

   There are two instances of the string bannerTextArea. The first is part of the definition for all the banner styles. This definition sets the banner styles' common attributes. The second instance sets the attributes for bannerTextArea only and looks like the following text:

   ```
   .bannerTextArea {
      color: white;
      font-size: 10pt;
      text-align: left;
      white-space: nowrap;
   }
   ```

4. Modify the code that follows the bannerTextArea definition to change the font as shown in the following code:

   ```
   .bannerTextArea {
      color: white;
      font-family: Comic Sans MS;
      font-size: 11pt;
      font-style: italic;
      text-align: left;
      white-space: nowrap;
   }
   ```

5. Save and close the CSS file.
6. Recompress your Actuate Java Component WAR or EAR file using the Java jar utility and redeploy it to the application server or servlet engine as an application.
7. Restart the application server or servlet engine that runs Java Component.

8 Refresh your web browser to view the changes. Figure 3-2 shows the new appearance of the banner.

Figure 3-2 Appearance of customized Java Component banner

Customizing page styles for BIRT Studio

To customize BIRT Studio pages, use the files in <context root>\bizRD\styles. This directory includes the following customizable CSS files:

- accordion.css defines styles for the report design area of the page, which displays the Available Data, Report Template Items, and other selectable tree views.
- dialog.css defines styles for dialog boxes that have shared characteristics, including the dialog boxes for template selection, file browsing, calculations, parameters, and so on.
- dialogbase.css defines the style of dialog containers, such as the button style, the Close icon style, and so on.
- title.css defines styles for the title bar of BusinessReport Studio pages.
- toolbar.css defines styles for the toolbar.
- wrcontextmenu.css defines the styles for BusinessReport Studio context menus.

Another file in this directory, webreporting.css, is not customizable.

For more information about using cascading style sheets, access the following URL:

`http://www.w3.org/Style/CSS/`

Modifying images

To use your own graphics, replace the default Java Component images. Java Component pages use images for the company logo in the banners, on the side menu, and for the background. Some pages use additional images that are related to their content. You can also add new images to pages.

Certain images are most easily changed by customizing a skin. You can customize the company logo and the My Folder icon for all skins. In addition, you can customize the open and closed folder icons and volume icon for a skin that is cloned from the Tree View skin. These and all other images that you can customize reside in <context root>\iportal\activePortal\private\skins \<skin name>\images. Update these images by using the skin customization pages to use new graphic files instead of changing the supplied graphic files.

Customizing the images described in Table 3-8 affects most Java Component web pages.

Table 3-8 Images in Java Component skins

Skins	Default image file	Description
All	logo.gif	Specifies the company logo to use in the banners
All	homefoldericon.gif	Specifies the image to use beside the My Folder link
Treeview	closedfoldericon.gif	Specifies the image to use to indicate a unexpanded folder in the hierarchical view of the volume and folders
Treeview	foldericon.gif	Specifies the image to use to indicate an expanded folder in the hierarchical view of the volume and folders
Treeview	volume_icon.gif	Specifies the image to use to indicate a volume in the hierarchical view of the volume and folders

An additional image of interest is <context root>\iportal\activePortal\private \skins\<skin name>\images\background.gif. This image is used by the classic skin and its clones to provide the background for every page. This image is one pixel high and 1280 pixels long, and is copied as necessary to fill the page. You change the contents of this image file directly to modify the background of a classic skin clone.

All other images reside in <context root>\iportal\activePortal\images. This set of images provides the features on the side menu in the classic skin and the tree in the Tree View. Update these feature images by changing the corresponding feature definition in the <context root>\iportal\WEB-INF\functionality-level.config file.

Other images are referenced by hard-coded path and file names in JSP and JavaScript files, such as the icons in <context root>\iportal\activePortal\private \filesfolders\views\categories.jsp. For example, categories.jsp specifies the location and file name, <context root>\iportal\activePortal\images \detailicon.gif, a magnifying glass icon that is used to obtain more details about a document or other item in a list. When you change the location or replace an image with a new file, you must update the JavaScript and JSP files that use them. Alternatively, make a backup copy of the original image and then reuse the original name for your new image. By reusing the original name, you do not need to make any changes in the JSP and JavaScript files using the image.

How to replace the detail icon with your own icon

Actuate Java Component uses a magnifying glass icon to display more information about files, channels, and jobs. For example, <context root>\iportal \activePortal\private\jobs\completedjob.jsp contains the following code using this image:

```
<img src="<html:rewrite
   page="/iportal/activePortal/images/detailicon.gif"/>"
   border="0" align="middle"
   alt="<bean:message bundle="iportalResources"
   key="TTIP_JOB_DETAIL"/>"
   title="<bean:message bundle="iportalResources"
   key="TTIP_JOB_DETAIL"/>" >
```

1. Extract the contents of the Java Component WAR file into a temporary directory.
2. Create your new details image in <context root>\iportal\activePortal \images. The default Actuate Java Component icon, detailicon.gif, is 12 pixels by 13 pixels. During development, use a new name, such as new_detailicon.gif.
3. Rename the existing details image, <context root>\iportal\activePortal \images\detailicon.gif, to another file name, such as detailicon_original.gif.
4. Rename your new details image to detailicon.gif.
5. Recompress your WAR file using the Java jar utility and redeploy it to the application server or servlet engine as an application.
6. Close your browser, re-open Java Component, and log in. The new detail icon appears in all places that Actuate Java Component had displayed the magnifying glass icon. In Figure 3-3, the default detailicon.gif image has been replaced by an image of a multicolored question mark.

Figure 3-3 Customized skin with modified detail icon

If you want to replace only some instances of detailicon.gif, search the files in the context root for all files that use that image. Then replace that file name with your image's file name in only some of the files. For example, you could use the default

magnifying glass in most places but change <context root>\iportal\activePortal \private\common\breadcrumb.jsp to use your own image.

Follow similar procedures to customize other images in Actuate Java Component pages that are not specified in the skin manager or in <context root>\WEB-INF \functionality-level.config.

Part Two

Actuate Java Component Reference

Chapter

4

Actuate Java Component configuration

This chapter contains the following topics:

- About Actuate Java Component configuration
- Configuring Java Component web applications
- Configuring the Actuate Java Component repository
- Configuring the BIRT Viewer and Interactive Viewer
- Configuring BIRT Studio
- Configuring BIRT Data Analyzer

About Actuate Java Component configuration

The Java Component applications are configured using files in the context root's \WEB-INF directory. For example, the web.xml configuration file for your context root is located in the following directory:

```
<context root>\WEB-INF\web.xml
```

Table 4-1 lists the configuration files discussed in this chapter.

Table 4-1 Actuate Java Component configuration files

File	Features	Description
erni_config.xml	BIRT Studio	Configures BIRT Studio functionality
functionality-level.config	Information Console	Configures the Deployment Kit user interface using functionality roles
iv_config.xml	BIRT Viewer	Configures BIRT Viewer and Interactive Viewer user interface
localemap.xml	All	Configures languages and locales
TimeZones.xml	All	Configures time zones
web.xml	All	Configures features of the Deployment Kit, including security, networking, caching, labeling and storage

Configuring Java Component web applications

Java Components provide the ability to organize, run, and view reports. You configure the user interface, logging, and caching for a Java Component using web.xml.

Configuring the Java Component using web.xml

Web.xml contains parameters that control Deployment Kit features. Table 4-2 describes the configuration parameters for the Information Console application.

Table 4-2 Actuate Java Component web.xml parameters

Parameter name	Description
AUTOSUGGEST_DELAY	Configure the delay before the parameters page opens an automatic suggestion tooltip for a parameter. The value is measure in milliseconds, and the default value is 500.

Table 4-2 Actuate Java Component web.xml parameters (continued)

Parameter name	Description
AUTOSUGGEST_LIST_SIZE	Specifies the number of autosuggest entries to display. By default, display everything.
CACHE_CONTROL	Specifies how a web browser caches information using one of the following values: ■ NO-CACHE indicates that the browser does not cache information and forwards all requests to the server. With NO-CACHE, the back and forward buttons in a browser do not always produce expected results, because choosing these buttons always reloads the page from the server. If multiple users access Java Component from the same machine, they can view the same cached data. Setting CACHE_CONTROL to NO-CACHE prevents different users viewing data cached by the browser. ■ NO-STORE indicates that information is cached but not archived. Reports in Excel format do not render reliably when using this setting. ■ PRIVATE indicates that the information is for a single user and that only a private cache can cache this information. A proxy server does not cache a page with this setting. ■ PUBLIC indicates that information may be cached, even if it would normally be non-cacheable or cacheable only within an unshared cache. ■ Unset (no value) is the default value. The browser uses its own default setting when there is no CACHE_CONTROL value. Caching information reduces the number of server requests that the browser must make and the frequency of expired page messages. Caching increases security risks because of the availability of information in the cache. For additional information about cache control, see the HTTP/1.1 specifications.
COOKIE_DOMAIN	Specifies the host name of the server setting the cookie. The cookie is only sent to hosts in the specified domain of that host. The value must be the same domain the client accesses. Actuate Java Component automatically sets this parameter. For example, if the client accesses http://www.actuate.com /iportal/login.do, the domain name is actuate.com.

(continues)

Table 4-2 Actuate Java Component web.xml parameters (continued)

Parameter name	Description
COOKIE_ENABLED	Indicates whether to use cookies to store information between user logins. The default value is true. If false, Java Component does not use cookies. Without cookies, many Java Component features are unavailable or do not persist across sessions. For example, without cookies, user name, language, and time zone settings always use their default values when a new browser session begins.
COOKIE_SECURE	Indicates whether to access and write cookies securely. If true, cookies are only written if a secure connection, such as HTTPS, is established. The default value is false, which enables cookies for all connection types.
DEFAULT_LOCALE	Specifies the default locale. Actuate Java Component sets this parameter value during installation. The locale map is <context root>\WEB-INF\localemap.xml.
DEFAULT_COLUMN_ PAGE_BREAK_INTERVAL	Specifies the number of columns to display on one page when viewing a cross tab. Must be a non-negative number. Default value is 10.
DEFAULT_PAGE_BREAK_ INTERVAL	Specifies the number of rows to display in one page when viewing a report. If set to 0, there are no page breaks.
DEFAULT_ROW_PAGE_ BREAK_INTERVAL	Specifies the number of rows to display on one page when viewing a cross tab. Must be a non-negative number. Default value is 40.
DEFAULT_TIMEZONE	Specifies the default time zone. Actuate Java Component sets this parameter value during installation. The time zone map is <context root>\WEB-INF\TimeZones.xml.
ENABLE_CLIENT_SIDE_ REDIRECT	Specifies whether URL redirection is done on the client side or the server side. Set the value to true for client side redirection. The default value is false. For more information about URL redirection, see "Using proxy servers with Actuate Java Component" in Chapter 1, "Introducing Actuate Java Components."
ENABLE_DEBUG_ LOGGING	Indicates whether to record debugging messages in a log file called Debug.log. Set the value to true to enable debug messages in the log file. The default value is false.
ENABLE_ERROR_ LOGGING	Indicates whether to log errors. This parameter's default value is true, which enables error logging. If you set this parameter to true, Java Component creates two error log files: ■ Admin.log records general errors. ■ Soapfault.log records communication errors.

Table 4-2 Actuate Java Component web.xml parameters (continued)

Parameter name	Description
ENABLE_JUL_LOG	Indicates whether to log Actuate Java Component activity. This parameter's default value is true, which enables logging. If you set this parameter to true, Java Component creates log files named reportService.<Service number>.<System name>.<Java Component start up time stamp>.<File number>.log.
ERROR_LOG_FILE_ ROLLOVER	Specifies the time period to wait before starting a new log file. Options are Daily, Monthly, Weekly, and Yearly. The default value is Monthly.
EXECUTE_REPORT_ WAIT_TIME	Specifies the time to wait, in seconds, for a report to execute. This parameter's default value is 20 seconds. For more information about the wait time parameter, see "execute report page" in Chapter 5, "Actuate Java Component URIs."
FILES_DEFAULT_VIEW	Specifies the default view for the files and folders list using one of the following values: ■ Categories, the default, displays files organized in rows by type. ■ Detail displays files organized in rows by name. ■ List displays files organized in columns with small icons. ■ Icon displays files organized in columns with large icons.
FORCED_GC_INTERVAL	Indicates the length in seconds of the interval that the Java Component application waits between forced garbage collections. To disable garbage collection, set this parameter to 0, the default value. If you use this parameter, 600 seconds is the recommended value. Use this parameter when tuning application server performance. If the value is too low, the application server performs garbage collection too frequently, slowing your system. If you set the value to high, you waste memory. If disabled, the application server controls garbage collection.
INSTALL_MODE	Indicates whether Java Component is installed with iServer. The value is set when Actuate Java Component is installed. Do not change this setting.
JUL_LOG_CONSOLE_ LEVEL	The level of Actuate Java Component activity to log to the console. Valid values are OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, in order of the number of messages to log. The default value is OFF.
JUL_LOG_FILE_COUNT	Specifies the number of log files for a particular time stamp, if the value of ENABLE_JUL_LOG is true.

(continues)

Table 4-2 Actuate Java Component web.xml parameters (continued)

Parameter name	Description
JUL_LOG_FILE_LEVEL	The level of Actuate Java Component activity to log in a file. Valid values are OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, in order of the number of messages to log. The default value is WARNING.
JUL_LOG_FILE_SIZE_KB	The maximum size, in kilobytes, for an Actuate Java Component activity log file. When a log file reaches this size, Java Component creates a new log file and increments its file number. If the log file number reaches the value of JUL_LOG_FILE_COUNT, Java Component resets the file number to zero and overwrites the first log file for the time stamp.
LOG_FILE_LOCATION	Indicates which directory contains the log files. If the value is not an absolute directory path name, Actuate Java Component locates the directory in the Java Component home directory. The default value is logs in the Java Component home directory.
LOGIN_TIMEOUT	Specifies the number of seconds to wait before a session times out. The minimum login timeout is 300 seconds. The maximum value is equivalent to java.lang.Long. Its default value is 1200 seconds.
MAX_BACKUP_ERROR_LOGS	Specifies the maximum number of backup error log files to keep. The default value is 10.
MAX_LIST_SIZE	Limits the number of items returned when getting folder items, jobs, job notices, scheduled jobs, and channels to reduce network traffic. The default value is 150.
PRELOAD_ENGINE_LIST	List of engines that will be loaded when application starts up. Allowed values are "birt" and "ess". Use a comma to separate the names if there are more than one. Engines that are not in the list will be loaded upon request. The default value is birt.
PROGRESSIVE_REFRESH	Controls the interval, in seconds, at which an Actuate report refreshes itself when running a progressive report. The report refresh time starts after the navigation bar loads. The report refreshes first after 15 seconds, then after 60 seconds, and then after the PROGRESSIVE_REFRESH interval. If the value is less than 60, Actuate Java Component uses 60 seconds. This parameter's default value is 1800 seconds.
PROGRESSIVE_VIEWING_ENABLED	Specifies whether a paginated report starts to display in the browser as soon as the first page has been generated. Valid values are true and false. The default value is true.

Table 4-2 Actuate Java Component web.xml parameters (continued)

Parameter name	Description
PROXY_BASEURL	Indicates a proxy server's URL if the network uses one between Java Components and the client. The default value is blank, which indicates that the network does not use a proxy server.
SECURITY_ADAPTER_ CLASS	Specifies the fully qualified class of the security manager that controls access to Actuate Java Component functionality for single sign-on. The default value is no name.
SESSION_DEFAULT_ PARAMETER_VALUE_ID	Specifies the name of the object that stores the HTTP session-level report parameters. This object is an instance of the com.actuate.parameter.SessionLevelParameter class, which is extensible. The default value is SessionDefaultParameterValue.
sessionTimeout	The number of milliseconds the web service Ajax Proxy maintains an idle session. The default value is 5000.
TRANSIENT_STORE_ MAX_SIZE_KB	Limits the amount of disk space that Actuate Java Component uses for temporary files. The default value is 102400, which is 100 MB.
TRANSIENT_STORE_ PATH	Path to Actuate Java Component transient files. The default value is set when Java Component is installed. When deploying more than one context root, set a unique path for each.
TRANSIENT_STORE_ TIMEOUT_MIN	Specifies, in minutes, how long to retain Actuate Java Component transient files. The default value is 40, which is 40 minutes.
VIEW_XLS_IN_ REQUESTER	Indicates that a spreadsheet report in Excel format always opens in the same browser as Java Component. The default value is false, indicating that excel documents open in a separate window.

Configuring Java Component functionality levels with functionality-level.config

A functionality level defines which Java Component user interface features are visible. For example, by default every functionality level shows About and Help links on the Java Component banner. The Intermediate, Advanced, and Administrator levels add a Search link to the banner, as shown in Figure 4-1.

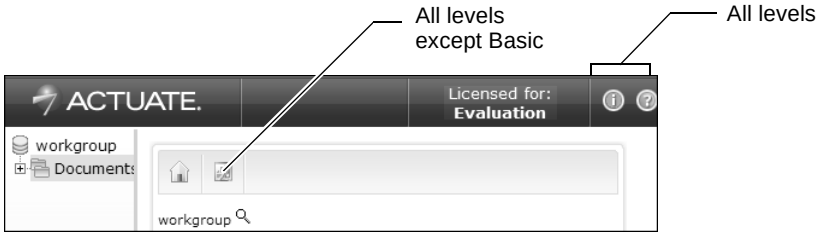

Figure 4-1 The banner as it appears for a user at the Administrator functionality level

Actuate Java Component provides four functionality levels by default. The default level is Intermediate. To change the functionality level, change the value of the DEFAULT_WORKGROUP_FUNCTIONALITY_ROLE parameter in the web.xml configuration file. See "Configuring the Actuate Java Component repository," later in this chapter, for more information. You customize a functionality level by creating or modifying entries in the following file:

```
<context root>\WEB-INF\functionality-level.config
```

You can modify the built-in levels but you cannot delete them.

The following example shows the definition of the Basic functionality level:

```
<Level>
   <Name>Basic</Name>
   <Role>All</Role>
   <FeatureID>Jobs</FeatureID>
   <FeatureID>Documents</FeatureID>
   <FeatureID>Channels</FeatureID>
   <SubfeatureID>DeleteFile</SubfeatureID>
   <SubfeatureID>InteractiveViewing</SubfeatureID>
   <AnalyticsExperienceLevel>Novice</AnalyticsExperienceLevel>
   <AnalyticsExperienceLevel>Standard</AnalyticsExperienceLevel>
   <AnalyticsExperienceLevel>Advanced</AnalyticsExperienceLevel>
</Level>
```

Every functionality level entry in the configuration file must have the five components shown in the following sections.

Name

Use a unique alphanumeric string for the functionality level name, enclosed within the <Name> and </Name> tags, such as <Name>Intermediate</Name>.

Role

The Role component defines the BIRT Viewer role assigned to the functionality level. The role is defined in the role tags in iv_config.xml. The default roles included for iv_config.xml are:

- All

- Active Portal Intermediate
- Active Portal Advanced
- Active Portal Administrator

Both the BIRT Viewer role and the functionality level must exist before you can assign the functionality level to a role. Enclose the security role name within <Role> and </Role> tags, such as <Role>Active Portal Intermediate</Role>.

Features

There are five features, which are described in Table 4-3.

Table 4-3 Features of functionality levels

Feature	Description
Channels	Provides access to channels. Channels are not relevant for Java Components.
Customization	Provides access to skin customization.
Documents	Provides access to files and folders.
Jobs	Supports submitting and accessing jobs. Jobs are not relevant for Java Components.
Search	Provides access to file and folder search. Search is not relevant for Java Components.

Enclose the feature within <FeatureID> and </FeatureID> tags. When you omit a feature from a functionality level, the corresponding side menu or banner item is not visible to anyone using that functionality level. For example, the Search feature is not available to the Basic functionality level, so the Search link does not appear in the banner for a user at the Basic functionality level.

Features

Functionality-level.config defines the features that are available to Java Component users as well as functionality levels. The following example shows the Documents feature definition from functionality-level.config:

```
<Feature>
   <ID>Documents</ID>
   <Labelkey>SBAR_DOCUMENTS</Labelkey>
   <Link>/getfolderitems.do</Link>
   <SmallIcon>/iportal/activePortal/images/
   filesfoldersicon16x16.gif
   </SmallIcon>
   <LargeIcon>/iportal/activePortal/images/filesfoldersicon.gif
   </LargeIcon>
</Feature>
```

The ID identifies the feature for Java Component. The label key appears on the side menu for Documents, Jobs, and Channels, or in the banner for Search and Customization. The link specifies the action that is executed for the feature. The small and large icons represent the feature in the side menu. Only the side menu features use the small and large icons.

Although you can customize the labels and links of all five features, do not change the <ID> or <Labelkey> tag values. Java Component uses these tags to identify the features and perform resource management. The Labelkey provides the resource to use for the feature's text label.

Changing the Link tag's value specifies a different action to execute. Changing the icon files changes the side menu's appearance. The small icons are used by the Tree View skin and are 16x16 pixels. The large icons are used by the Classic skin and are 32x32 pixels. The Tabbed skin does not use icons. Link and icon file names are relative to <context root>.

Subfeatures

A subfeature corresponds to an action you can perform using the Java Component user interface. Table 4-4 describes the subfeatures.

Table 4-4 Subfeatures of the features described in Table 4-3

Feature	Subfeature	Description
Channels	SubscribeChannel	Supports subscribing to channels. Channels are not relevant for Java Components.
Documents	CreateFolder	Supports creating folders when the user has the appropriate privileges.
Documents	DeleteFile	Supports deleting files when the user has the appropriate privileges.
Documents	DeleteFolder	Supports deleting folders when the user has the appropriate privileges.
Documents	ShareFile	Supports sharing files when the user has the appropriate privileges.
Jobs	JobPriority	Supports setting job priority, up to the user's maximum job priority. Jobs are not relevant for Java Components.
Jobs	SelfNotification WithAttachment	Supports e-mail notification for successful jobs. Jobs are not relevant for Java Components.
None	InteractiveViewing	Supports the BIRT Viewer.
None	AdvancedData	Used with BIRT Studio.

Specify one subfeature to a line and enclose each subfeature within <SubfeatureID> and </SubfeatureID> tags. Each subfeature is associated with a feature. You cannot include a subfeature in a functionality level if its corresponding feature is not available to that functionality level.

Analytics experience levels

Analytics experience levels are not relevant for Java Components.

Configuring Java Component locale using localemap.xml

Open <context root>\WEB-INF\localemap.xml to see a listing of the available locales in Java Component. Add locales to this file by following the exact format of the existing locales. To see each locale defined in the file, search for one of the following strings:

```
<Locale
```

or:

```
<DisplayName>
```

Searching for <Locale places the mouse pointer on the line with the ID for the locale. Searching for <DisplayName> places the mouse pointer on the line with the descriptive name for the locale.

In general, the locale names have the following syntax:

```
<language>_<country>
```

For example, ar_EG is Arabic (Egypt). When a single language is spoken in multiple countries, the language remains the same and the country can have several values. For example, en_US is the locale for English (United States) while en_AU is the locale for English (Australia). en_BZ is the locale for English (Belize). Some countries can have several locales, one for each language. For example, Canada has both en_CA for English (Canada) and fr_CA for French (Canada).

You specify a default locale for a custom web application in <context root>\WEB-INF\web.xml

Configuring Java Component locales using TimeZones.xml

Open <context root>\WEB-INF\TimeZones.xml to see a listing of the available time zones in Java Component. Add time zones to this file by following the exact format of the existing time zones. To see each time zone in the file, search for the following string:

```
<TimeZone
```

or:

```
<DisplayName>
```

Searching for <TimeZone places the mouse pointer on the line with the ID for the time zone. Searching for <DisplayName> places the mouse pointer on the line with the descriptive name for the time zone.

Some time zone names have short abbreviations for the ID. All time zone names have a full descriptive ID, such as Samoa Standard Time or Greenwich Standard Time. The DisplayName provides the relative time from Greenwich Standard Time and one or more locations that the time zone includes.

You specify a default time zone for a custom web application in <context root> \WEB-INF\web.xml.

Configuring the Actuate Java Component repository

Actuate Java Component provides the ability to organize, run, and view reports in a repository. You configure the security and repository for the Java Component using parameters in web.xml. The Java Component repository operates as a standalone or workgroup entity on the file system. Table 4-5 describes the configuration parameters for the Deployment Kit.

Table 4-5 Actuate Java Component web.xml parameters

Parameter name	Description
DEFAULT_WORKGROUP_ FUNCTIONALITY_ROLE	Specifies the functionality role for all users. The default value is Active Portal Intermediate.
DEFAULT_WORKGROUP_ SKIN	Specifies the skin for all users. The default value is Tree View.
REPOSITORY_CACHE_ TIMEOUT_SEC	Specifies how long a repository cache is valid. When the cache becomes invalid, any user actions refresh the cache for the duration. The default value is 900 seconds.
STANDALONE_ ACCESS_MANAGER	Specifies the class of the security manager that controls access to Java Component functionality. The default value is com.actuate.iportal.repository.jar.localfs.LocalAccessManager.
STANDALONE_ ALLOW_ANONYMOUS	Specifies whether access to Java Component functionality requires a user name. Valid values are true and false. The default value is true.
STANDALONE_ ANONYMOUS_ USERNAME	If the value of the STANDALONE_ALLOW_ANONYMOUS parameter is true, this parameter specifies the user name that denotes unauthenticated access to the Java Component application. The default value is anonymous.

Table 4-5 Actuate Java Component web.xml parameters

Parameter name	Description
STANDALONE_ HOME_FOLDER	Specifies the root folder for users' individual home folders in a repository. This folder is a subfolder of the repository root folder. The default value is /home.
STANDALONE_ PUBLIC_FOLDER	Specifies the root folder for public documents in a repository. This folder is a subfolder of the repository root folder. The default value is /public.
STANDALONE_ REPOSITORY_CLASS	Specifies the class that provides repository functionality to an Java Component application. The default value is com.actuate.iportal.repository.jcr.fs.FileSystemRepository.
STANDALONE_ REPOSITORY_FILE_ AUTHENTICATION	Specifies whether authentication controls access to Java Component functionality. Valid values are true and false. If the value is false, when an unknown user attempts to log in, the Java Component accepts the attempt and creates a home directory for the user. If the value is true, the Java Component uses the class defined by STANDALONE_ACCESS_MANAGER to validate the login attempt. The default value is false.
STANDALONE_ REPOSITORY_PATH	Path to the repository for Actuate Java Component files. The default value is set when Java Component is installed.

Configuring the BIRT Viewer and Interactive Viewer

The BIRT Viewer provides the ability to view a BIRT report. The Interactive Viewer supports modifying many aspects of the report's layout and formatting. These viewers are available as Java Components. Parameters in web.xml configure these viewers. For information on those configuration parameters, see *Working with Actuate BIRT Viewers*.

Configuring BIRT Studio

BIRT Studio is a report design tool that you use to design BIRT reports. This designer is available as a Java Component. Parameters in web.xml configure it. For information on those configuration parameters, see *Using BIRT Studio - Java Component Edition*.

Configuring BIRT Data Analyzer

The BIRT Data Analyzer extends the functionality of BIRT Interactive Viewer to perform analytics on a cross tab. You can configure performance enhancements for the Data Analyzer in web.xml. For information on those configuration parameters, see *Using BIRT Data Analyzer*.

Chapter

5

Actuate Java Component URIs

This chapter contains the following topics:

- Actuate Java Component URIs overview
- Actuate Java Component URIs quick reference
- Common URI parameters
- Java Component Struts actions
- Actuate Java Component URIs reference
- Actuate BIRT Viewer URIs reference

Actuate Java Component URIs overview

This chapter describes Actuate Java Component URIs. Java Component JSPs manage content. The following sections provide quick reference tables and detailed reference information about Actuate Java Component URIs. An Actuate Java Component URI is a directive to Actuate Java Component to perform an action, such as showing a list of files, rather than change the appearance of the application.

Java Component pages use the .do extension for the Struts action mapping to a page. The complete page name appears as part of the reference material. Actuate Java Component page and folder names are case-sensitive.

Actuate Java Component URIs quick reference

Table 5-1 lists the Actuate Java Component URIs. For more information about the Java Component directory structure, see "Understanding Java Component directory structure" in Chapter 3, "Creating a custom Java Component web application."

Table 5-1 Actuate Java Component URI pages

Actuate Java Component page	Description
about page	Displays information about Actuate Java Component.
authenticate page	Performs authentication and maintains user, cluster, and volume information.
banner page	Displays a banner at the top of each Actuate Java Component page.
browse file page	Provides file and folder browsing functionality for the submit request pages.
browse page	See browse file page.
delete file status page	Displays whether a file was successfully deleted.
detail page	Supports error handling and presenting object details.
drop page	Supports deleting files or cancelling running jobs.
error page	Retrieves an error message from the exception or the request and displays it.
execute report page	Submits a run report job request to the server.
executereport page	See execute report page.
getfiledetails page	See file or folder detail page.

Table 5-1 Actuate Java Component URI pages

Actuate Java Component page	Description
getfolderitems page	See file and folder index page.
home page	Provides the link from the My Folder button to the Actuate Java Component home page.
license page	Displays information about Actuate Java Component version and licensing.
login banner page	Provides the banner for the Actuate Java Component login page.
login page	Logs into the reporting web application.
logout page	Logs the user out of the current session and clears all user settings, such as filters.
page not found page	Displays an error message when a JSP is unavailable in Java Component.
parameters page	Presents a list of the request parameters.
viewer page for Actuate BIRT reports	Displays Actuate BIRT documents along with the toolbar.

Common URI parameters

All Actuate Java Component URIs have the parameters shown in Table 5-2. String values that are too long are truncated for all parameters. The web browser that you use determines the length of parameters. The common URI parameters support Actuate Java Component authentication using cookies.

Table 5-2 Common Actuate Java Component URI parameters

URI parameter	Description
forceLogin	True to force a login, false to display the login page. The default is false. The login operation is described in "Understanding the authentication process" in Chapter 9, "Using Actuate Java Component security."
iPortalID	The unique authentication ID assigned to the user upon successful login. Use this parameter in conjunction with the userID parameter to ensure that a user's personalized settings appear in the Java Component pages.

(continues)

Table 5-2 Common Actuate Java Component URI parameters (continued)

URI parameter	Description
locale	The current user's locale, such as U.S. English (en-US). Java Component locale names have the form nn_CC. nn is the language abbreviation and CC is the country code in both formats.
password	The password associated with the userID.
serverURL	Contains the URI that accesses the Actuate web application, such as http://Services:8000.
timezone	The current user's time zone.
userID	The user's unique identifier, required to log in to the repository. Use this parameter in conjunction with the iPortalID parameter to ensure that a user's personalized settings appear in the Java Component pages.
volume	The volume to which the user is connected.

The following Java Component URI shows most of the common URI parameters in use:

```
http://localhost:8080/iportal/getfolderitems.do
   ?folder=/Training&locale=en_AU&userID=Mike
   &password=pw123&serverURL=http://Seamore:8000
   &timeZone=Australia/Perth
```

This URI lists the contents of the Training folder on the application server named Seamore at port 8000. The locale is set to Australian English and the time zone is Australia/Perth (GMT plus eight hours). The user is Mike and the password is pw123. Note that the password is shown in plain text, as entered. If entered on a JSP or in a web form, it would be detected and encrypted.

Java Component Struts actions

The following tables summarize the global forwards and actions defined in struts-config.xml.

Table 5-3 lists the global forwards defined in struts-config.xml.

Table 5-3 Actuate Java Component global forwards

Action	Forward
authexpired	/login.do
browsefile	/browsefile.do

Table 5-3 Actuate Java Component global forwards

Action	Forward
error	/private/common/errors/errorpage.jsp
executedocument	/executedocument.do
executereport	/executereport.do
goto	/private/common/goto.jsp
login	/login.do
logout	/logout.do
skinerror	/private/common/errors/error.jsp
viewpage	/servlet/ViewPage
viewsoi	/viewsoi.do

Table 5-4 lists the action, input JSP, and forward name and path defined in struts-config.xml.

Table 5-4 Actuate Java Component actions

Action	Input JSP	Forward name path
/browsefile	/iportal/activePortal /private/newrequest /browse.jsp	name=success path=/iportal/activePortal/private /newrequest/browse.jsp
/cancelreport		name=Succeeded path=/iportal/activePortal/viewer /closewindow.jsp name=Failed path=/iportal/activePortal/viewer /closewindow.jsp?status=failed name=InActive path=/iportal/activePortal/viewer /closewindow.jsp?status=inactive
/deletefile		name=success path=/iportal/activePortal/private /filesfolders/deletefilestatus.jsp name=confirm path=/iportal/activePortal/private /filesfolders/confirm.jsp
/executedocument		name=success path=/executereport.do

(continues)

Table 5-4 Actuate Java Component actions (continued)

Action	Input JSP	Forward name path
/executereport	/private/newrequest /newrequest.jsp	name=viewbirt path=/iv name=viewreport path=/servlet/DownloadFile name=viewroi path=/viewer/viewframeset.jsp name=viewxlsreport path=/servlet name=wait path=/iportal/activePortal/private /newrequest/waitforexecution.jsp
/getfiledetails		name=success path=/iportal/activePortal/private /filesfolders/filedetail.jsp
/getfolderitems		name=success path=/iportal/activePortal/private /filesfolders/filefolderlist.jsp
/getportletfolderitems		name=success path=/iportal/portlets/filefolderlist /filefolderlistportlet.jsp
/iPortalLogin	/iportal/login.jsp	name=iPortalLoginForm path=/iportal/login.jsp name=landing path=/landing.jsp
/iv	/iportal/activePortal /private/newrequest /newrequest.jsp	name=iv path=/iv name=viewbirt path=/iv
/login	/iportal/activePortal /private/login.jsp	name=loginform path=/iportal/activePortal/private /login.jsp name=success path=/getfolderitems.do name=landing path=/landing.jsp
/logout		name=login path/login.do

Table 5-4 Actuate Java Component actions (continued)

Action	Input JSP	Forward name path
/submitjob	/iportal/activePortal /private/newrequest /newrequest.jsp	name=createquery path=/query/create.do name=query path=/query/submit.do name=success path=/iportal/activePortal/private /newrequest/submitjobstatus.jsp name=viewreport path=/servlet/DownloadFile name=viewroi path=/iportal/activePortal/viewer /viewframeset.jsp name=viewxlsreport path=/servlet
/tableList	/iportal/activePortal /private/parameters /table /tableparameters.jsp	name=close path=/iportal/activePortal/private /parameters/table/close.jsp name=tableRowEditor path=/iportal/activePortal/private /parameters/table/roweditor.jsp
/treebrowser		name=success path=/iportal/activePortal/private /filesfolders/treebrowser.jsp
/viewsoi	/iportal/activePortal /private/newrequest /newrequest.jsp	name=viewxlsreport path=/servlet
/waitforreport execution	/iportal/activePortal /private/newrequest /waitforexecution.jsp	name=success path=/iportal/activePortal/viewer /viewreport.jsp name=fail path=/iportal/activePortal/viewer /closewindow.jsp

Actuate Java Component URIs reference

This section provides the detailed reference for Actuate Java Component URIs. In the definitions, <context root> represents the name of your Actuate Java Component context root.

Table 5-5 lists the topics this chapter covers and the file names discussed in each topic. All pages are under the Java Component context root.

Table 5-5 Actuate Java Component pages

Topic	Java Component file
about page	iportal\activePortal\private\options\about.jsp
authenticate page	iportal\activePortal\authenticate.jsp
banner page	iportal\activePortal\private\common\banner.jsp
browse file page	browsefile.do iportal\activePortal\private\query\browse.jsp
delete file status page	iportal\activePortal\private\filesfolders \deletefilestatus.jsp
detail page	
■ error detail page	iportal\activePortal\errors\detail.jsp getfiledetails.do
■ file or folder detail page	iportal\activePortal\private\filesfolders\filedetail.jsp
drop page	
■ file or folder drop page	deletefile.do
error page	errors\error.jsp iportal\activePortal\private\common\errors\error.jsp
execute report page	executereport.do
home page	iportal\activePortal\private\common\breadcrumb.jsp
index page	
■ file and folder index page	getfolderitems.do iportal\activePortal\private\filesfolders \filefolderlist.jsp
license page	iportal\activePortal\private\options\license.jsp
list page	
■ file and folder list page	getfolderitems.do iportal\activePortal\private\filesfolders \filefolderlist.jsp
login banner page	iportal\activePortal\private\login_banner.jsp
login page	login.do iportal\activePortal\private\login.jsp

Table 5-5 Actuate Java Component pages

Topic	Java Component file
logout page	logout.do
page not found page	iportal\activePortal\errors\pagenotfound.jsp
parameters page	iportal\activePortal\private\newrequest \parameters.jsp
viewer page for Actuate BIRT Reports	IVServlet

about page

Displays the about page, containing information about Actuate Java Component. Called when the user chooses the About tab on the Options page.

The default about page for Java Component is similar to Figure 5-1.

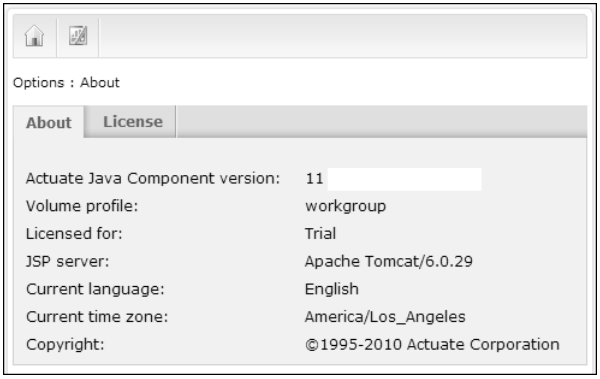

Figure 5-1 Java Component about page

Name <context root>\iportal\activePortal\private\options\about.jsp

Parameters The about page uses the common URI parameters.

Used by iportal\activePortal\private\options\optionspage.jsp

authenticate page

Performs user authentication and maintains the user, cluster, and volume information authentication data during the user's session. Pages that require validation of user credentials before permitting access to folders or files use the authenticate page. In Java Component, only pages for the DHTML Viewer use

the authenticate page. The remaining Java Component pages use the Struts framework for authentication.

Name <context root>\iportal\activePortal\authenticate.jsp

Parameters The authenticate page uses the common URI parameters.

Used by iportal\activePortal\errors\error.jsp
iportal\activePortal\viewer\closewindow.jsp
iportal\activePortal\viewer\print.jsp
iportal\activePortal\viewer\requestsearch.jsp
iportal\activePortal\viewer\saveas.jsp
iportal\activePortal\viewer\searchframe.jsp
iportal\activePortal\viewer\searchreport.jsp
iportal\activePortal\viewer\searchtoolbar.jsp
iportal\activePortal\viewer\viewdefault.jsp
iportal\activePortal\viewer\viewframeset.jsp
iportal\activePortal\viewer\viewnavigation.jsp
iportal\activePortal\viewer\viewreport.jsp
iportal\activePortal\viewer\viewtoc.jsp
iportal\activePortal\private\newrequest\waitforexecution.jsp

banner page

Provides the banner that appears across the top of all Actuate Java Component web pages. The default banner displays the Actuate logo, user name, cluster name, and volume name, and provides links for Logout, Options, and Help. The banner page obtains the user name, cluster name, and volume name from variables maintained by the authenticate page.

Name <context root>\iportal\activePortal\private\common\banner.jsp

Used by iportal\activePortal\private\login.jsp
iportal\activePortal\private\channels\channelnoticelist.jsp
iportal\activePortal\private\channels\channeloperationstatus.jsp
iportal\activePortal\private\filesfolders\deletefilestatus.jsp
iportal\activePortal\private\filesfolders\filedetail.jsp
iportal\activePortal\private\filesfolders\filefolderlist.jsp
iportal\activePortal\private\jobs\getjobdetails.jsp
iportal\activePortal\private\jobs\joboperationstatus.jsp
iportal\activePortal\private\jobs\selectjobs.jsp
iportal\activePortal\private\newrequest\newrequest.jsp
iportal\activePortal\private\newrequest\newrequest2.jsp
iportal\activePortal\private\newrequest\submitjobstatus.jsp
iportal\activePortal\private\options\options.jsp
iportal\activePortal\private\query\create.jsp

iportal\activePortal\private\query\execute.jsp

browse file page

Contains file and folder browsing functionality used by other submit request pages.

Name <context root>\browsefile.do

<context root>\iportal\activePortal\private\query\browse.jsp

Parameters workingFolder is the name of the folder for which to display contents in the folder browser window. The browse file page also uses the common URI parameters.

Used by iportal\activePortal\private\newrequest\browse.jsp
iportal\activePortal\private\query\browse.jsp

delete file status page

Summarizes the result of a deletion performed by the drop page and indicates whether a file was successfully deleted. The delete file status page includes authenticate to obtain user session data. Java Component performs the deletion as part of an action and then forwards to the delete file status page.

Name <context root>\iportal\activePortal\private\filesfolders\deletefilestatus.jsp

Used by Not applicable

detail page

Displays detailed information about Repository objects. There are two detail pages:

<context root>\iportal\activePortal\errors

<context root>\iportal\activePortal\filesfolders

error detail page

Provides a template error page that can be embedded in another page.

Name <context root>\iportal\activePortal\errors\detail.jsp

Used by iportal\activePortal\private\common\errors\error.jsp
iportal\activePortal\viewer\print.jsp

iportal\activePortal\viewer\saveas.jsp
iportal\activePortal\viewer\searchframe.jsp
iportal\activePortal\viewer\viewdefault.jsp
iportal\activePortal\viewer\viewtoc.jsp

file or folder detail page

Displays detailed information about the selected viewable folder or file. Users request file details by choosing the magnifying glass icon to the right of files listed on the folder page, or folder details by choosing the magnifying glass icon to the right of the folder name in the breadcrumb. Users can request another viewable document or delete the current file or folder from the file or folder detail page. filedetail.jsp uses the HTML code in <context root>\iportal\activePortal \private\filesfolders\filedetailcontent.jsp to display the information.

The default detail page for the Home folder is similar to Figure 5-2.

Figure 5-2 Home folder detail page

Name <context root>\getfiledetails.do

<context root>\iportal\activePortal\private\filesfolders\filedetail.jsp

Parameters Table 5-6 describes the parameters for the file or folder detail page. The file or folder detail page also uses the common URI parameters.

Table 5-6 File or folder detail URI parameters

URI parameter	Description
name	The full path name of the repository object for which to show details. This parameter is ignored if objectID is also specified.
objectID	The repository object's unique identifier.
version	The repository object's version number. The default is the latest version.

Used by Not applicable

drop page

Deletes one or more files or folders.

file or folder drop page

Deletes the specified file or folder. The file or folder drop page includes the authenticate page to obtain user session data.

Name <context root>\deletefile.do

Parameters Table 5-7 describes the parameters for the file or folder drop page. The file or folder drop page also uses the common URI parameters.

Table 5-7 File or folder drop URI parameters

URI parameter	Description
ID	The unique identifier of the repository object to delete.
name	The full path name of the repository object to delete. Multiple name parameters, to delete more than one file or folder at a time, are allowed. This parameter is ignored if ID is also specified.
redirect	URI to which to redirect the job deletion page. The default redirect page is processedaction_status.

Used by Not applicable

error page

Displays the specified error message. Java Component uses two pages. All Java Component code uses <context root>\iportal\activePortal\private\common \errors\error.jsp.

Name <context root>\iportal\activePortal\errors\error.jsp

<context root>\iportal\activePortal\private\common\errors\error.jsp

Used by iportal\activePortal\private\login.jsp
iportal\activePortal\private\common\closewindow.jsp
iportal\activePortal\private\common\sidebar.jsp
iportal\activePortal\private\common\errors\errorpage.jsp
iportal\activePortal\private\options\options.jsp
iportal\activePortal\private\query\create.jsp
iportal\activePortal\private\query\execute.jsp
iportal\activePortal\private\templates\template.jsp

iportal\activePortal\viewer\closewindow.jsp
iportal\activePortal\viewer\print.jsp
iportal\activePortal\viewer\saveas.jsp
iportal\activePortal\viewer\searchframe.jsp
iportal\activePortal\viewer\searchreport.jsp
iportal\activePortal\viewer\viewframeset.jsp

execute report page

Submits a run report job request.

When executing a report job or query, a Cancel button appears after a specified wait time passes. To change the time, set the EXECUTE_REPORT_WAIT_TIME configuration parameter in the appropriate Actuate Java Component configuration file.

For reports that accept run-time parameters, you can set the parameter in the URL by adding an ampersand (&), the parameter name, and an equal (=) sign, followed by the parameter value in quotes. The following URL illustrates running a BIRT report immediately with the Territory run-time parameter set to EMEA:

```
http://localhost:8080/iportal/executereport.do?__requesttype=
   immediate&__executableName=%2fPublic%2fBIRT and BIRT Studio
   Examples%2fSales by Territory.rptdesign&userid=Administrator
   &__saveOutput=false&Territory="EMEA"&invokeSubmit=true
```

The execute report page also accepts dynamic filter parameters for BIRT Reports in the URL, but the value of the parameter must form a complete expression, such as &Territory=([Territory] = "EMEA").

Name <context root>\executereport.do

Parameters Table 5-8 describes the parameters for the execute report page. The execute report page also uses the common URI parameters.

Table 5-8 Execute Report URI parameters

URI parameter	Description
__ageDays	Use with __ageHours to determine how long output objects exist before they are automatically deleted. Use only if __archivePolicy is set to Age. __ageDays can be any positive number.
__ageHours	Use with __ageDays to determine how long output objects exist before they are automatically deleted. Use only if __archivePolicy is set to Age. __ageHours can be any positive number.
__executableName	The name of the executable file for this request.

Table 5-8 Execute Report URI parameters

URI parameter	Description
invokeSubmit	Controls whether the browser is redirected to the parameter screen or whether the report job is run immediately. If true, the report job is executed without displaying the parameters. If false, the parameters are displayed. False is the default.
__outputDocName	The name and path of the resulting BIRT document. This parameter is only usable for BIRT reports when the BIRT_SAVE_REPORT_DOCUMENT_ENABLED parameter is set to true in web.xml. If the given path is absolute, then executereport saves the report to that path. If the given path is relative, then executereport saves the report to the path set in the BIRT_SAVE_REPORT_DOCUMENT_PATH web.xml parameter.
__priority	Specifies the job submission priority. Values are High, Medium, and Low.
__priorityValue	Specifies a number ranging from 1 to 1000 and corresponding to the job submission priority. Only specify values allowed by your functionality level.
__progressive	Indicates whether to display the report document after it generates. If false, the report document displays after it generates. If true, the report document displays progressively, as it generates.
__serverURL	Contains the URI that accesses the JSP engine, such as http://Services:8000.
__wait	If "wait", Java Component waits for the report generation to be completed before displaying it. If "nowait", Java Component displays the first page right away even if the report job is not completed.

For example, the following URL executes the Sales By Territory.rptdesign report immediately with the Territory run-time parameter set to EMEA:

```
http://localhost:8080/iportal/executereport.do?
   __requesttype=immediate&__executableName=%2fPublic%2fBIRT and
   BIRT Studio Examples%2fSales by Territory.rptdesign&
   userid=anonymous&__saveOutput=false&Territory="EMEA"&
   invokeSubmit=true
```

The following parameter names are reserved for internal use only by the execute report page:

- doframe

- inputfile
- jobType
- name
- selectTab

Used by Not applicable

home page

Provides two sets of links. On the right side it provides a graphical and a text shortcut link from the My Folder button to the current user's Actuate Java Component home folder. If the Java Component installation includes BIRT Studio, there is another shortcut link, BusinessReport Studio, to the BIRT Studio. On the left side, it provides the links and other text for the breadcrumb, or path from the repository root to the current folder.

Users access their home page by choosing the My Folder link below the Actuate Java Component page banner.

Figure 5-3 shows the default My Folder and breadcrumb links.

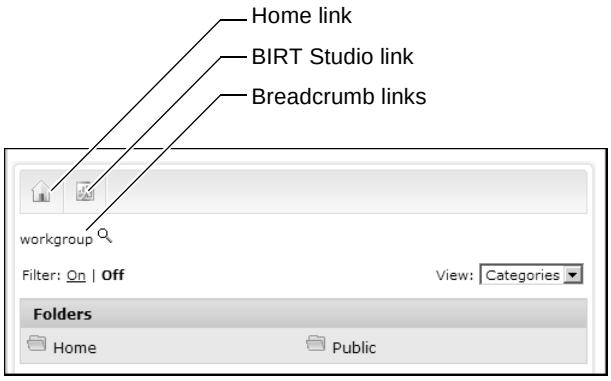

Figure 5-3 My Folder and breadcrumb links

Name <context root>\iportal\activePortal\private\common\breadcrumb.jsp

Used by iportal\activePortal\private\skins\tabbed\templates\mypagetemplate.jsp
iportal\activePortal\private\skins\tabbed\templates\template.jsp
iportal\activePortal\private\skins\classic\templates\template.jsp
iportal\activePortal\private\skins\treeview\templates\template.jsp

index page

Provides the entry point and structure for the parts of Actuate Java Component generated from multiple files.

file and folder index page

The default entry point to the Actuate Java Component web application. The file and folder index page provides the entry point and structure to support the Files and Folders functionality. The structure is a table that Actuate Java Component uses to format and present files and folders data. Page content varies depending on the Actuate Java Component directive.

The file and folder index page uses the banner page to provide the reporting web page banner. filefolderlist.jsp uses the HTML code in <context root>\iportal\activePortal\private\filesfolders\filefolderlistcontent.jsp to display files and folders data.

Name <context root>\getfolderitems.do

<context root>\iportal\activePortal\private\filesfolders\filefolderlist.jsp

Parameters Table 5-9 describes the parameters for the file and folder index page. The file and folder index page also uses the common URI parameters.

Table 5-9 File and folder index URI parameters

URI parameter	Description
startUpMessage	Specifies a message to appear when Actuate Java Component calls this page.
subpage	Specifies the content of the page. Possible values are: ■ _list: include list ■ _detail: include detail Specifying any other value for subpage invokes the page not found page.

license page

Displays the license page, containing information about Actuate Java Component version and licensing. Called when the user chooses the License tab on the Options page.

The default license page for Java Component is similar to Figure 5-4.

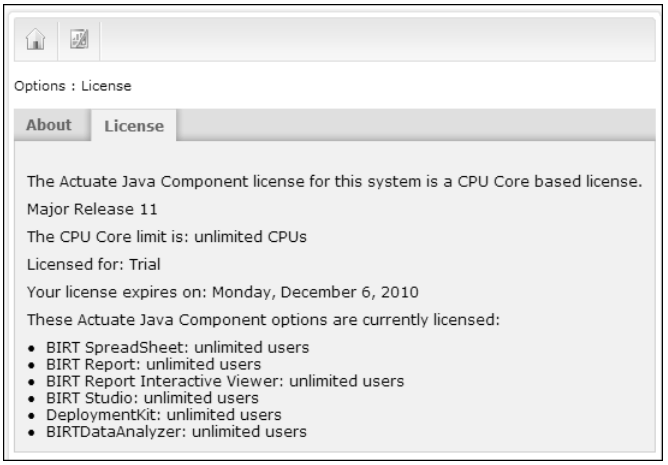

Figure 5-4 Java Component license page

Name <context root>\iportal\activePortal\private\options\license.jsp

Parameters The license page uses the common URI parameters.

Used by iportal\activePortal\private\options\optionspage.jsp

list page

Lists files in a container, such as a folder.

file and folder list page

Presents a list of objects that reside in the current working repository folder. Users request folder listings by choosing links on the reporting web page. The file and folder list page includes a filter section where users specify criteria for viewing report documents.

When users access a repository for the first time, Actuate Java Component displays their home folder, if they have one, or the top folder in the repository. All files and folders in that folder that they have permission to view appear in the Actuate Java Component listing page. Users can specify a filter to choose the types of files to view.

Name <context root>\getfolderitems.do

<context root>\iportal\activePortal\private\filesfolders\filefolderlist.jsp

Parameters Table 5-10 describes the parameters for the file and folder list page. The file and folder list page also uses the common URI parameters.

Table 5-10 File and folder list URI parameters

URI parameter	Description
applyFilter	If true, apply filter. If false, filter not applied. To use the showDocument, showExecutables, and showFolder parameters, applyFilter must be true.
filter	The filter specifying the file and folder names to list. Filter is a string. The default is "".
folder	The folder for which to list the contents. Folder name is a string. If no folder is specified, List uses the last working folder known for the session if cookies are enabled. If cookies are not enabled, List uses the user's home folder as specified in the user settings.
onlyLatest	If true, show only the latest version of a file if multiple versions exist. If false, show all versions of a file if multiple versions exist. The default is false.
resetFilter	Any non-null value for resetFilter causes the filter to return to its original state. Users can reset the filter by choosing the Default button on the listing page.
showDocument	If true, show all viewable documents. If false, do not show viewable documents. The default is true. To use this parameter, applyFilter must be true.
showExecutables	If true, show all report executables. If false, do not show report executables. The default is true. To use this parameter, applyFilter must be true.
showFolders	If true, show all folders. If false, do not show folders. The default is true. To use this parameter, applyFilter must be true.

Used by Not applicable

login banner page

Displays the Actuate Java Component web application banner. Banner elements include the company logo, system name, and help link.

Name <context root>\iportal\activePortal\private\login_banner.jsp

Used by iportal\activePortal\private\login.jsp

login page

Displays the Actuate Java Component login page for logging in to the Actuate Java Component web application. The login page includes the login banner page to display the Actuate Java Component application banner.

Name <context root>\login.do

<context root>\iportal\activePortal\private\login.jsp

Parameters Table 5-11 describes the parameters for the login page. The login page also uses the common URI parameters.

Table 5-11 Login page URI parameters

URI parameter	Description
loginPostback	False to display the login page and true to display the destination page instead of the login page if the login is successful.
targetPage	Specify a relative URI to which login redirects the user on successful login. The default is the file and folder list page.

Used by Not applicable

logout page

Ends the user's Actuate Java Component session. The logout page gathers the user's session information, clears it, and returns the user to the login page.

Name <context root>\logout.do

Parameters Table 5-12 describes the parameters for the logout page. The logout page also uses the common URI parameters.

Table 5-12 Logout page URI parameters

URI parameter	Description
daemonURL	Contains the URI that accesses the Process Management Daemon, such as http://Server:8100.
user	The name of the user to log out. Either user or the common URI parameter authID must be specified. If authID is specified, user is ignored.

Used by Not applicable.

page not found page

Displays an error message when Actuate Java Component cannot find the page that a user specifies. This page is a Java Component page only.

Name <context root>\iportal\activePortal\errors\pagenotfound.jsp

Used by Not applicable

parameters page

Displays report job parameters. Parameters include the headline, output file name, and report executable file name. Users access the parameters list by choosing Parameters.

Parameters looks like Figure 5-5.

Figure 5-5 Parameters page

Name <context root>\iportal\activePortal\private\newrequest\parameters.jsp

Used by iportal\activePortal\private\newrequest\newrequestpage

Actuate BIRT Viewer URIs reference

To view and interact with Actuate BIRT reports, you use the Actuate BIRT servlet. All BIRT Viewer options and varieties use the same URL. For detailed information about the BIRT servlet URL, see *Working with Actuate BIRT Viewers*.

parameters page

Chapter

6

Actuate Java Component JavaScript

This chapter contains the following topics:

- Actuate Java Component JavaScript overview
- Actuate Java Component JavaScript reference

Actuate Java Component JavaScript overview

This section describes the Actuate Java Component JavaScript files. Actuate Java Component JavaScript files provide functionality and dynamic content to Actuate Java Component web applications. Actuate Java Component JavaScript files reside in <context root>\iportal\js.

Actuate Java Component JavaScript reference

Table 6-1 lists and describes the Actuate Java Component JavaScript files.

Table 6-1 Java Component JavaScript files

Name	Description
allscripts.js	Defines global variables, resources, and common methods such as deleteFile and viewActiveRequests
array.js	Contains functionality for handling arrays and array elements
browsertype.js	Determines the web browser in use and provides functionality appropriate to the browser, such as opening a file in a new window and capturing a keystroke event
converter.js	Provides character encoding
cookie.js	Provides cookie functionality, including reading, writing, and clearing browser cookies
drift.js	Adjusts layers and window display for Java Component
encoder.js	Contains the encode and unencode methods
help.js	Provides context-sensitive help functionality for Java Component
layer.js	Provides layer functionality, such as createLayer, deleteLayer, getWidth, showLayer
popupmenu.js	Defines the methods for manipulating pop-up menus
report.js	Provides the JavaScript components for report viewing
resize.js	Provides the JavaScript component for resizing a page for Java Component
strutscommon.js	Provides JavaScript components for using the Struts framework with Java Component

Chapter

7

Actuate Java Component servlets

This chapter contains the following topics:

- Java Component Java servlets overview
- Java Component Java servlets quick reference
- Java Component Java servlets reference

Java Component Java servlets overview

Java servlets extend web server functionality. Java Component uses Java servlets to manage binary content and to perform tasks such as uploading and downloading binary files. Actuate provides an abstract framework of servlets that provide common functionality to Java Component. You cannot modify the Actuate Java servlets.

About the base servlet

All Actuate servlets derive from the base servlet:

```
com.actuate.reportcast.servlets.AcServlet
```

The base servlet has no URI parameters. It provides Actuate servlets with the functionality for performing the following tasks:

- Parse and validate parameters specified in Java Component URI directives.
- Create XML API structures based on Actuate Java Component requests.
- Submit XML streams to the Actuate SOAP API.
- Handle responses from the Actuate SOAP API, including detecting errors.
- Store constant session information, such as the name space and SOAP endpoint.
- Read from and write to cookies.
- Stream report data or errors to the web browser.

Invoking a servlet

You invoke servlets using the following syntax:

```
http://<server>:<port>/<context root>/servlet/<servlet alias>
```

where

- server is the name of the machine hosting the application server.
- port is the port on which the application server listens for requests.
- context root is the Java Component context root.
- servlet is a keyword indicating that a servlet follows.
- servlet alias is the name to which the servlet is mapped in the Java Component installation's web.xml file. A typical location for web.xml is <context root>\WEB-INF\web.xml.

Servlet names are case-sensitive. Do not modify the servlets, their names, or their mapping in web.xml.

Java Component Java servlets quick reference

Table 7-1 lists the Java Component Java servlets.

Table 7-1 Actuate Java Component servlets

Java Component servlet	Description
ExecuteReport servlet	Submits a request to run a report
Interactive Viewer servlet	Displays an Actuate BIRT report document

Java Component Java servlets reference

This section provides the detailed reference for Java Component servlets.

ExecuteReport servlet

Submits a request to the web service to run a report job. The execute report servlet is equivalent to do_executereport.jsp. This servlet supports executing spreadsheet reports. Excel does not support executing reports using do_executereport.jsp.

Name com.actuate.reportcast.servlets.ExecuteReportServlet

Invoke the ExecuteReport servlet as:

http://<web server>:<port>/<context root>/servlet/<report executable>

where the report executable is the ROI or SOX report file to execute.

URL parameters Table 7-2 lists and describes the parameters for the ExecuteReport servlet.

Table 7-2 ExecuteReport URI parameters

URI parameter	Description
__ageDays	Use with __ageHours to determine how long output objects exist before they are deleted. Use only if __archivePolicy is set to age. __ageDays can be any positive number.

(continues)

Table 7-2 ExecuteReport URI parameters (continued)

URI parameter	Description
__ageHours	Use with __ageDays to determine how long output objects exist before they are deleted. Use only if __archivePolicy is set to age. __ageHours can be any positive number.
__archiveBeforeDelete	Indicate whether to archive the output objects of the current request before deleting them, according to __archivePolicy's setting. Set to true to archive objects before deleting them. The default value is false. This parameter has no effect if __archivePolicy is set to folder.
__archivePolicy	The archive policy to implement for the objects created as output for the current request. Values are folder, age, and date. Set folder to use the archive policy that is already set for the folders to which the output is distributed. Set age to delete objects older than a specific time period. Set date to delete objects on a specific date.
__dateToDelete	The date on which to delete the output objects of the current request. Use only if __archivePolicy is set to date. __dateToDelete must be a date in a locale-specific format. The default format is mm/dd/yyyy.
__folder	The path name of the folder that contains the report executable.
__headline	A descriptive tag line for a report. Appears on the Channel Contents page. Use the character string %20 to represent spaces in the headline string.
__limit	Indicate whether to limit the number of versions of the output files for the current request. Set __limit to limit to curtail the number of versions. Any other value means that the number of versions is unlimited.
__limitNumber	The number of versions to which to limit the output files for the current request. Use only if __limit is set to limit. __limitNumber can be any positive number.
__outputName	Specifies a name for the report output.
__overwrite	If true, overwrite any existing output. If false, do not overwrite existing output.
__priority	Specifies the job submission priority. Values are High, Medium, and Low.

Table 7-2 ExecuteReport URI parameters (continued)

URI parameter	Description
__priorityValue	Specifies a number corresponding to the job submission priority.
__redirect	Specifies a relative or absolute URL to go to after do_executereport.jsp submits the report. The default is Submittedjob_Status.jsp.
serverURL	Contains the URL that accesses the JSP engine, such as http://Services:8080.
__timeToDelete	Specifies a time at which to delete an archived report document. Applies only to scheduled report jobs.
__versionName	Contains a string value for the new version name of this report. The value can include a date/time expression enclosed in braces, {}, to ensure a unique version name.
volume	Contains a string value specifying the volume for this report.
__wait	If "wait", Java Component waits for the report generation to be completed before displaying it. If "nowait", Java Component displays the first page right away even if the report is not completed.

Interactive Viewer servlet

Displays an Actuate BIRT report document with tools to affect the document and design files. The viewer has two modes, standard and interactive.

The Standard Viewer displays the report with toolbar options to save, print, show the TOC, and launch interactive mode, as shown in Figure 7-1.

Figure 7-1 Standard Viewer

The Interactive Viewer displays the report with toolbar options to navigate the report and provides context menus to edit and format report elements, as shown in Figure 7-2.

Figure 7-2 Interactive Viewer

The viewer supports the rptdocument file format.

Name

com.actuate.iv.servlet.IVServlet

Invoke the Interactive Viewer servlet as:

http://<web server>:<port>/<context root>/iv

URI parameters

Table 7-3 lists and describes the URI parameters for the Interactive Viewer servlet.

Table 7-3 IV URI parameters

URI parameter	Description
__bookmark	Name of the element of a report to display instead of the whole report file
__floatingfooter	Boolean value to add a margin under the footer
__format	A format for the displayed report: ■ pdf: Adobe pdf ■ xls: MS Excel ■ doc: MS Word ■ ppt: MS PowerPoint ■ ps: PostScript ■ html: HTML ■ flashchartsxml: display a Flash object for a fusion chart ■ flashgadgetsxml: display a Flash gadget for a fusion chart ■ reportlet: used together with __bookmark to show a particular part/element of the report
__from_page_range	The page range of a report to display

Table 7-3 IV URI parameters

URI parameter	Description
__from_page_style	The page style to use for a report in pdf or ps formats ■ auto: page size and content size remains the same ■ actuateSize: change the page size to fit the content ■ fitToWholePage: change the content size to fit the page size Used with the __format parameter
__imageid	Name of the report file to display
__instanceid	Name of the report file to display
__launchIV	Boolean value that enables Interactivity
__locale	Code for a locale
__page	A number for a page to render
__report	Name of the report file to display
__rtl	Name of the report file to display
repositoryType	The name of the object to download
serverURL	Contains the URL that accesses JSP engine, such as http://ESL02835:8000

Chapter

8

Actuate Java Component JavaBeans

This chapter contains the following topics:

- Java Component JavaBeans overview
- Java Component JavaBeans package reference
- Java Component JavaBeans class reference

Java Component JavaBeans overview

This section describes the Java Component JavaBeans. Java Component JavaBeans provide functionality, business logic, and dynamic content to Java Component web applications. Java Component JavaBeans are in aciportal.jar, which resides in <context root>\WEB-INF\lib.

Java Component JavaBeans package reference

Table 8-1 lists and describes the Actuate packages used in Java Component.

Table 8-1 Java Component packages

Package	Contents
com.actuate.activeportal .beans	JavaBeans that maintain information used by the Action classes.
com.actuate.activeportal .forms	JavaBeans derived from the Jakarta Struts org.apache.struts.action.ActionForm object. These JavaBeans store and validate the request parameters in HTTP requests.
com.actuate.activeportal.list	An interface, IContentList, that defines the behavior of lists of items such as files and channels. Several classes in com.actuate.activeportal.forms use this interface.

Java Component JavaBeans class reference

Documents

Table 8-2 lists and describes Java Component com.actuate.activeportal.forms classes that support the Document pages.

Table 8-2 Document classes

Class	Description
BrowseFileActionForm	Supports browsing through the available files, including using filters to search.
FileListActionForm	Retrieves a list of folders or files. This ActionForm supports setting filters specifying characteristics of objects. Stores the most recent list of items.

Table 8-2 Document classes

Class	Description
GeneralFilterActionForm	The base ActionForm for several other ActionForms. Provides methods that handle filters. For example, you can request all folders and only the most recent version of all executable files.
GetFileDetailsActionForm	Stores the details of a file or folder. AcGetFileDetailsAction gets the details and stores them in this JavaBean.

General

Table 8-3 describes the Java Component com.actuate.activeportal.beans class that supports general functionality.

Table 8-3 General bean class

Class	Description
LinkBean	Generates an HTML link tag using the link, linkAttributes, and text properties. By default, the link class is hyperlink. After setting these properties, use the toString() method to generate an HTML link tag in the following format: `<A HREF="link" linkAttributes>text</A>`

Table 8-4 lists and describes Java Component com.actuate.activeportal.forms classes that support general functionality.

Table 8-4 General forms classes

Class	Description
BaseActionForm	The base ActionForm for all other Java Component ActionForms. Provides methods related to postback.

Jobs

Table 8-5 lists and describes Java Component com.actuate.activeportal.forms classes that support jobs.

Table 8-5 Job classes

Class	Description
JobActionForm	The base ActionForm for QueryActionForm and SubmitJobActionForm. Stores values used in submitting a job or query, such as the document, parameters, and schedule.
SubmitJobActionForm	Contains the information for submitting a job from the requester page. This class extends JobActionForm.

Chapter

9

Using Actuate Java Component security

This chapter contains the following topics:

- About Actuate Java Component security
- Protecting corporate data
- Understanding the authentication process
- Customizing Java Component authentication
- Creating a custom security adapter

About Actuate Java Component security

A reporting web application is accessible to any user who has a web browser and the URI for the application. This chapter discusses the Actuate Java Component security features and how to use them to:

- Ensure that users access only those objects in the repository for which they have permission.
- Protect sensitive reports.

The types of security you can provide for Actuate Java Component are:

- Default application server authentication. The Deployment Kit does not have any security implemented automatically. The application server controls access to the file system and web content.
- User authentication using the iPortal Security Extension (IPSE). Use IPSE to customize and control the user login and authentication process. For details about implementing custom user authentication, see "Customizing Java Component authentication," later in this chapter.

Protecting corporate data

An Actuate Java Component provides a structured content generation solution for web applications. Deploying Actuate applications developed for the internet, such as Java Component, requires planning for network security.

Internet applications support access to information within an organization from outside that organization. Because the organization's internal network is connected to the internet, there is the risk of unauthorized access to the corporate network and to the data that resides on that network.

Organizations use one or a combination of the technologies described in the following sections to prevent unauthorized access to the corporate network and protect authentication transactions from intrusion.

Protecting corporate data using firewalls

Typically companies use firewalls to prevent unauthorized access to corporate networks and data. A firewall is a system or group of systems that restrict access between two networks, such as an organization's internal network and the internet. Firewalls keep unauthorized users out. As a result, firewalls prevent damage caused by malicious programs such as worms and viruses from spreading to other parts of your network. At the same time, firewalls allow legitimate business to tunnel through the firewall and be efficiently conducted on your network.

Firewalls can be used to restrict access between two internal networks, for example, the accounting and engineering networks. Security teams configure firewalls to allow traffic using specific protocols, such as HTTP, over specific network addresses and ports. Be sure that your firewall allows access for the Actuate Java Component ports.

Protecting corporate data using Network Address Translation

Companies also use Network Address Translation (NAT). NAT routers and software support private networks using unregistered, private IP (Internet Protocol) addresses to connect to the internet.

Protecting corporate data using proxy servers

Proxy servers, specialized web servers or hardware that operate on or behind a firewall, improve efficient use of network bandwidth and offer enhanced network security. For more information about proxy servers and Actuate Java Component, see Chapter 1, "Introducing Actuate Java Components."

Understanding the authentication process

The authentication process involves the following steps, in this order:

- A user or client makes a request by choosing a link on an Actuate Java Component page or by typing an Actuate Java Component URI in a web browser. The Java Component application processes the request.
- If a custom security adapter parameter is set in the web.xml file, the Java Component attempts to load the custom security adapter class. If the class loads successfully, the following steps occur:
  - The Java Component calls the custom security adapter's authenticate( ) method with the parameters that the browser sent.
  - The authenticate( ) method performs the custom validation.
  - The Java Component calls the getUserName( ), getPassword( ), and getUserHomeFolder( ) methods to retrieve the user information the Actuate web service requires.
  - Optionally, the Java Component calls the getExtendedCredentials( ) method. If this method returns null, there are no extended credentials to send to the web service.
  - The application server provides the necessary information to the access manager.

Customizing Java Component authentication

To customize Actuate Java Component authentication, complete the following general tasks:

- Write a custom security class that extends an IPSE class, implementing all the appropriate methods. Your class must be thread-safe and cannot depend on any one thread handling a particular request.
- Compile, compress, and copy the new class to the lib directory for your Java Component application. The lib directory for your Java Component application resides on a path like this one:

  ```
  <context root>\WEB-INF\lib
  ```

- Set the value of the parameter in the <context root>\WEB-INF\web.xml file to the fully qualified name of your custom security class. A fully qualified name contains both the package and class names. For single sign-on authentication, set the SECURITY_ADAPTER_CLASS configuration parameter value to the custom security class.

Creating a custom security adapter

The Java Component security adapter is designed so that other applications can authenticate users and log into Java Component using a URL. When a URL activates a custom Java Component security adapter, access is granted based on the security adapter's logic. A Java Component security adapter establishes an additional layer of logic to the existing Java Component, as shown in Figure 9-1.

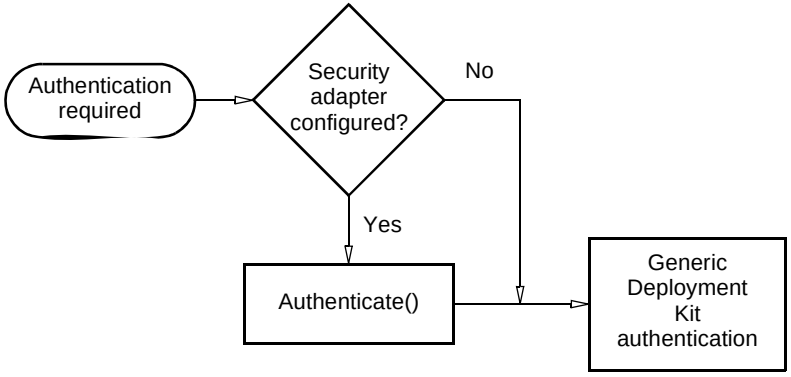

Figure 9-1 Java Component security model with an optional security adapter

The Java Component Login module creates a Properties object that contains the values of configuration settings that correspond to the class's public fields before

calling the authenticate( ) method. These values are gathered from the entries in the <context_root>\WEB-INF\web.xml configuration file.

To create a custom security adapter, perform the following steps:

- Ensure that your application can access the IPSE Java classes.
- Create a Java class that implements the custom security adapter class for IPSE.
- Compile, compress, and move the new class into the class libraries for the Java Component.
- Set the Java Component configuration file web.xml to use the new class.
- Deploy the Custom Security Adapter.

Accessing the IPSE Java classes

The Java Component library, com.actuate.iportal.jar, contains the IPSE Java classes. This library is located in the lib subdirectory in the Java Component installation. The class, com.actuate.iportal.security.iPortalSecurityAdapter, in this library provides the framework for custom authentication. A custom security adapter providing an IPSE implementation extends this class.

Specifically, the JRE needs to access the following JAR files:

- <context root>\WEB-INF\lib\com.actuate.iportal.jar
- <context root>\WEB-INF\lib\xercesImpl.jar

Additionally, the JRE needs to access the following JAR files from the application server:

- servlet-api.jar
- jsp-api.jar

For example, when deploying a Java Components application on Tomcat 6.0, these JAR files are in the <Apache Installation Directory>/Tomcat 6.0/lib directory.

Creating a custom security adapter class

Extend the iPortal security adapter class to customize authentication. The iPortal security adapter requires access to the following libraries:

- javax.servlet.http.*
- com.actuate.iportal.security.iPortalSecurityAdapter

iPortalSecurityAdapter provides a set of empty methods. Extend this class and override any of the methods to provide custom IPSE authentication. To establish a secure session with Information Console using a custom security adapter, the following methods are required:

- A constructor
- authenticate()
- getPassword()
- getUserName()
- getUserHomeFolder()

The login module of the Java Component calls methods in the custom security class to perform authentication and to retrieve login credentials. The authenticate( ) method returns a boolean value to indicate whether the login credentials provided are acceptable. The getter methods return the authenticated credentials. Each user name and password must correspond to an authentic user account. For example, to support a URL that authenticates using a single parameter, code, override authenticate( ) to retrieve the parameter from the HttpServletRequest and set the user name, password, and home folder as in the following class:

```
import javax.servlet.http.*;
import com.actuate.iportal.security.iPortalSecurityAdapter;

public class SecurityCode extends
   com.actuate.iportal.security.iPortalSecurityAdapter {
   private String userName = null;
   private String password = null;
   public SecurityCode( ) {}

   public boolean authenticate(
      HttpServletRequest httpservletrequest) {
      String param = httpservletrequest.getParameter("code");
      boolean secured = true;
      if ("12345".equalsIgnoreCase( param )) {
         userName = "user1";
         password = "user1";
      } else if ("abc".equalsIgnoreCase( param )) {
         userName = "BasicUser";
         password = "";
      } else {
         secured = false;
      }
      return secured;
   }

   public String getUserName() { return userName; }
   public String getPassword() { return password; }
   public String getUserHomeFolder() { return userName; }
   public byte[] getExtendedCredentials() { return null; }
   public boolean isEnterprise() { return false; }
}
```

Users or pages attempting to authenticate a session with a Java Components application that implements the security adapter above must use URL parameters defined in the authenticate method. Because Java Components have no native security, a custom adapter becomes the sole security module.

How to build the IPSE application

1. Compile the IPSE application. Use a command similar to this one in a console window:

   ```
   javac SecurityCode.java
   ```

2. Create a JAR file to contain the IPSE application. Use a command similar to this one in a console window:

   ```
   jar cvf SecurityCode.jar SecurityCode.class
   ```

3. Using Windows Explorer, copy SecurityCode.jar to this directory:

   ```
   <your application context root>\WEB-INF\lib
   ```

How to deploy the IPSE application

1. Using a UTF-8 compliant code editor, open the following file:

   ```
   <your application context root>\WEB-INF\web.xml
   ```

2. Navigate to the parameter name SECURITY_ADAPTER_CLASS.

3. Change the param-value parameter of the SECURITY_ADAPTER_CLASS to the fully qualified class name of your security manager class. Use an entry similar to this one:

   ```
   <param-name>SECURITY_ADAPTER_CLASS</param-name>
   <param-value>SecurityCode</param-value>
   ```

4. Save and close web.xml.

5. To have Actuate Java Component read the new security class from the web.xml file, restart the application server or servlet container.

Understanding a security adapter class

Implement the security manager by writing a class that extends com.actuate.iportal.security.iPortalSecurityAdapter. This class contains the following methods.

authenticate( )

Syntax boolean authenticate(javax.servlet.http.HttpServletRequest request)

Description Required method that evaluates the current user's security credentials. The Login module calls authenticate( ) to validate the current user's security credentials. If authenticate( ) returns false, the user is redirected to the login page.

Returns True for successful credential evaluation and false otherwise.

Throws An AuthenticationException indicating the reason for the failure, if credential evaluation is not successful.

getExtendedCredentials( )

Syntax byte[] getExtendedCredentials( )

Description Retrieves the current user's extended security credentials.

Returns A byte array representing any extended credentials for the iServer to use to authenticate the user, or null if there are no extended credentials to evaluate.

getPassword( )

Syntax String getPassword( )

Description Required method that retrieves the current user's password. The Login module calls getPassword( ) and uses the password to establish a connection to the application server and file system.

Returns A string that is the password to use to establish the connection.

getUserHomeFolder( )

Syntax String getUserHomeFolder( )

Description Retrieves the current user's home folder. The Login module calls getUserHomeFolder( ) to access the user's files.

Returns A string that is the user's home folder. It is null if there is no home folder for the user.

getUserName( )

Syntax String getUserName( )

Description Retrieves the current user's login name. The Login module calls getUserName( ) to establish a connection to the application server and file system.

Returns A string containing the user name that the application server recognizes.

isEnterprise( )

Syntax boolean isEnterprise( )

Description Evaluates whether the user connects to an Encyclopedia volume. The Login module calls isEnterprise( ) to determine whether to use a repository on the file system.

Returns False.

Chapter

10

Customizing Java Component online help

This chapter contains the following topics:

- About Actuate Java Component online help files
- Using a custom help location
- Creating a localized help collection
- Customizing icons and the company logo
- Changing help content

About Actuate Java Component online help files

Actuate provides online help for Java Components using the internet by default. To customize online help for Java Components, extract the documentation from ajc_doc.zip from the Actuate Localization and Online Documentation installation into your decompressed WAR file. Switch the help location for Java Components to local by configuring web.xml. Then, customize the online help as needed before recompressing and deploying the Java Components application.

How to switch the help location for a Java Component

Use the following procedure to switch the help location of a Java Component. Switching the help location is required for any of the customization tasks detailed in this chapter.

1. Extract the contents of the Java Components WAR or EAR file into a temporary directory.
2. Copy ajc_doc.zip from the Actuate Localization and Online Documentation installation media. Extract the contents of ajc_doc.zip into the temporary directory for the Java Component WAR file, which generates the help content in the <temporary directory>\help directory.
3. Using a UTF-8 compliant code editor, open the web.xml configuration file.
4. Navigate to the lines that define DEFAULT_LOCALE, similar to the following code:

```
<param-name>AC_DOC_BASE</param-name>
<param-value>
   http://www.actuate.com/documentation/R11</param-value>
```

5. Change the AC_DOC_BASE value to local, as shown in the following code:

```
<param-name>AC_DOC_BASE</param-name>
<param-value>local</param-value>
```

6. Save web.xml.
7. Recompress your WAR file using the Java jar utility and redeploy it to the application server or servlet engine as an application.
8. Restart the application server or servlet engine that runs Java Component.

Understanding the Java Component help directory structure

The local Java Component help files are grouped into directories under the context root for Java Component. The localized help directory under the context root is the container for the help implementation.

Figure 10-1 illustrates the Java Component help directory structure.

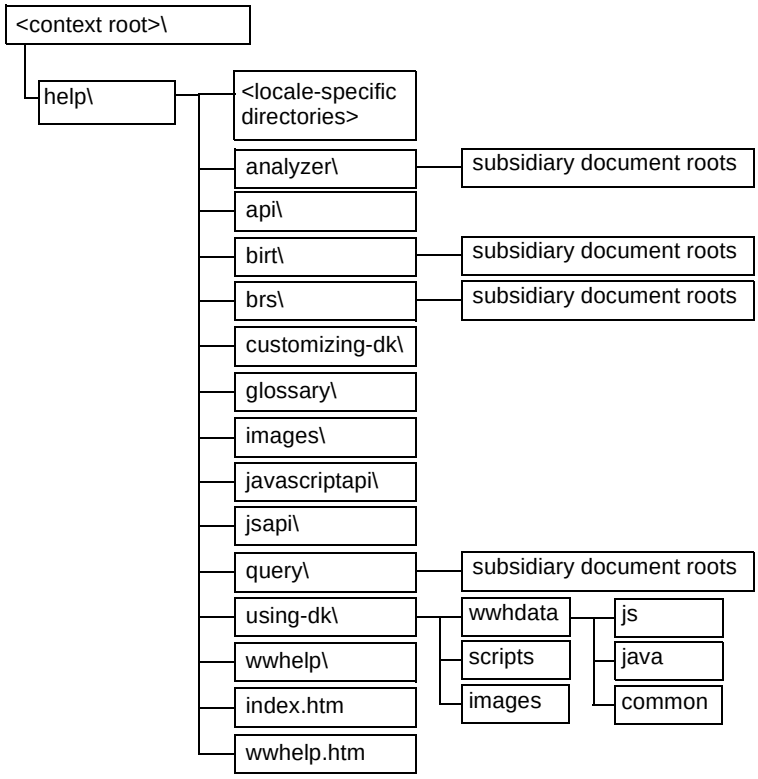

Figure 10-1 Java Component help directory structure

Actuate uses JavaScript (.js) and HTML (.html) files to implement Java Component help. The files that support top-level help styles and images reside in the wwhelp directory. Files that support help content pages and help navigation reside in a document root directory. A document root contains the help files for a specific top-level help topic, such as birt or glossary.

Understanding a help collection

The wwhelp directory contains files that support grouping multiple document roots into a collection. If you open the help using index.htm, the table of contents frame displays the top-level help topics, as shown in Figure 10-2.

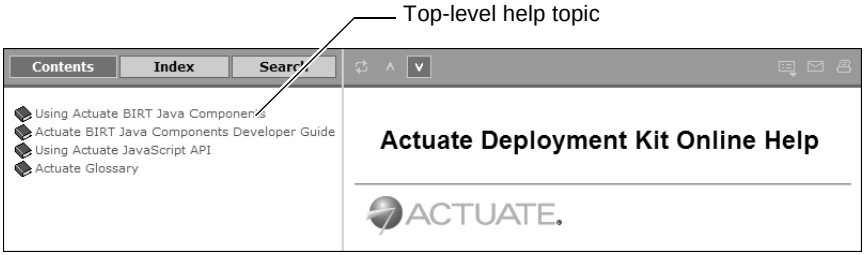

Figure 10-2 Appearance of top-level help topics

A collection has a one-to-one correlation between each top-level help topic and a document root. Each top-level help topic represents a complete book. Table 10-1 lists these applications and the directory containing the corresponding help collection.

Table 10-1 Applications and help collection directories

Application	Directory
Using Actuate BIRT Java Components	using-dk
Actuate BIRT Java Component Developer's Guide	customizing-dk
Using Actuate JavaScript API	javascriptapi
Actuate Glossary	glossary

The help directory contains subdirectories that provide the help collections for applications launched by Java Component. Table 10-2 lists each document root in the Java Component Online help collection and its corresponding top-level help topic.

Table 10-2 Top-level help topics

Help topic	Document root
Actuate BIRT Viewer and Interactive Viewer	birt
Actuate Query	query
BIRT Data Analyzer	analyzer
BIRT Studio	brs

Understanding a document root

The content files for a top-level help topic reside in a corresponding document root. For example, the using-dk document root contains DKusing-intro.2.1.html, DKusing-intro.2.2.html, and so on. These files are the content files for the help. Each document root also contains an index.html file. Opening this file displays the topic and content files for the book.

Within each document root is a wwhdata\common directory that contains the JavaScript files that organize help content and that link the help files to the application. Table 10-3 lists and describes the customizable <document root> \wwhdata\common contents.

Table 10-3 Help content management files

File	Purpose
files.js	Lists the content files to be used and in what order
title.js	Specifies the title for the browser window and the top-level table of contents text
topics.js	Designates the targets for context-sensitive help keys the Java Component emits

Within each document root, a wwhdata\js directory contains JavaScript files that organize the navigation frame. This frame includes the table of contents (TOC), index, and search frames. Table 10-4 lists and describes the customizable <document root>\wwhdata\js contents.

Table 10-4 Help navigation files

File	Purpose
index.js	Organizes the index links and hierarchy
search.js	Designates specific search values and priority
toc.js	Specifies the table of contents frame hierarchy, linking behavior, and text

Understanding context-sensitive help

The Java Component application links to its online help files using wwhelp.html located in <context root>\help. Typically, the links that activate this context-sensitive help are in the Java Component application, as shown in Figure 10-3.

Figure 10-3 Java Component help link for login page

These links in the Java Component emit a URL for the wwhelp.html file and append two parameters to that URL, context and topic. The URL looks like following example:

```
http://host:8080/ajc11/help/wwhelp.htm#context=UserConsole
   &topic=Document_list
```

where

- host is the name of the web server serving your online help.
- 8080 is the port number for the web and http service.
- /ajc11 is the web application's context root,
- /help/wwhelp.htm is the path to the help control file.
- context=UserConsole is the context parameter that specifies the document root for the required help collection. This parameter's value is the context for Java Component help, UserConsole, and directs the request to the Java Component help collection. The context value is determined by the Java Component application.
- topic=Document_list is the topic parameter that locates the required help page. This parameter's value is the topic for viewing and navigating the documents and folders page, Document_list, which is mapped to an anchor in the DKmanaging-reports.3.07.html file. The topic value is determined by the Java Component application.

Understanding locale support

Actuate provides help in US English. The documentation installer places this help in <context root>\help. The installer creates directories for all available locales within <context root>\help. The locale directory names are the locale code of the form <ll_cc> where ll is a language code and cc is a country code. The directory names are all in lower-case letters. Each locale directory contains a wwhelp.htm file and directories for each help collection listed in Table 10-2, as shown in Figure 10-4 for the ac_is locale.

The wwhelp.htm files in each locale directory and its collection directories redirect to the files directly in <context_root>\help. To support localized online help, place localized files in the appropriate locale directory and modify the wwhelp.htm files to not redirect to <context_root>\help.

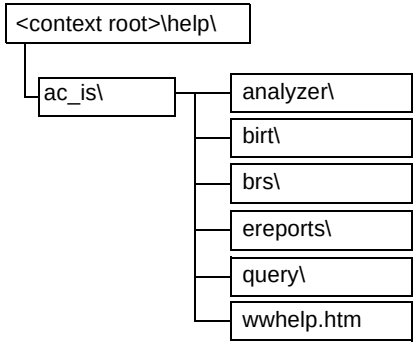

Figure 10-4 ac_is locale directory structure

Using a custom help location

You can use any help system hosted by a web server to provide online help for a Java Component system. To make an external help system available to the Java Component application, the wwhelp.html file must redirect help requests to that external system. Any specific help target can link to any specific page.

To redirect help requests from Java Component to an alternate URL, edit or replace the wwhelp.html file in <context root>\help. You can further specify different targets using the context and topic parameters in the URLs emitted by Java Component in help requests.

Customizing the help location with wwhelp.htm

Use the following procedure to create a wwhelp.htm file that redirects Java Component context-sensitive help requests to another URL.

1. In a text editor, open a new document.
2. Write the required pieces of an HTML file, as shown in the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
<script type="text/javascript" language="JavaScript1.2">
   <!--
...
// -->
  </script>
</head>
<body>
</body>
</html>
```

3 Within the script block, write the javascript method GetParameter to capture URL parameters, as shown in the following code:

```
// get parameters from the URL
method GetParameter( name )
{
   var regexS = "[\\?&]"+name+"=([^&#]*)";
   var regex = new RegExp( regexS );
   var results = regex.exec( window.location.href );
   if( results == null )
       return "";
   else
       return results[1];
}
```

4 As shown in the following code, create a method to perform the following tasks:

- Operate the page.
- Use GetParameter to obtain the topic and context from the URL.
- Open a URL based upon the topic and context.

```
method LaunchHelp()
{
// Get URL parameters
   var context = GetParameter( 'context' );
   var topic = GetParameter( 'topic' );

   var baseURL = "http://myhelpserver/viewer/wwhelp.htm";

   // Begin flow control using context
   switch (context)
   {
   // map the "BIRTIV" context to an outside URL
     case "BIRTIV" :
     self.location = baseURL + "?single=true&context=" +
   context + "&topic=" + topic ;
     break;

   // map the "UserConsole" context to an outside URL
     case "userconsole" :
         baseURL = "http://myhelpserver/ajc11/wwhelp.htm";
     self.location = baseURL + "?single=true&context=" +
   context + "&topic=" + topic ;
     break;

   //the default behavior
    default :
```

```
     self.location = baseURL ;
    }
}
```

The LaunchHelp() method gets the context and topic information from the URL with two calls to GetParameter. The baseURL is set to the myhelpserver application's online help. The flow control switch statements activate specific URLs depending upon the context. Because the myhelpserver application uses the same context and topic variables as standard Java Component help, they are used directly in constructing the URL when activating the self.location methods.

5 Replace the <body> tag with the body tag in the following line:

```
<body onLoad="LaunchHelp();">
```

The onLoad parameter activates LaunchHelp() when the page loads.

6 Save your file as wwhelp.htm in the <context root>\help directory.

7 Test your results by opening Java Component and selecting a help link. The resulting page is from the custom application. For example, the help link on the login page pictured in Figure 10-3 would link to http://myhelpserver /ajc11/help/wwhelp.htm?single=true&context=UserConsole&topic= Document_list.

Creating a localized help collection

Actuate Java Component supports localizing help collections by placing localized help files into the help directory for the appropriate locale. The <context_root>\ help directory contains several locale-specific help directories. For example, the United States English help subdirectory is <context_root>\help\en_us. Other help locale directories can be populated with localized help to provide help for customers in other locales and in other languages. In order to maintain proper help navigation and context-sensitive help links, localized help pages must have the same name as the help pages provided by Actuate.

How to create a localized help collection

Use the following procedure to create a localized online help collection for Java Component that maintains context-sensitive help requests and help navigation.

1 Copy the all of the non-locale-specific directories from <context_root>\help into the appropriate locale-specific directory. For example, for the Italian locale, copy the files into <context_root>\help\it_it.

2 Create localized versions of existing help files in a separate directory.

3 In the locale-specific directory, copy the localized versions of the help files over the English files of the same name. The localized help can be accessed using the following URL:

```
http://localhost:8700/ajc11/help/<locale-specific directory>/
   wwhelp.htm
```

For example, for the Italian locale-specific help, use the following URL:

```
http://localhost:8700/ajc11/help/it_it/wwhelp.htm
```

4 Test your results by opening Java Component, selecting the new locale on the login page, and selecting a help link. The resulting page is from the custom application. For example, the help link on the login page shown in Figure 10-3 would link to http://localhost:8700//ajc11//help/it_it/wwhelp /wwhimpl/common/html/wwhelp.htm#href=using-dk /DKmanaging-reports.3.02.html#229645&single=true.

How to make locale-specific online help the default help

Use the following procedure to make a locale-specific help collection the default help for Java Component.

1 Open wwhelp.htm in the <context root>\help directory in a text editor. Find the following line:

```
setTimeout("location.replace(\"/wwhelp/wwhimpl/common/html/
   switch.htm" + Parameters + "\");", 1);
```

Add the locale-specific directory to the URL string, as shown in the following code:

```
setTimeout("location.replace(\"/<locale-specific directory>/
   wwhelp/wwhimpl/common/html/switch.htm" + Parameters + "\");",
   1);
```

For example, to set the Italian locale as the default locale for context-sensitive help, change the line to the following one:

```
setTimeout("location.replace(\"/it_it/wwhelp/wwhimpl/common/
   html/switch.htm" + Parameters + "\");", 1);
```

2 Save and close wwhelp.htm.

3 Copy the all of the non-locale-specific directories from <context_root>\help into each English locale-specific directory - en_au, en_bz, en_ca, en_gb, en_ie, en_nz, en_us, and en_za. For example, for US English, copy the files into <context_root>\help\en_us.

4 In each English locale-specific directory, open wwhelp.htm in a text editor. Find the following line:

```
setTimeout("location.replace(\"../wwhelp/wwhimpl/common/html/
   switch.htm" + Parameters + "\")
```

Add the locale-specific directory to the URL string, as shown in the following code:

```
setTimeout("location.replace(\"/<locale-specific directory>/
   wwhelp/wwhimpl/common/html/switch.htm" + Parameters + "\");",
   1);
```

For example, to set US English help to the en_us locale for context-sensitive help, change the line to the following one:

```
setTimeout("location.replace(\"/en_us/wwhelp/wwhimpl/common/
   html/switch.htm" + Parameters + "\");", 1);
```

5 Test your results by opening Java Component and selecting a help link. The resulting page is from the custom application. For example, the help link on the login page shown in Figure 10-3 would link to http://localhost:8700 /ajc11/help/it_it/wwhelp/wwhimpl/common/html/wwhelp.htm #href=using-dk/DKmanaging-reports.3.02.html#229645&single=true.

Then, test an English locale by selecting an English locale on the login page and then selecting a help link. The resulting page is from the English locale help. For example, the help link on the login page shown in Figure 10-3 would link to http://localhost:8700/ajc11/help/en_us/wwhelp/wwhimpl /common/html/wwhelp.htm#href=using-dk /DKmanaging-reports.3.02.html#229645&single=true for the US English locale.

Customizing icons and the company logo

The online help pages organize navigation and content into frames, as shown in Figure 10-5.

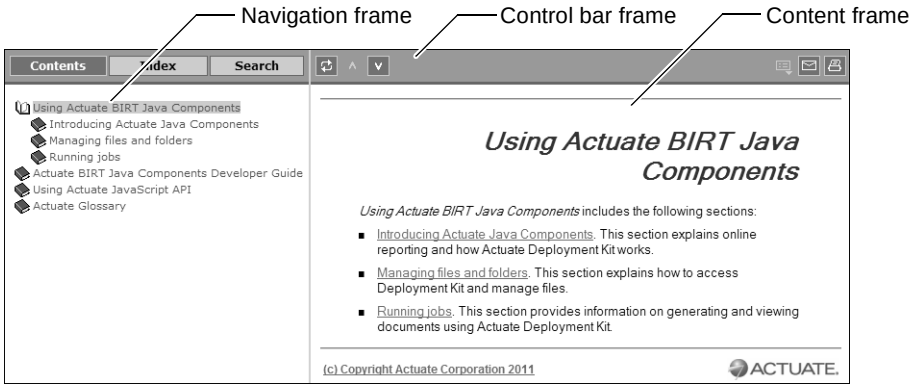

Figure 10-5 Help frames

To change the fonts, colors, and icons of your customized help, you change each frame's content or style file individually.

Changing the corporate logo

The corporate logo is displayed in the content frame based on the image tags in the content pages. Figure 10-2 shows the title page for the help system. This page contains a large logo image. Individual content pages contain a small logo in the footer, as shown in Figure 10-5. To change this logo, change the image tag on every content page.

Changing the corporate logo on the title page

Because this title page is not directly tied to any one document, the title page does not reside in an individual document root. The path of the title page, default.htm, is:

```
<context root>\help\wwhelp\wwhimpl\common\html\default.htm
```

Changing the image tag for the logo in this file changes the logo on the title page.

How to change the logo on the title page

Use the following procedure to change the company logo that is displayed on the help title page in the content frame.

1. Copy your corporate logo image file into the <context root>\help\wwhelp \images directory.
2. In a text editor, open the <context root>\help\wwhelp\wwhimpl\common \html\default.htm file.
3. Locate the following block of code:

```
<!-- Table for floating ActuAteLogo -->
<Table Cols="1" Border="0" cellpadding="0" cellspacing="0"
   height="80%" width="100%">
   <tr>
   <td height="300" valign="bottom" >
   <P align="right">
   <IMG Alt="Actuate Corporation"
   src="../../../images/actuate_logo.gif"><br />
   <br />
   </P>
   </td>
   </tr>
</Table>
```

4. Change actuate_logo.gif to the name of your corporate logo image file.
5. Change the value of the Alt parameter to your company name.

6 Save and close default.htm.

Changing the logo in the help content pages

The footers in the content pages display the Actuate corporate logo by default. To change the corporate logo displayed on a content page, you alter the HTML markup to use a different logo. Actuate uses the corporate logo as a link to the Actuate corporate web application. You can change this link so that the image is a link to your corporate web application.

How to change the corporate logo on a help content page

Use the following procedure to alter the corporate logo and corporate web application link in a content page.

1 Copy your corporate logo image file into the <document root>\images directory for the help topic content you wish to change. For example, to change the logo in the Using Actuate BIRT Java Components help topic, the document root is the <context root>\help\using-dk directory.

2 In a text editor, open the first content page file in the document root that you wish to update. For example, the first content page in the Using Actuate BIRT Java Components documentation is DKusing-intro.2.01.html.

3 Locate the following block of code:

```
<table align="right" border="0" cellspacing="0"
   cellpadding="0">
<tr>
   <td align="right" width="95%" >
      <span style="font-size: 10px ;font-family: Arial,
      Helvetica, sans-serif">
      <a href="http://www.actuate.com">
      <img src="images/actuate_logo_sm.gif" width="115"
      height="22" border="0" alt="Actuate Corporation" /></a>
      <!--
      <a href="mailto:info@actuate.com">info@actuate.com</a>
      -->
      </span>
   </td>
   <td width="5%" />
</tr>
</table>
```

4 Change http://www.actuate.com to the address of your corporate web application.

5 Change actuate_logo_sm.gif to the name of your corporate logo image file.

6 Change the value of the Alt parameter to your company name.

7 Change the width and height attributes to display the logo image properly.

8 Save and close the content file.

9 Repeat steps 2 through 8 for each content file you need to change.

Changing icons

To change the icons for the controls in the navigation frame and the control bar frame, replace the current image files with different ones. The icon images are located in the <context root>\help\wwhelp\wwhimpl\common\images directory. Replacing these image files changes the icons used for the control bar and navigation frames. Table 10-5 lists and the describes the image files for the icons.

Table 10-5 Help content management files

Image	Filename	Purpose	Location
	bkmark.gif	Bookmark the current page.	The control bar frame
	bkmarkx.gif	The bookmark method is not available.	The control bar frame
	doc.gif	Open a single file in the table of contents.	The navigation frame
	email.gif	E-mail a link to the current page.	The control bar frame
	emailx.gif	E-mailing a link is not available.	The control bar frame
	fc.gif	Expand a help topic or sub-topic in the table of contents.	The navigation frame
	fo.gif	Collapse a help topic in the table of contents.	The navigation frame
	frameset.gif	Open the control frame.	The control bar frame
	next.gif	Go to the next page.	The control bar frame
	nextx.gif	There is no next page available.	The control bar frame

Table 10-5 Help content management files

Image	Filename	Purpose	Location
	prev.gif	Go to the previous page.	The control bar frame
	prevx.gif	There is no previous page available.	The control bar frame
	print.gif	Print the current page.	The control bar frame
	printx.gif	Printing is not available for this page.	The control bar frame
	related.gif	View related topics.	The control bar frame
	relatedx.gif	The related topics method is not available.	The control bar frame
	sync.gif	Synchronize the frames so that the control frame matches the content frame.	The control bar frame
	syncx.gif	Synchronizing frames is not available.	The control bar frame

Changing the browser window title

To change the title displayed in the browser's title bar when viewing online help, alter the title.js file for each document root. The browser's title bar appears as shown in Figure 10-6.

Figure 10-6 The browser title bar

How to change the text displayed in the browser's title bar

Use the following procedure to change the text displayed in the browser's title bar when you access help.

1 Navigate to the <document root>\wwhdata\common directory for the help topic you want to customize. For example, to change the text displayed in the

browser title bar when you open the Using Actuate BIRT Java Components help topic, the <document root> is the <context root>\using-dk directory.

2. In a text editor, open title.js.
3. Locate the line in the code that uses the return method. For the Using Actuate BIRT Java Components help topic, it is the following line:

```
return "Using Actuate BIRT Java Components";
```

4. Change the quoted text value to the text you need to display in the browser title bar.
5. Save and close title.js.

Changing help content

Every piece of content in the Actuate Java Component help system is customizable. The possible content changes fall into the following general categories:

- Changing existing help content
- Adding or removing help topics
- Adding and removing content files
- Changing the table of contents
- Changing the index

Changing existing help content

You can modify any of the existing HTML pages of the Java Component help for any help topic to change the information they contain. These HTML files contain specific <a> tags used for internal navigation and context-sensitive help. In general these tags must remain unchanged to maintain context-sensitive help and internal navigation functionality. Table 10-6 lists the tags and their use.

Table 10-6 Help content reserved tags

Tag examples	Purpose
<a name="147349">	An anchor for a specific place in a file. This tag is used by internal links and context-sensitive links.

Table 10-6 Help content reserved tags

Tag examples	Purpose
<a href="javascript: WWHClickedPopup ('UserConsole', 'DKgenerating-reports.4.02.html #147349');"	Internal link. This tag is an internal link to an anchor. In this example: UserConsole is the context, a reserved help topic label. DKgenerating-reports.4.02.html is the file that the link opens. #147349 is the text of the anchor tag that the link accesses.

How to modify the content of existing pages

Use the following procedure to change the help content.

1. Navigate to the document root directory for the help topic you want to change. For example, to change the content of a page in the Using Actuate BIRT Java Components help topic, the document root is the <context root> \using-dk directory.
2. In a text editor, open the content page you need to change. For example, to change the content of the Chapter 2 Managing folders and files page, open the DKmanaging-reports.3.01.html file.
3. Modify the text, being careful not remove any <a> tags that provide internal links and context-sensitive links.
4. Save and close the content file.

Adding or removing help topics

To add or remove help topics from the application help, you delete or create the document root for that help topic. To prevent the navigation pane controls from generating erroneous links to that help topic, you must also alter the help book list, books.js, located in the <context root>\help\wwhelp\wwhimpl\common \private directory. The books.js file also controls the order in which the help topics appear in the table of contents.

How to remove a help topic from the Java Component help system

Use the following procedure to remove a help topic from the Java Component help system.

1. Navigate to the <context root>\help\wwhelp\wwhimpl\common\private\ directory.
2. In a text editor, open the books.js file.

3 Find the following code:

```
function  WWHBookGroups_Books(ParamTop)
{
   ParamTop.fAddDirectory("using-dk", null, null, null, null);
   ParamTop.fAddDirectory("customizing-dk", null, null, null,
   null);
   ParamTop.fAddDirectory("javascriptapi", null, null, null,
   null);
   ParamTop.fAddDirectory("glossary", null, null, null, null);
}
```

4 Delete the line that adds the directory for the topic that you need to remove.

5 Save and close the books.js file.

6 In the file system, delete the document root for the topic that you removed in step 4.

Adding and removing content files

Individual content files are added or removed from the document root for each top-level help topic. To make the content file available for linking and viewing from the help system, you must also alter the file list, files.js, located at <document root>\wwhdata\common. The files.js file also controls the order of the files in the array for reference by other files. For example, the content of files.js for the using-dk document root looks like the following code:

```
function  WWHBookData_Files(P)
{
P.fA("Using Actuate BIRT Java Components","about-dkreports.html");
P.fA("Introducing Actuate Java Component","DKusing-
   intro.2.01.html");
P.fA("Using Actuate Java Components","DKusing-intro.2.02.html");
P.fA("About Actuate Deployment Kits","DKusing-intro.2.03.html");
…}
```

This code establishes the following structure:

- Each file, about-dkreports.html, DKusing-intro.2.01.html, DKusing-intro.2.02.html, and DKusing-intro.2.03.html, is available for linking and display by Java Component help.
- The first file in the array is about-dkreports.html, which is referenced by the array number 0. The second file in the array is DKusing-intro.2.01.html and is referenced by the array number 1 and so on.

The order of the files in the array always begins with and proceeds from 0. The file array is an internal mechanism that supports referencing these files by number within the help topic.

How to add a content file to the Java Component help system

Use the following procedure to add a content file to the Java Component help system.

1. Copy your content file into the document root directory for the help topic you need to enhance. For example, to add a new file to the Using Actuate BIRT Java Components help topic, the document root is the <context root>\using-dk directory.
2. Navigate to the <document root>\wwhdata\common directory.
3. In a text editor, open the files.js file.
4. Find the following code:

```
function  WWHBookData_Files(P)
{
P.fA("Using Actuate BIRT Java Components","about-
   dkreports.html");
P.fA("Introducing Actuate Java Components","DKusing-
   intro.2.01.html");
P.fA("Using Actuate Java Components","DKusing-
   intro.2.02.html");
```

5. Add a P.fA(…); entry for the file to add, placing it where you need it to appear in the file array relative to the other entries.

   P.fA(…); requires two parameters. The first is a string that describes the file. The second is the name of the file. Both parameter values must individually be within quotation marks and separated by a comma.
6. Change the parameter values for the other P.fA(…) calls as needed.
7. Save and close files.js.

Changing the table of contents

Help topics are established in the table of contents by the title.js file in the <document root>\wwhdata\js\ directory for each help topic. For example, the title.js file for the using-dk document root looks like the following code:

```
method  WWHBookData_Title()
{
  return "Using Actuate BIRT Java Components";
}
```

This code indicates that the table of contents text for this help topic is Using Actuate BIRT Java Components. Figure 10-7 shows the hierarchy produced by the code above.

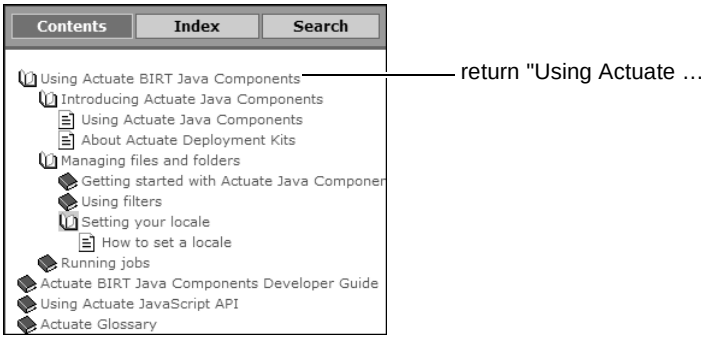

Figure 10-7 The help topic entry in the table of contents

The table of contents displays nested help topics as listed in the toc.js file located in the <document root>\wwhdata\js directory. The toc.js file also controls the following items:

- The table of contents hierarchy
- The text that appears in the table of contents
- The file that opens when a user selects a table of contents entry

For example, part of table of contents entry for the Using Actuate BIRT Java Components chapter in the toc.js file for the using-dk document root looks like the following code:

```
var A=P.fN("Introducing Actuate Java Components","1");
var B=A.fN("Using Actuate Java Components","2");
B=A.fN("About Actuate Deployment Kits","3");
A=P.fN("Managing files and folders","4");
B=A.fN("Getting started with Actuate Java Components","5");
var C=B.fN("Navigating BIRT Deployment Kit","6");
C=B.fN("About the banner","7");
C=B.fN("About the side menu","8");
...
var D=C.fN("How to delete a file","15");
B=A.fN("Using filters","16");
C=B.fN("Enabling the filter option","17");
...
B=A.fN("Setting your locale","21");
C=B.fN("How to set a locale","21#661087");
```

This code establishes the following structure:

- The top-level entry, A, is file "1". File 1 is in position 1 of the internal file array established by files.js. For example, in the using-dk document root, this file is DKusing.

- Entries are created to reside in the next level under the top-level entry using the variable B. Entries in the third level of the table of contents are created using the variable C, and in the fourth level using the variable D. The entries link to file or anchors within a file referenced by the internal file array number. For example, "21#661087' links to the <a name="661087"> anchor in file "21" of the file array, DKmanaging-reports.3.18.html.
- The text that appears in the table of contents for each entry is explicitly defined. For example, the text for the top-level entry is "Using Actuate BIRT Java Components".

Figure 10-8 shows the hierarchy produced by this code.

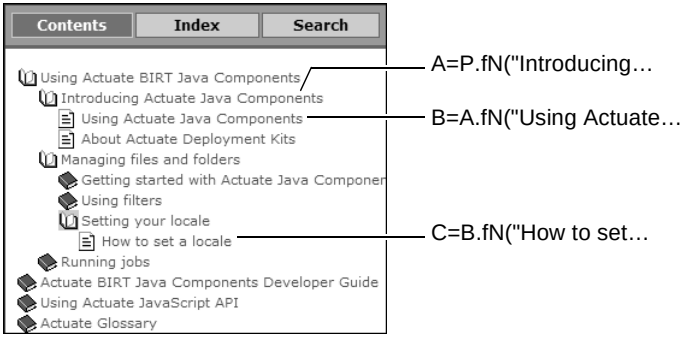

Figure 10-8 The table of contents hierarchy for using-dk

How to add a content file link to the table of contents hierarchy

Use the following procedure to add a content file link to the table of contents hierarchy for the Java Component help system.

1. If you are linking to an anchor, navigate to the document root directory. Open the content file that contains the anchor to which the table of contents will link. Determine the value of the name attribute for the anchor. Then, close the content file without saving it.
2. Navigate to the <document root>\wwhdata\common directory.
3. In a text editor, open the files.js file and determine the internal file array number for the content file, either that you opened in step 1 or that you are linking to directly. Close files.js without saving it.
4. Navigate to the <document root>\wwhdata\js directory.
5. In a text editor, open the toc.js file.
6. Add an entry to toc.js for the table of contents entry using the following format:

```
var B=A.fN("About business reporting using Actuate
   products","1#147349");
```

- var is a keyword that must precede the entry if B has not been defined as a variable in this file prior to this line. Do not use var if B has already been defined.
- B is the table of contents hierarchy level of the new table of contents entry.
- A is the table of contents hierarchy level above the level of the new table of contents entry.
- "About business reporting using Actuate Products" is the string to display in the table of contents for this entry.
- 1 is the array number of the target file established in step 3.
- #147349 is a number sign (#) followed by the value of the name attribute for the anchor established in step 1, if it is applicable. Do not append any additional characters to the array number of the target file if you are just linking to the file and not to a marker.

7 Save and close toc.js.

Changing the index

The index displays keywords for help topics from individual content files. The index.js file located in the <document root>\wwhdata\js directory contains the index entries. The index.js file controls the following items:

- The index hierarchy
- The text that makes up the index entries
- The content to which the index entries link

For example, in the using-dk document root, the index entry for QBE expressions, starting at the letter Q, looks like the following code:

```
A=P.fA("Q",null,null,"002");
B=A.fA("QBE expressions");
C=B.fA("creating",new Array("31#481958","32#482151","33#482228"));
C=B.fA("defining ad hoc parameters for",new
   Array("31#481476","32#482106"));
C=B.fA("entering literal characters in",new
   Array("33#482364","34#481292"));
C=B.fA("formatting date values and",new Array("33#482248"));
C=B.fA("matching string values and",new
   Array("33#482288","33#482372","34#440077","34#481319"));
C=B.fA("retrieving blank characters and",new Array("34#481319"));
C=B.fA("retrieving null values and",new Array("33#482238"));
B=A.fA("query operators",new Array("31#481958"));
```

This code establishes the following structure:

- The top-level entry, A=P.fA, is the label "Q". This entry links to the "002" frame, which is the navigation frame.
- The first entry below "Q" is the "QBE expressions" entry. This entry is one level down in the hierarchy, B=A.fA, of the index for "Q". "QBE Expressions" is merely a label and does not link to anything.
- On the next level down in the hierarchy, C=B.fA, has seven entries, one for each of the sub-topics of QBE Expressions. Each entry has a label and an array of links to topics that the user can choose.

Figure 10-9 shows the hierarchy produced by this code.

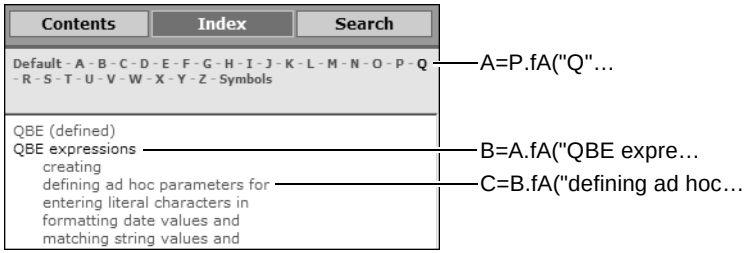

Figure 10-9 The index hierarchy for using-dk

How to add a marker link to the index hierarchy

Use the following procedure to add a marker link to the index hierarchy of the Java Component help system.

1. Navigate to the document root directory. Open the content file that contains the anchor to which the index entry will link. Determine the value of the name attribute for the anchor. Then, close the content file without saving it.
2. Navigate to the <document root>\wwhdata\common directory.
3. In a text editor, open the files.js file and determine the internal file array number for the content file that you opened in step 1. Close files.js without saving it.
4. Navigate to the <document root>\wwhdata\js directory.
5. In a text editor, open the index.js file.
6. Add an entry to index.js for the index entry and anchor link using the following format:

   ```
   var B=A.fA("QBE expressions",new Array("3#394929"));
   ```

   - var is a keyword that must precede the entry if B has not been defined as a variable in this file prior to this line. Do not use var if B has already been defined.
   - B is the index hierarchy level of the new index entry.

- A is the index hierarchy level above the level of the new index entry.
- "QBE expressions" is the string to display in the index for this entry.
- 3 is the array number of the target file established in step 3.
- #394929 is a number sign (#) followed by the value of the name attribute for the anchor established in step 1.

To link the index entry to more than one marker, add each marker link to the list within the new Array parameters. Enclose each anchor reference in quotation marks. Delimit the anchor references with commas, shown in the following example:

```
var B=A.fA("QBE expressions",
   new Array("3#394929","3#394380","3#394677"));
```

7 Save and close index.js.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X