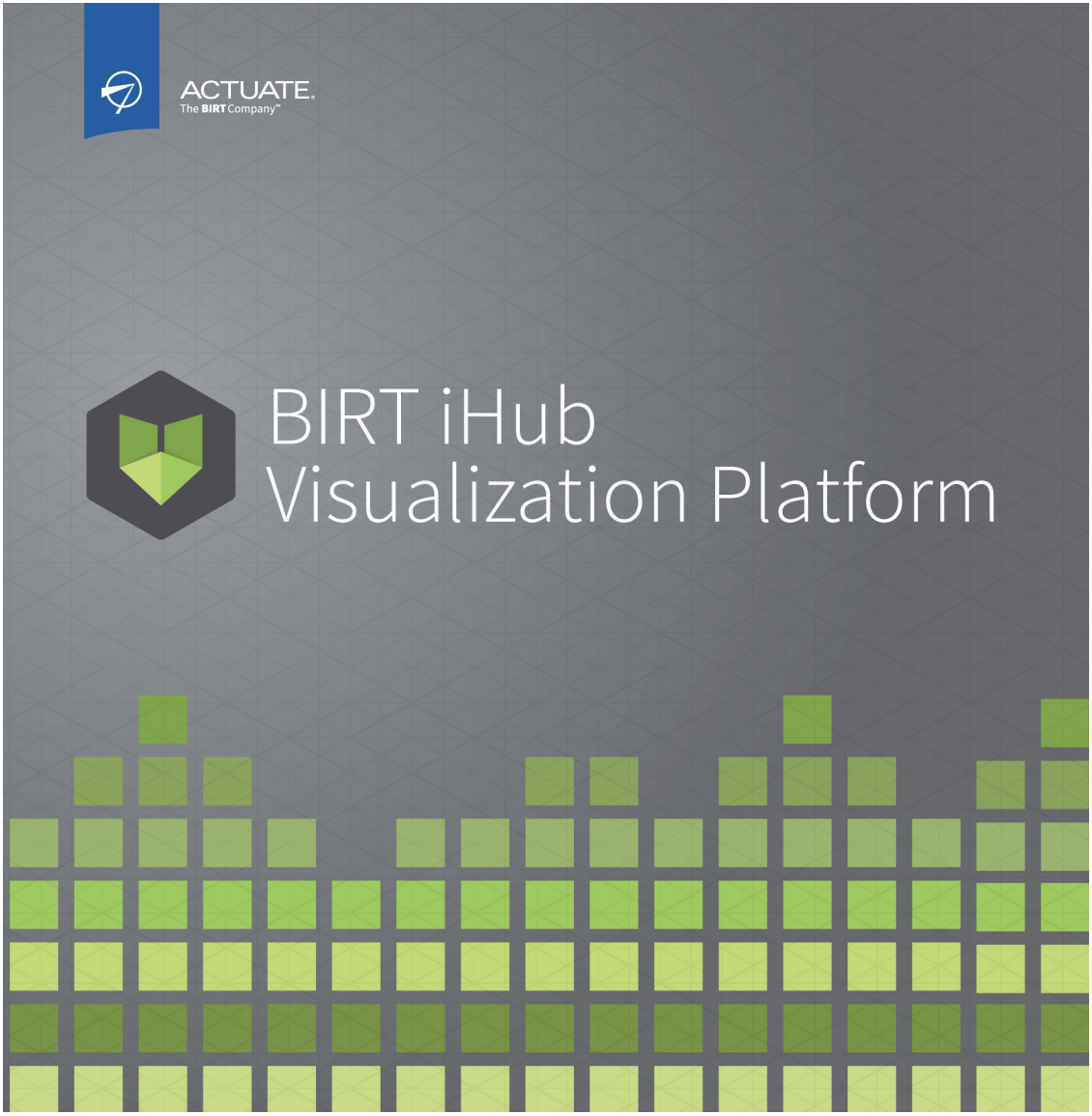


Actuate BIRT Java Components Developer Guide

Document No. 141215-2-771302 October 15, 2014

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 Components.

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 Components and how to customize its appearance and layout.
- *Chapter 1. Introducing Actuate Java Components.* This chapter introduces Actuate Java Components 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 Components web application.* This chapter explains how to work with Java Components JSP files to design custom reporting web applications.
- *Part 2. Actuate Java Component Reference.* This part describes the code components that make up Java Components, such as URIs, JavaScript files, servlets, tags, beans, and security facilities.
- *Chapter 4. Actuate Java Components configuration.* This chapter describes the Java Components configuration files and how to use them.
- *Chapter 5. Configuring BIRT Viewer.* This chapter describes how to configure and extend the BIRT Viewer and Interactive Viewer Java Components.

- *Chapter 6. Configuring BIRT Studio.* This chapter describes how to configure the BIRT Studio Java Components.
- *Chapter 7. Actuate Java Components URIs.* This chapter describes the Java Components JSPs and URL parameters.
- *Chapter 8. Actuate Java Components JavaBeans.* This chapter lists the Java Components JavaBeans.
- *Chapter 9. Using Actuate Java Components security.* This chapter introduces the iPortal Security Extension (IPSE) and explains how to use it.
- *Part 3. Using Actuate JavaScript API.* This part describes the JavaScript API, a JavaScript extension library for Actuate Java Components that allows you to build custom web content and dynamic event handlers in a web page or BIRT application.
- *Chapter 10. Creating a custom web page using the Actuate JavaScript API.* This chapter introduces the Actuate JavaScript API (JSAPI) and explains how to use it in a web page.
- *Chapter 11. Creating dynamic report content using the Actuate JavaScript API.* This chapter describes how to use the JSAPI to enable event handlers, interactive charts, and User Interface controls from within a BIRT Report Design.
- *Chapter 12. Working with Interactive Crosstabs.* This chapter describes how to create, access, view, and modify Interactive Crosstabs.
- *Chapter 13. Actuate JavaScript API classes.* This chapter lists all the standard Actuate JavaScript API classes and their methods.
- *Chapter 14. BIRT Interactive Crosstabs API classes.* This chapter lists all the cross tab classes and their methods.

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 Components architecture

About Actuate Java Components

Actuate Java Components are a collection of related web applications that support accessing and working with report information using a web browser. Web developers and designers use Actuate Java Components' industry-standard technology to design custom e.reporting web applications to meet business information delivery requirements.

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

When deployed, the context root is the 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 Components is the root directory of the web archive (.war) file when it is extracted.

Actuate Java Components 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 Components you have purchased, you must move the license file received from Actuate into the <context root>\WEB-INF directory of the Java Components web archive (.war) file.

How to license Java Components

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

2. Create a temporary directory, such as C:\Temp\ajc on a Microsoft Windows server or /temp/ajc on a UNIX server. If you use an existing directory, ensure that this directory is empty.
3. Extract the contents of the Java Components 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 Components WAR file:

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

     The Java Components 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/ajc
     cp /dev/dsk/cd/ActuateJavaComponent.war  .
     jar -xf ActuateJavaComponent.war
     ```

     The Actuate Java Components 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 Components 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 Components

To deploy a report to the web, you need:

- An Actuate Java Components 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 Actuate Java Components for this directory.

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

Customizing Java components for installation

When you deploy Java Components on an application server, you can use customized Java Components applications. To do this, you need to extract the contents of the Actuate Java Components WAR or EAR file and customize the files directly. To deploy the customized application, recreate a WAR or EAR file from the customized files 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 Components are deployed, you cannot further customize styles, add pages, or make any other changes that affect the Actuate Java Components file structure without extracting the contents of the WAR or EAR file, modifying the contents, and re-deploying it.

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

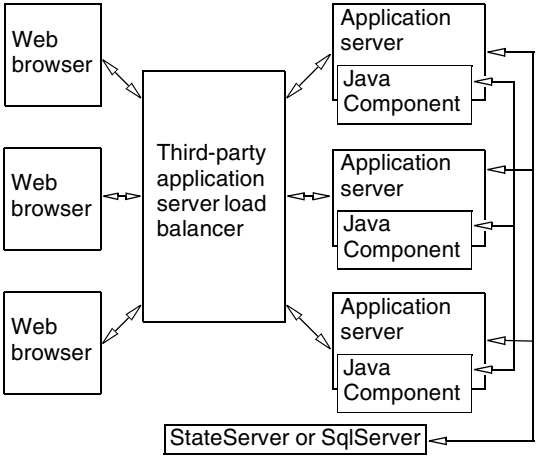


Figure 1-1 Load-balancing architecture for Java Components

About using a cluster of application servers

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

How to customize and deploy Actuate Java Components

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

1. Extract the contents of the Actuate Java Components WAR file into a temporary directory.
2. Customize the Actuate Java Components JavaScript, styles, 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 Components architecture

This section describes the general operation, authentication, and structure of Java Components as parts of a combined web application. The Actuate Java Components architecture is illustrated in Figure 1-2.

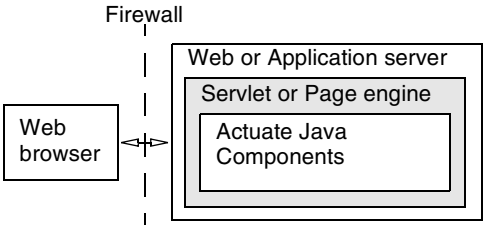


Figure 1-2 Actuate Java Components architecture overview

A user submits a request by choosing a link that specifies an Actuate Java Components URI. As shown in Figure 1-2, the web or application server passes the URI to the servlet or page engine, which invokes the appropriate Java Components application 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 Components manage 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 Components

When setting up a proxy server with Actuate Java Components, 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 Components can perform the redirection.

To redirect a page without using a reverse proxy, Actuate Java Components forward the URL to redirect to the processRedirect.jsp page and update 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.AcSsoLoginAction"
    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="/dashboard/jsp/myfiles.jsp"
     redirect="true" />
    <forward name="dashboard" path="/dashboard" redirect="true" />
    <forward name="ajcLogin" path="/ajclanding.jsp"
     redirect="true" />
    <forward name="landing" path="/login.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="dashboard" path="/dashboard" redirect="true" />
   <forward name="ajcLogin" path="/ajclanding.jsp"
      redirect="true" />
   ]<forward name="landing" path="/login.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 Components pages

Actuate Java Components use 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, the application logic for most Java Components 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 Components, 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 Components, 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 Components URIs

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

About Actuate Java Components URIs

Actuate Java Components URIs consist of the context root and port of the web server where you install and deploy the JSPs or servlets. Actuate Java Components 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 Components 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 Components accepts 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 Components.
- 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 Components security."

Using a special character in a URI

Actuate Java Components 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

(continues)

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

Character	Encoded substitution
equal (=)	%3d
exclamation (!)	%21
greater than (>)	%3e
less than (<)	%3c
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 Components 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 Components <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 Components communicate using 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 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.

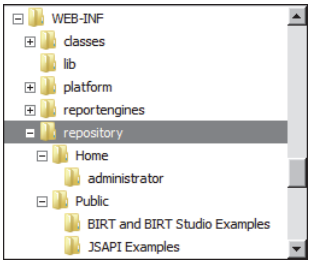


Figure 2-1 Actuate Java Component folder structure

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 Actuate 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 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._23.0.0.v20131216
```

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

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

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

This procedure uses com.actuate.birt.model.defaultsecurity_23.0.0_rsa as an example of a custom security emitter.

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

```
$InstallationDirectory/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_23.0.0_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 2-3 shows all supported modes.

Table 2-3 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 2-4. All padding settings are applicable to all algorithms.

Table 2-4 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…

#By default private key is set to following
symmetric-key=527c23…

#By default public key is not set
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: 23.0.0.v20131216
Require-Bundle: org.eclipse.birt.report.model,
 org.eclipse.core.runtime,
 org.eclipse.birt.core;bundle-version="3.7.0"
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
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

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 Designer environment. When an encryption plug-in sets the value of this attribute to true, the BIRT 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 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:

      ```
      <installation directory>/BRDPro/eclipse/plugins
         /com.actuate.birt.model.defaultsecurity_23.0.0.<version>
      ```

   2. Paste the copied folder in the same folder:

      ```
      <installation directory>/BRDPro/eclipse/plugins
      ```

3 Rename:

```
<installation directory>/BRDPro/eclipse/plugins/Copy of
   com.actuate.birt.model.defaultsecurity_23.0.0.<version>
```

to a new name, such as:

```
<installation directory>/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_23.0.0_rsa
```

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

1 Open:

```
<installation directory>/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_23.0.0_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
Bundle-Name: Actuate Default Security Plug-in
Bundle-SymbolicName: com.actuate.birt.model.
     defaultsecurity.rsa;singleton:=true
Bundle-Version: 23.0.0.<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:

```
<installation directory>/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_23.0.0_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:

```
<installation directory>/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_23.0.0.<version>/
   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:

```
<installation directory>/BRDPro/eclipse/plugins
   /com.actuate.birt.model.defaultsecurity_23.0.0_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
#private key value of RSA algorithm: 308202760201003....
#public key value of RSA algorithm: 30819f300d0....
#The default private key is set to the following
symmetric-key=308202760....
#The default public key is set to the following
public-key=30819f300d0.....
```

5. Save and close encryption.properties.

6. To test the new default RSA encryption, open a BIRT 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 2-5.

Table 2-5 Key generation classes and parameters

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

Table 2-5 Key generation classes and parameters

Class name	Encryption algorithm parameter
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 <installation directory>\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_23.0.0.<version>
```

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 <installation directory>\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_23.0.0.<version>
```

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 Designer
- Actuate BIRT Designer Professional
- Actuate Java Components:
  - Actuate BIRT Viewer
  - Actuate BIRT Interactive Viewer
  - Actuate BIRT Studio
  - 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 Designer and BIRT 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 CSV and the JAR file name is org.eclipse.birt.report.engine.emitter.csv.jar.

How to deploy and use a custom emitter in Actuate BIRT Designer

1 Copy the emitter to:

```
<BIRT Designer installation directory>\eclipse\plugins
```

2 Reopen the designer.

3 Open a report design and choose Run➛View Report. The new CSV format appears in the list of formats, as shown in Figure 2-3.

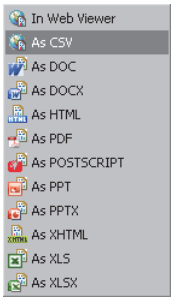


Figure 2-3 List of available formats in BIRT Designer

4 Select the CSV option. A file download dialog box appears as shown on Figure 2-4. Select Save to save the file. The default file name is the report name with the .csv file extension.

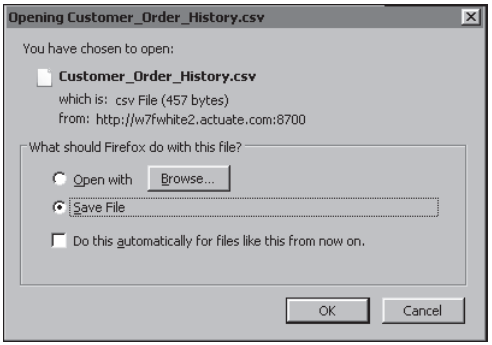


Figure 2-4 Open/Save exported content

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.

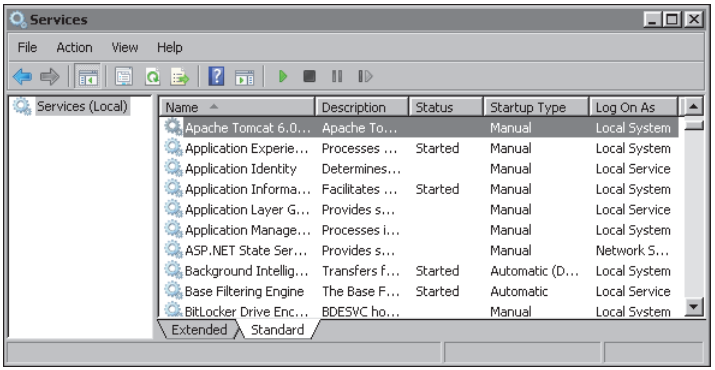


Figure 2-5 Restarting the Apache Tomcat Service

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

4 Select Export Content from the viewer menu.

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

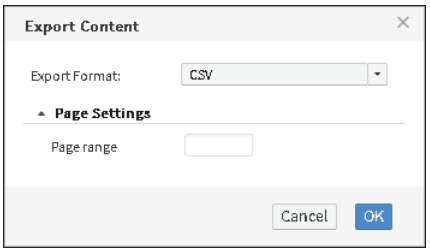


Figure 2-6 Export Content in Actuate BIRT Viewers

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 Components web application

This chapter contains the following topics:

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

Java Components web application structure and contents

Java Components generates web pages using a set of default JSPs then sends the web pages to a web browser. Actuate Java Components 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 Components 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 Components.
- View contains the pages that display data prepared by actions. This component is the presentation layer for Java Components.
- Controller contains the servlets that implement actions. This component is the program control logic for Java Components 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 Components 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 Components 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 the Java Components directory structure

The Java Server Pages (JSPs) that implement Actuate Java Components URIs are grouped by function into directories under the context root. The context root is the web directory in which an Actuate Java Components 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 Components is the root directory of the web archive (.war) file. The Java Components 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 Components directory structure.

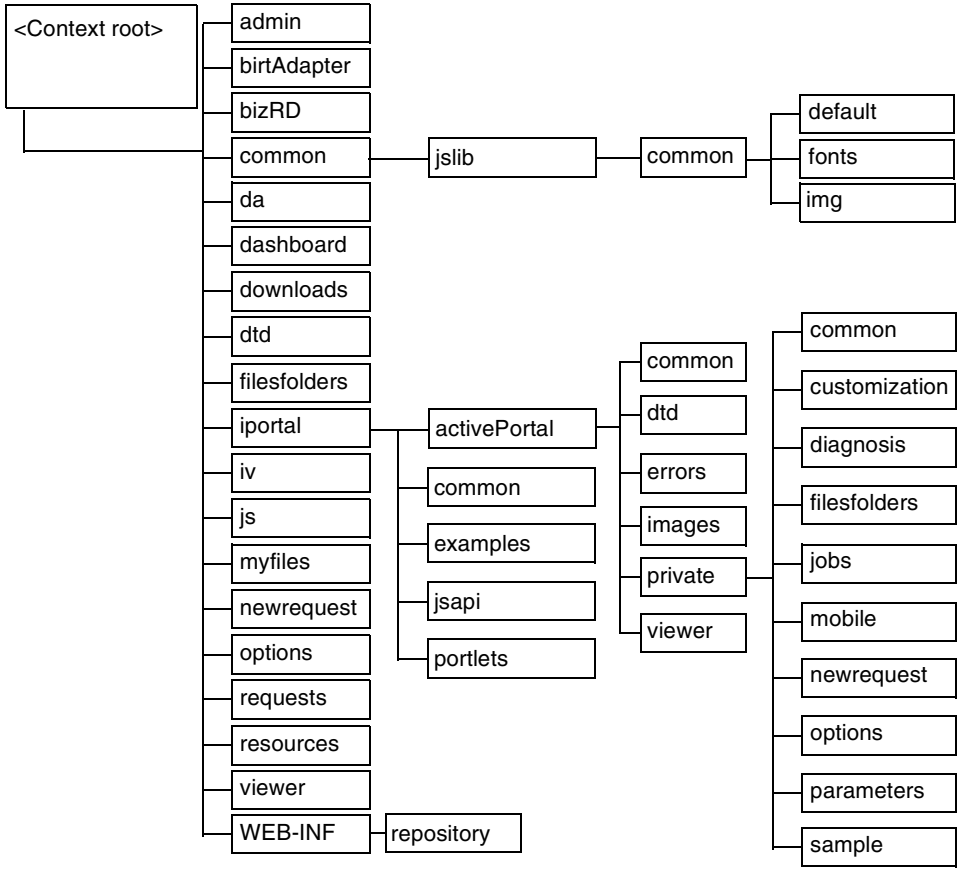


Figure 3-1 Actuate Java Components directory structure

Actuate Java Components 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.
admin	UI files for application administration. Do not change.
birtAdapter	BIRT Viewer integration files.
bizRD	Pages that support BIRT Studio.
common	Common files for the user interface, style, and images.
da	BIRT Data Analyzer support files.
dashboard	Dashboard support files.
downloads	Downloaded files.
dtd	Document type definitions.
filesfolders	Pages that support working with files and folders.
images	Java Components user interface images and icons.
iportal	The Java Components application internals.
iv	Pages that support BIRT Interactive Viewer.
js	JavaScript files that control specific web page elements such as search, toolbar, and table of contents.
META-INF	The Java Components manifest file.
myfiles	Files for listing repository content controlled by myfiles.jsp.
newrequest	Pages that support new requests, such as parameter processing, scheduling, and job status pages.
options	Options-specific pages, such as notification 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.

Table 3-1 <Context root> directories

Directory	Contents
WEB-INF	Files that manage session information such as current user login, roles, and volume.

Table 3-2 lists and describes the directories in the common/jslib/common path.

Table 3-2 <Context root>/common/jslib/common directories

Directory	Contents
default	Style sheets for most pages in Java Components
font	Font definition files.
img	Icon, button, and logo files.

Table 3-3 lists and describes the iportal directories.

Table 3-3 <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
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 3-4 lists and describes the <context root>\iportal\activePortal directories.

Table 3-4 <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 Components application.
common	Common elements included in all reporting web pages, such as banner and side menu elements.
dtd	Document type definitions.

(continues)

Table 3-4 <Context root>/iportal/activePortal directories (continued)

Directory	Contents
errors	Error pages.
images	Images for reporting web pages, such as buttons, icons, lines, and arrows.
private	Most Java Components 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.
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 Components 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 Components application's context root, always use the jar utility to generate the web archive (.war) file after licensing or customization.

Building a custom Java Components context root

An Actuate Java Components web application resides in a context root. You specify the Java Components 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 Components web content to the JSPs in the context root.

You can create several Actuate Java Components 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:8080/marcom
```

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

```
http://MyCorp:8080/marcom/login.do
```

1. Extract the contents of the Java Components 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 Components 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 Components 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 Components 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 8080.

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 Components web application called marcom. The application looks like the default Actuate Java Components 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 Components web application" later in this chapter.

The default Actuate Java Components 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 Components 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 Components. Java Components resolves relative hyperlinks from the context root. For example, in the standard Java Components installation, the following code refers to an images directory at the same level as the Java Components context root directory:

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

All Actuate Java Components 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 Components without making the corresponding change to struts-config.xml. If you do not change the file name consistently in all places, Java Components cannot locate your custom files.

Activating a new web application

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

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 Components web application

Java Components'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 Components web pages, as described in "Customizing a Java Components web application," later in this chapter.

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

Customizing Java Components configuration

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

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

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

How to customize Java Components configuration parameters

Use the following procedure to customize configuration parameters for Java Components. 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 Components.

How to set a default Java Components locale and time zone

The default locale and timezone for Java Componentss 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 Components.

Customizing a Java Components web application

Actuate Java Components 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 Components web application manually:

- Cascading style sheet (.css) files

  CSS files define fonts, colors, and other visual design attributes of a Java Components 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 Components 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 Components Model-View-Controller design. For more information about Struts, access the following URL:

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

- Java

  Java Components uses Java classes to provide functionality. You can create your own Java classes for your custom web application. For more information on the Java Components Java classes, see Chapter 8, "Actuate Java Components 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 Components web pages are defined primarily by JSPs. For more information about the Actuate JavaServer Pages, see Chapter 7, "Actuate Java Components 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 Components 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 Components 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 Components controls web page navigation with Jakarta Struts action paths. An action path is a uniform resource identifier (URI) called directly by Java Components or by a user to access the Java Components 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" />
   <forward name="dashboard" path="/dashboard" redirect="true"/>
</action>
```

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

```
<forward name="success"
   path="/iportal/activePortal/private/filesfolders
   /filefolderlist.jsp" />
```

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

```
<forward name="dashboard"
   path="/iportal/activePortal/mydashboard.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 7, "Actuate Java Components URIs."

To add a new web page to Java Components, 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 Components web pages need access to session information. Your application server and Java Components 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 Components 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 Components 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 Components 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() | "" );
```

Modifying global style elements

Java Components's configuration determines many of its essential methods. Configuring your web application customizes how it operates internally, and affects the user's experience. Perform cosmetic customization tasks using the Java Components skins and style sheets, as described in "Modifying global style elements," later in this chapter.

Set configuration parameters for the Java Components application to tune performance and to control service and application execution. For example, you can perform the following tasks using configuration parameters:

- Setting the default locale
- Modifying global style elements

Configure the Java Components application by changing configuration file contents, such as web.xml. The following section describes the customization procedure using the text editor.

How to customize Java Components configuration parameters

Use the following procedure to customize configuration parameters for Java Components. In this procedure, it is assumed that <context root> \WEB-INF\web.xml is the configuration file.

1. Make a backup copy of web.xml.
2. 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>
   ```

   - <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>
```

3 Save web.xml.

4 Restart the application server or servlet engine that runs Java Components and clear your browser cache.

Setting the default locale

The default locale and time zone 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.

How to set a default Java Components locale and time zone

1 Using a UTF-8 compliant code editor, open the web.xml configuration file.

2 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.

3 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.

4 Save web.xml.

5 Restart the application server or servlet engine that runs Java Components and clear your browser cache.

6 Open the Java Components web application. The login page for the custom application appears.

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 an Java Components web application. To

modify the appearance of the entire Java Components web application, change global style elements.

Global style definitions are located in the <context root>\common\jslib\themes \default\yggdrasil.css file. To change the company logo displayed in the banner, modify the background:url property of the .ac .navbar .actuate class definition. This class definition in yggdrassil.css includes other properties, which is shown in the following code:

```
.ac .navbar .actuate {
  text-indent: -9876px;
  outline: none;
  float: left;
  display: block;
  margin: 12px 0px 0px 0px;
  padding-right: 11px;
  background: url(../img/actuate_logo_navbar.png) no-repeat;
  width: 98px;
  height: 25px;
  border-right: #555 1px solid;
}
```

To change the logo, replace the default value of actuate_logo_navbar.png with a custom logo file name, either as an absolute path, or as the same relative path as the default image by saving the image file in the <context root>\common\jslib \themes\img directory.

To change the background color or height of the top navigation bar, modify the value of the .ac .navbar-inner class definition.This class definition in yggdrassil.css includes other properties, which is shown in the following code:

```
.ac .navbar-inner {
  position: relative;
  height: 43px;
  padding-left: 15px;
  padding-right: 0px;
  background: #363636;
  -webkit-box-shadow: 0px 3px 3px rgba(0, 0, 0, 0.125);
  -moz-box-shadow: 0px 3px 3px rgba(0, 0, 0, 0.125);
  box-shadow: 0px 3px 3px rgba(0, 0, 0, 0.125);
  border-bottom: #222222 1px solid;
  z-index: 100;
  *zoom: 1;
}
```

Replace the default value of the height parameter to change the height of the top navigation bar and change the value of the background parameter to change the color of the top navigation bar.

How to customize the company logo for Java Components

1. Copy your custom logo image file to the following directory:

   ```
   <context root>\common\jslib\themes\img
   ```

2. Open the yggdrassil.css file for editing.

3. Navigate to the following lines:

   ```
   background: url(../img/actuate_logo_navbar.png) no-repeat;
   width: 98px;
   height: 25px;
   ```

4. Change the filename of the background image to the name of custom logo image file and change the width and height dimensions to match the custom image dimensions. For example, if the custom image file was called examplecorp_logo.gif and it had a height of 153 pixels and a width of 37 pixels, you would use the following entry:

   ```
   background: url(../img/ExampleCorp_logo.gif) no-repeat;
   width: 153px;
   height: 37px;
   ```

5. Navigate to the following lines:

   ```
   .ac .navbar-inner {
     position: relative;
     height: 43px;
     padding-left: 15px;
     padding-right: 0px;
     background: #363636;
   ```

6. Change the value of the height and the background to compliment the custom logo. For example, if the custom logo has a yellow background and requires 60 pixels of height with its padding to be centered vertically on the navigation bar, you would use the following entry:

   ```
   .ac .navbar-inner {
     position: relative;
     height: 60px;
     padding-left: 15px;
     padding-right: 0px;
     background: #FFFF00;
   ```

7. Save web.xml. Then, restart the Web Server to apply the changes.

8. Open the Java Components application in a web browser to view the new custom navigation bar. For example, if you followed the above steps for examplecorp_logo.png image, your landing page would appear as shown in Figure 3-2.

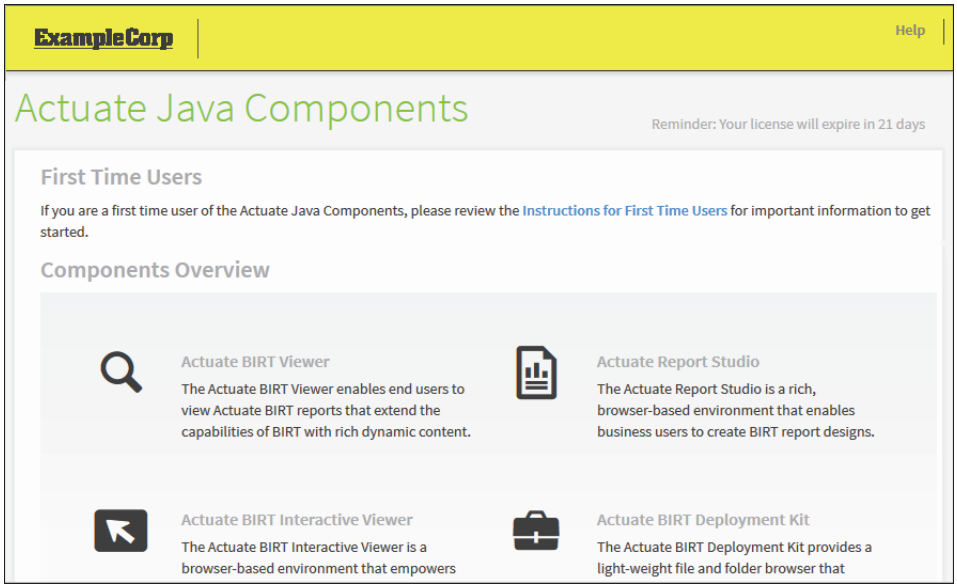


Figure 3-2 Viewing a custom logo and banner in Java Components

Part Two

Actuate Java Component Reference

Chapter

4

Actuate Java Components configuration

This chapter contains the following topics:

- About Actuate Java Components configuration
- Configuring Java Components web applications
- Configuring the Deployment Kit and repository

About Actuate Java Components 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 Components configuration files

File	Features	Description
erni_config.xml	BIRT Studio	Configures BIRT Studio functionality
functionality-level.config	Actuate Java Components	Configures the Actuate Java Components user interface by user groups
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 Components web applications

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

Configuring Java Components using web.xml

Web.xml contains parameters that control Deployment Kit features. Table 4-2 describes the configuration parameters for the Actuate Java Components application.

Table 4-2 Actuate Java Components 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 Components 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 Components 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 Components 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 Components 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 Components do not use cookies. Without cookies, many Java Components 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.
DATAFIELDS_DISPLAY_ ORDER	Controls the display order of Files and Folders. Valid values are: ascending, descending, and none. The default value is ascending.
DEFAULT_LOCALE	Specifies the default locale. Actuate Java Components set 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 Components set 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 Components," 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.

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

Parameter name	Description
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 Components create two error log files: ■ Admin.log records general errors. ■ Soapfault.log records communication errors.
ENABLE_JUL_LOG	Indicates whether to log Actuate Java Components activity. This parameter's default value is true, which enables logging. If you set this parameter to true, Java Components create log files named reportService.<Service number>.<System name>.<Java Components 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 7, "Actuate Java Components 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 time interval, in seconds, that the 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	Reserved. Do not change this setting.

(continues)

JUL_LOG_CONSOLE_ LEVEL	The level of Actuate Java Components 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.

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

Parameter name	Description
JUL_LOG_FILE_COUNT	Specifies the number of log files for a particular time stamp, if the value of ENABLE_JUL_LOG is true.
JUL_LOG_FILE_LEVEL	The level of Actuate Java Components 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 Components activity log file. When a log file reaches this size, Java Components create a new log file and increments its file number. If the log file number reaches the value of JUL_LOG_FILE_COUNT, Java Components reset the file number to 0 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 Components locate the directory in the Java Components home directory. The default value is logs in the Java Components 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 Components use 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 Components 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 Components 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 Components use for temporary files. The default value is 102400, which is 100 MB.
TRANSIENT_STORE_PATH	Path to Actuate Java Components transient files. The default value is set when Java Components are 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 Components transient files. The default value is 40, which is 40 minutes.

Restricting access to Actuate Java Components features using functionality levels

Actuate Java Components provides functionality levels that control which features are available to a user. By default, each user can access all of the functionality level features. To restrict access to features for user groups, the Actuate Java Components administrator can modify functionality levels and add additional levels by editing the configuration file. The standard location for the Actuate Java Components configuration file is <context root>\WEB-INF \functionality-level.config.

When configuring security roles, make sure that any roles specified in the configuration file also exist in the Encyclopedia volume. Because all users automatically belong to the All security role, all users receive the functionality associated with the Basic or the Open functionality level plus the functionality associated with any other roles they have. When restricting access to features, remove the feature from the Open functionality level or comment out the Open

level completely and use the Basic functionality level. Understanding the provided functionality levels.

There are four default functionality levels provided in comments. When the comment tags are removed, the provided functionality levels give the following access.

Users with the Basic level can perform the following tasks:

- Access Documents
- Delete their own files.

Basic level users cannot perform any other modifications.

Users at the Intermediate level have all the Basic level access, and can also perform the following tasks:

- Search documents.
- Upload and download files.
- Use the interactive viewer, if this option is licensed.

Users at the Advanced level have all the Intermediate level access, plus they can perform the following tasks:

- Create and delete folders.
- Share files and folders.

Users at the Administrator level can perform all Advanced level tasks.

Customizing functionality levels

Customize or add functionality levels by modifying or creating a level definition in functionality-level.config. A functionality level definition consists of five parts:

- Level name
  The level name must be a unique alphanumeric string, enclosed within <Name> and </Name> tags.
- Matching security role
  The name of the security role that corresponds to the functionality level. Both the security level and the functionality level must exist before the functionality level can be assigned to a user. Enclose the role name with <Role> and </Role> tags.
- Available features
  Table 4-3 describes the available features.

Table 4-3 Features for functionality levels

Feature	Description
Documents	Provides access to files and folders
Search	Provides access to the file search facility

Features are specified one per line and are enclosed within <FeatureID> and </FeatureID> tags. When a feature is omitted from a functionality level, the corresponding side menu or banner item is hidden to anyone assigned that functionality level. For example, the Search feature is not provided in the Basic functionality level, so the Search link does not appear for users with the Basic functionality level.

- Available subfeatures

  Subfeatures correspond to actions that you can perform through Actuate Java Components. Most subfeatures are associated with a feature. A subfeature cannot be included in a functionality level if its corresponding feature is not included. The subfeatures are described in Table 4-4.

Table 4-4 Subfeatures for functionality levels

Subfeature	Feature	Description
AddFile	Documents	Permits adding files when the user has the appropriate privileges
AdvancedData	NA	Permits the modifying and synchronizing of data sets in BIRT Studio
CreateFolder	Documents	Permits creating folders when the user has the appropriate privileges
DeleteFile	Documents	Permits deleting files when the user has the appropriate privileges
DeleteFolder	Documents	Permits deleting folders when the user has the appropriate privileges
DownloadFile	Documents	Permits downloading files when the user has the appropriate privileges
InteractiveViewing	NA	Permits opening Interactive Viewer
ShareFile	Documents	Permits sharing files when the user has the appropriate privileges

Subfeatures are specified one per line, enclosed within <SubfeatureID> and </SubfeatureID> tags.

The following code shows a sample functionality level entry:

```
<Level>
   <Name>ViewAndSearch</Name>
   <Role>All</Role>
   <FeatureID>Documents</FeatureID>
   <FeatureID>Search</FeatureID>
   <SubfeatureID>ShareFile</SubfeatureID>
   <SubfeatureID>DeleteFile</SubfeatureID>
</Level>
```

The level is named ViewAndSearch and is available to all security roles. Users with ViewAndSearch functionality can run jobs, access documents, and search for files. In addition, they can share and delete their own files.

Preserving functionality levels and features

The functionality-levels.config file is overwritten during upgrade installations. This change ensures that new levels, features, and subfeatures are available to you with your new Actuate Java Components installation. If you have modified your existing functionality-level.config file, make a backup of the changes before the upgrade. Use the backed-up file to access your changes and merge them into the new functionality-level.config file.

Configuring Java Components locale using localemap.xml

Open <context root>\WEB-INF\localemap.xml to see a listing of the available locales in Java Components. 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 Components locales using TimeZones.xml

Open <context root>\WEB-INF\TimeZones.xml to see a listing of the available time zones in Java Components. 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 Deployment Kit and repository

Actuate Java Components provide the ability to organize, run, and view reports in a local repository managed by the Deployment Kit. You configure the security and repository for the Java Component using parameters in web.xml. The Java Components 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 Deployment Kit web.xml parameters

Parameter name	Description
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.

(continues)

Table 4-5 Deployment Kit web.xml parameters (continued)

Parameter name	Description
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.
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 Components files. The default value is set when Java Components are installed.

Chapter

5

Configuring BIRT Viewer

This chapter contains the following topics:

- Configuring the Actuate BIRT Viewer toolbar using iv_config.xml
- Configuring Actuate BIRT Viewer using web.xml
- Configuring default export formats
- Configuring a BIRT Viewer Java Extension

Configuring the Actuate BIRT Viewer toolbar using iv_config.xml

Actuate BIRT Viewer provides a toolbar and context menus that support many of the formatting, sorting, and grouping tasks you perform on data. The toolbar and menus also support adding or deleting columns or groups, and working with fonts and text alignment. You also can print reports and export content or data. Enable or disable each of these features using the configuration file iv_config.xml. Its location is:

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

The iv_config.xml file contains lists of features that are enabled or disabled. The following section of iv_config.xml shows the default feature control for all users

```
<FeatureControl>
   <Role>All/Role>
   <Features>
      <Feature>
         <FeatureName>SaveDesign</FeatureName>
         <Availablity>true</Availablity>
      </Feature>
      <Feature>
         <FeatureName>SaveDocument</FeatureName>
         <Availablity>true</Availablity>
      </Feature>
      …
   </Features>
</FeatureControl>
```

All <FeatureControl> tags are placed within the <FeatureConfiguration> element. Each feature is described by the <Feature> tag, and its availability is described with the <Availability> tag. If a particular feature is available, availability is set to true. If the feature is not available, it is set to false.

Exception stack trace display is also controlled in the iv_config.xml file. When an exception occurs, the viewer can display a stack trace can be displayed in the exception dialog. The trace is used for support purposes, and is set to false by default. This value is not associated with user roles and is placed outside the <FeatureControl> tag. The format of the iv_config.xml file appears as shown in the following code:

```
<IVConfig>
 <!-- flag to sign the report is running in iportal or BRDPro -->
  <RunningMode>Iportal</RunningMode>
 <!-- customize file name generator -->
```

```
  <ExportNameConfig>
   com.actuate.iv.utility.filename.DefaultFileNameGenerator
   </ExportNameConfig>
 <!-- Config features -->
   <FeatureConfiguration>
      <!-- All -->
      <FeatureControl>
         …
      </FeatureControl>
      <FeatureControl>
         …
      </FeatureControl>
         …
   </FeatureConfiguration>
</IVConfig>
```

Table 4-1 lists the features that can be set, and how the availability tag affects them.

Table 4-1 Actuate BIRT Viewer feature set

Feature	Availability tag description
AdvancedSort	Shows or hides the Advanced Sort item in the context menu
Aggregation	Shows or hides the Aggregation item in the context menu
Analyze	Enables or disables the Analyze item in the cross tab toolbar and context menu
AutoEnableIV	Enables or disables interactivity by default
CalculatedColumn	Shows or hides the New Computed Column and Edit Computed Column items in the context menu
ChartSubType	Shows or hides the Chart Subtype item in the context menu
ChartProperty	Shows or hides the Chart Property item in the context menu
ColumnEdit	Shows or hides the Hide Column, Show Column, and Delete Column items in the context menu
CollapseExpand	Shows or hides the Hide Detail and Show Detail items in the context menu
ColumnResize	Shows or hides the Column Width item in the context menu

(continues)

Table 4-1 Actuate BIRT Viewer feature set (continued)

Feature	Availability tag description
ConditionalFormat	Shows or hides the Conditional Formatting item in the context menu
EditReport	Shows or hides the Enable Interactivity item in the toolbar main menu
ExportData	Shows or hides the Export Data menu item in the toolbar main menu
ExportElement	Shows or hides the Export Content menu item in the context menu
ExportElementData	Shows or hides the Export Data menu item in the context menu
ExportReport	Shows or hides the Export Content menu item in the toolbar main menu
FacebookComments	Shows or hides the Facebook Comment menu item in the toolbar main menu
FlashGadgetFormat	Shows or hides the Format Flash Gadget item from the gadget context menu
FlashGadgetType	Shows or hides the Change Type from the gadget context menu
Filter	Shows or hides the Filter item in the context menu
Format	Shows or hides the Format, Change Font, and Alignment items in the context menu
GrandTotal	Enables or disables the Grand Total option in the aggregation dialog
GroupEdit	Shows or hides the Move To Group, Add Group, and Delete Group items in the context menu
HideShowItems	Shows or hides the Hide/Show Item menu item in the toolbar main menu
Highlight	Shows or hides highlighting
HoverHighlight	Shows or hides the mouse over rectangles on page elements
LinkToThisPage	Shows or hides the Link To This Page item in the main menu
MainMenu	Enables or disables the toolbar's main menu
MoveColumn	Shows or hides the Move to Left and Move to Right items in the context menu
PageBreak	Shows or hides the Page Break item in the context menu

Table 4-1 Actuate BIRT Viewer feature set (continued)

Feature	Availability tag description
PageNavigation	Shows or hides the page navigation icons in the navigation bar
Parameter	Shows or hides the Parameter item in the toolbar
Print	Shows or hides the Print menu item in the toolbar main menu
Resize	Enables or disables the Resize feature
ReorderColumns	Shows or hides the Reorder Columns item in the context menu
SaveDesign	Shows or hides the Save Design menu item in the toolbar main menu
SaveDocument	Shows or hides the Save Document menu item in the toolbar main menu
ScrollControl	Shows or hides the scroll control panel in page content
ServerPrint	Shows or hides the Server Print menu item in the toolbar main menu
ShareFormat	Shows or hides the Copy Format menu item in the context menu
ShareStyle	Shows or hides the Copy Style menu item from the context menu
ShowTooltip	Shows or hides tooltips
Sort	Shows or hides the Sort Ascending and Sort Descending items in the context menu
SubTotal	Enables or disables the SubTotal option in the aggregation dialog
SuppressDuplicate	Shows or hides the Repeat Values and Do Not Repeat Values items in the context menu
SwitchView	Shows or hides the Switch View item in the context menu
Toc	Shows or hides the TOC menu item in the toolbar main menu
TextEdit	Shows or hides the edit text icon on text elements
Toolbar	Shows or hides the toolbar
ToolbarHelp	Shows or hides the toolbar Help menu item
TopBottomNFilter	Shows or hides the Top/BottomN item in the context menu

Configuring Actuate BIRT Viewer using web.xml

Actuate BIRT Interactive Viewer (IV) parameters in web.xml affect how BIRT reports are run and viewed. Table 4-2 describes these configuration parameters for BIRT reports, BIRT Viewer, and Interactive Viewer.

Table 4-2 Actuate BIRT Viewer web.xml parameters

Parameter name	Description
ALLOW_EXPORT_PAGE_LIMIT	Indicates the maximum number of pages that can be exported or printed at a time from Actuate BIRT Viewer. For example, if the value of this parameter is 200, no more than 200 pages will be exported or printed from a report using the viewer.
ALLOW_IV_PAGE_LIMIT	Specifies whether Java Components check for a page limit before triggering an operation.
AUTOSUGGEST_DELAY	Configure the delay before the parameters page opens an automatic suggestion for a parameter. The value is measured in milliseconds, and the default value is 500.
AUTOSUGGEST_FETCH_SIZE	The number of autosuggest parameter values to load on the parameters page. The default value is -1, which loads all values.
AUTOSUGGEST_LIST_SIZE	The number of autosuggest parameter values to display on the Parameters page when active. If more values exist than are displayed, the user can scroll through the other values. The default value is 10.
BIRT_ARCHIVE_MEMORY_TOTALSIZE	The total memory available for BIRT report document files, in kilobytes. The default value is 50 megabytes.
BIRT_CHART_CONVERT_TO_IMAGE_TIME_OUT	Sets the time out for conversion from chart to image in a BIRT report. The default value is 6.
BIRT_CHART_MAX_ROW	The maximum number of rows bound to a chart in a BIRT report. The default value is 10000 rows.
BIRT_CHART_MAX_VARIABLE_SIZE	The maximum size for a variable used in a Flash chart, measured in bytes. The default value is 0, which allows a variable to be of any size.
BIRT_CUBE_FETCH_LIMIT_COLUMN_EDGE	The maximum column limit for accessing a data cube. The value must be a non-negative integer; 0 indicates no limit.
BIRT_CUBE_FETCH_LIMIT_ROW_EDGE	The maximum row limit for accessing a data cube. The value must be a non-negative integer; 0 indicates no limit.

Table 4-2 Actuate BIRT Viewer web.xml parameters (continued)

Parameter name	Description
BIRT_DATA_RESULTSET_MAX_BUFFER_SIZE	The result set buffer size, in megabytes, for a data set in a BIRT report. The default value is 10 megabytes.
BIRT_HTMLRENDEROPTION_ENGCASSTYLE	Enables the agentStyleEngine property for the HTML render option for a BIRT report. This setting is related to using a browser's internal CSS capabilities when rendering reports in HTML. It provides better column alignment and faster rendering, especially in a browser other than Microsoft Internet Explorer. The default value is true.
BIRT_JDBC_CONNECTION_POOL_SIZE	Specifies the number of idle connections cached by BIRT JDBC connection pool. The default value is 10.
BIRT_JDBC_CONNECTION_POOL_TIMEOUT	Specifies how long an idle connection will remain in the BIRT JDBC connection pool in seconds. The default value is 3600.
BIRT_LINKED_DATA_MODEL_DATA_MODEL_SIZE	Sets an upper limit on data loaded into memory by a data model at runtime, measured in megabytes. The default value is 0.
BIRT_RESOURCE_PATH	The path to Actuate BIRT shared resources, including libraries and templates for the BIRT report designs and BIRT Studio. The default value is <context root>\resources.
BIRT_SCRIPT_LIB_PATH	Path for the BIRT script libraries (JARs). The default value is <context root>\scriptlib.
BIRT_VIEWER_LOCALE	Locale that determines formatting for numbers and dates on BIRT reports. The default value is the locale of the machine on which Java Components are installed.
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 Components 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.

(continues)

Table 4-2 Actuate BIRT Viewer web.xml parameters (continued)

Parameter name	Description
CACHE_CONTROL *(continued)*	■ NO-STORE indicates that information is cached but not archived. ■ 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.
DEFAULT_LOCALE	The default locale. The default locale is en_US. Users can select a locale when they log in.
DEFAULT_TIMEZONE	The default time zone. The default time zone is Pacific Standard Time (PST).
DISPLAY_ATTRIBUTE_ ITEM	Sets whether to collapse attribute nodes in the dimension tree. False sets nodes to collapse; True sets nodes to expand. The default value is false.
EXPORT_AS_ ATTACHMENT	Determines whether a Microsoft Excel, PowerPoint, or Word report for BIRT Viewer is opened in the Microsoft Internet Explorer browser or a separate application. ■ When the value is true, the exported report opens in a separate Microsoft Word, Microsoft PowerPoint, or Microsoft Excel application. ■ When the value is false, the exported report opens in the browser window with Microsoft Word, Microsoft PowerPoint, or Microsoft Excel embedded inside the browser. The Firefox browser always opens these report formats in a separate application.
IV_ENABLE_IV	Determines whether the Enable Interactivity option is usable in the BIRT Viewer control menu. If false, the Enable Interactivity option is disabled.

Table 4-2 Actuate BIRT Viewer web.xml parameters (continued)

Parameter name	Description
JAVA_REPORT_API_ IMAGE_CACHE_ EXPIRATION	Specifies how long in seconds to cache images for Actuate BIRT reports and business reports. The default value is 86,400, which is one day.
JREM_TASK_QUEUE_SIZE	Specifies the maximum queue length for the Java Report Engine thread pool. The default value is 1000.
JREM_THREAD_POOL_ SIZE	Specifies the maximum number of threads in the Java Report Engine thread pool. The default value is 10.
JREM_THREADPOOL_ MAXSYNC_ TASKRUNTIME	Specifies the maximum time a synchronous report generation is allowed to run. The default value is 600.
JREM_THREADPOOL_ MONITORTHREAD_ POLLINGINTERVAL	Controls the interval in seconds at which the Java Report Engine thread pool checks for Java report execution time-out or queue time-out. The default value is 30.
JREM_THREADPOOL_ SYNC_TASKQUEUE_ TIMEOUT	Specifies the maximum time, in seconds, that a Java synchronous request stays in the Java Report Engine task queue before timing out, in seconds. The default value is 300.
NUMBER_OF_FILTER_ VALUES	Specifies the number of distinct values to display when a user chooses to filter a report on a column in BIRT Viewer. The default value is 200.
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.
PROXY_BASEURL	Indicates a proxy server's URL if the network uses one between the BIRT Viewer web application and the client. The default value is blank, which indicates that the network does not use a proxy server.
REPOSITORY_CACHE_ TIMEOUT_SEC	Specifies, in seconds, how long to retain temporary files that BIRT Viewer creates when a user modifies the appearance of a report. The default value is 900, which is 15 minutes.
TEMP_FOLDER_ LOCATION	Path to the folder where temporary files are created.

Configuring default export formats

You can export a BIRT report to various formats from the BIRT viewer. These formats include doc, docx, pptx, pdf, postscript, ppt, pptx, xls, and xlsx. The Actuate Java Component platform provides a sample .xml file for each format, which you can use to configure the default export options for that format. For example, you can configure the sample .xml file for the XLSX format to set Enable pivot table if it fits on one page to false by default in Export Content, the dialog that appears when you choose to export a BIRT report to XLSX format, as shown in Figure 4-3.

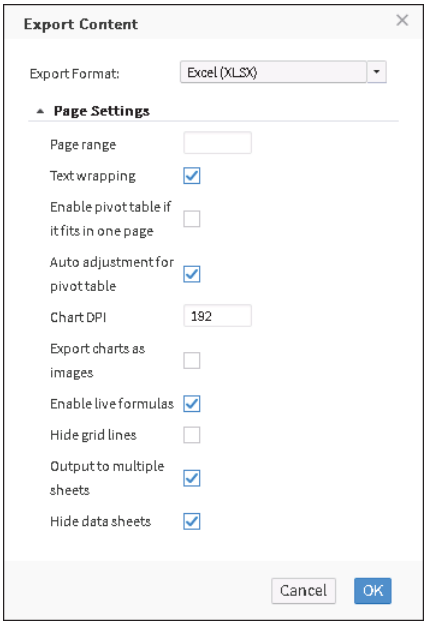


Figure 4-3 Viewing default export property values on Export Content

The location of the folder containing the sample .xml files is:

```
<context root>\WEB-INF\platform\plugins
   \com.actuate.birt.report.engine.emitter.config_<RELEASE_NUMBER>
   .v<date_stamp>
```

Each .xml file has the following name:

```
org.eclipse.birt.report.engine.emitter.<FORMAT_TYPE>.xml
```

For example, Listing 4-8 shows the XML you can modify in org.eclipse.birt.report.engine.emitter.xlsx.xml, to configure the export options for the XLSX output format.

Listing 4-8 Sample .xml file content

```
<RenderOptions>
   <!-- Enables the emitter. Set to "false" will hide this emitter
      in export content dialog. -->
<!--
   <emitter enabled="true"/>
-->
<!--
   <option name="excelRenderOption.wrappingText" default="true"
      enabled="true"/>
   <option name="ChartDpi" default="192" enabled="true"/>
   <option name="excelRenderOption.ExportChartsAsImages"
      default="false" enabled="true"/>
   <option name="excelRenderOption.EnableLiveFormula"
      default="true" enabled="true"/>
   <option name="excelRenderOption.hideGridlines" default="false"
      enabled="true"/>
   <option name="excelRenderOption.multipleSheet" default="true"
      enabled="true"/>
   <option name="excelRenderOption.AutoFitPivotTable"
      default="true" enabled="true"/>
   <option name="excelRenderOption.EnablePivotTable"
      default="true" enabled="true"/>
-->
</RenderOptions>
```

The <RenderOptions> element contains the following child elements:

- <emitter>
  Specifies whether the output format is a selectable option in Export Content—Export Format. Possible values are true or false. If this element is commented out in the .xml file, the value of <emitter> is true.

- <option>
  Represents an export option in Export Content. Contains the following attributes, which support configuring the export option:

  - name
    Name of the export option. Required.

  - default
    Default value of the export option. The value of the export option if the user does not change it or if the export option on Export Content is hidden. Required.

  - enable

Determines whether the export option appears on Export Content. If set to true, the option appears. If set to false, the option is hidden. Optional. The value of <enable> is true if this attribute is omitted in the <option> element.

How to configure default document export options

1. Stop the web or application server.
2. Modify the .xml file for the export format for which you want to configure default output export options. Table 4-3 lists the default settings parameters for each output format.
3. Start the web or application server.

Table 4-3 Configurable default settings for exporting content

Format	Option name	Permissible values
All formats	ChartDpi	Sets the chart resolution. Value can be any integer greater than 0.
AFP	afpRenderOption .pageDPI	Permissible values are 240, 300, 600, 1440.
AFP	xifRenderOption .plexMode	Permissible values are Simplex, Duplex, Tumble.
AFP	afpRenderOption .allowBlackAndWhiteImg	Permissible values are true or false.
AFP	afpRenderOption .allowSingleColorImg	Permissible values are true or false.
AFP	afpRenderOption .allowGrayscaleImg	Permissible values are true or false.
AFP	afpRenderOption .allowFullColorRGBImg	Permissible values are true or false.
AFP	afpRenderOption .allowFullColorCMYKImg	Permissible values are true or false.
DOCX	EmbedHtml	Permissible values are true or false.
PDF, PostScript	pdfRenderOption .bidiProcessing	Enables bidirectional text support. Permissible values are true or false.
PDF, PostScript	pdfRenderOption .textWrapping	Enables text wrapping. Permissible values are true or false.

Table 4-3 Configurable default settings for exporting content (continued)

Format	Option name	Permissible values
PDF	pdfRenderOption .hyphenation	Enables splitting words with a hyphen at line breaks. Permissible values are true or false.
PDF, PostScript	pdfRenderOption .fontSubstitution	Enables font substitution for unknown fonts. Permissible values are true or false.
PDF, PostScript	pdfRenderOption .pageOverflow	Controls rendering content when the content exceeds the page size. Integer values indicate the following options: ■ 1: clips the content ■ 2: scales the content to fit the page ■ 4: (default) divides the content into multiple pages ■ 8: expands the page to fit content
PDF	pdfRenderOption .embeddedFonts	Embeds fonts in the output document. Permissible values are true or false.
PDF	RenderChartInSVG	Renders charts as vector graphics. Permissible values are true or false.
PDF	repaginateForPDF	Permissible values are true or false.
PPT/PPTX	BIDIProcessing	Enables bidirectional text support. Permissible values are true or false..
PPT/PPTX	TextWrapping	Enables text wrapping. Permissible values are true or false.
PPT/PPTX	FontSubstitution	Sets font substitution for unknown fonts. Permissible values are true or false.
XLS/XLSX	excelRenderOption .wrappingText	Enables text wrapping. Permissible values are true or false.
XLS/XLSX	excelRenderOption .EnablePivotTable	Enables pivot tables. Permissible values are true or false.
XLS/XLSX	excelRenderOption .AutoFitPivotTable	Enables BIRT Viewer to automatically adjust content for display in pivot tables. Permissible values are true or false.
XLS/XLSX	excelRenderOption .ExportChartsAsImages	Renders charts as images only. Permissible values are true or false.

(continues)

Table 4-3 Configurable default settings for exporting content (continued)

Format	Option name	Permissible values
XLS/XLSX	excelRenderOption .EnableLiveFormula	Enables formulas for derived values. Permissible values are true or false.
XLS/XLSX	excelRenderOption .hideGridlines	Hides grid lines. Permissible values are true or false.
XLS/XLSX	excelRenderOption .multipleSheet	Enables multiple worksheet output. Permissible values are true or false.

In earlier releases, you configured default export options for a particular output format by creating a RenderDefaults.cfg file, and placing it in the JAR file for that output format emitter. The main advantage of configuring export options using an XML file as described in this section is that you do not need to work with a JAR file.

For backward compatibility, the BIRT Hub 3 release of BIRT Java Components supports configuring default export options using a RenderDefaults.cfg file. BIRT Java Compnents use this file if the .xml file this section describes does not exist. If neither file exists, BIRT Java Components use the default export options specified in the RenderOptions.xml file the format emitter JAR file contains.

Configuring a BIRT Viewer Java Extension

The BIRT Design Engine API provides the IBirtViewerExtension, IBirtViewerContext, IBirtViewerOp, and IBirtViewer Session Java interfaces to extend the functionality of BIRT Viewer. Classes implementing these interfaces are associated with the BIRT Viewer web application when they are added in the birtviewer-extension.xml configuration file. Its location is:

```
<context root>\WEB-INF\birtviewer-extension.xml
```

For example, to enable myIVExtension.jar to operate on BIRT Viewer, add an entry to birtviewer-extension.xml as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<BirtWebViewerExtension>
<!--  Class name for Interactive viewer extension -->
  <InteractiveViewer>com.actuate.sample.MyIvExtension
   </InteractiveViewer>
  </BirtWebViewerExtension>
```

Save the compiled code archive to the <context root>\WEB-INF\lib folder for the BIRT Viewer web application and restart the application to enable the custom interface. The interfaces are part of the com.actuate.birtviewer.extension package of the BIRT Design Engine API.

The IBirtViewerExtension interface defines the event handler methods that activate implemented code, as follows:

```
package com.actuate.birtviewer.extension;

public interface IBirtViewerExtension{

// Event handler that runs when a design changes
void afterDesignChange( IBirtViewerContext context, IBirtViewerOp
   operation, ReportDesignHandle designHandle );

// Event handler after design get opened.
void afterDesignOpen( IBirtViewerContext context,
   ReportDesignHandle designHandle );

// Event handler that runs when a design is saved
void afterDesignSave( IBirtViewerContext context, IBirtViewerOp
   operation, ReportDesignHandle designHandle, String path );

//Event handler that runs when viewer creates a new session object
void afterViewerSessionStart( IBirtViewerContext context );

// Event handler that runs before a design change occurs
void beforeDesignChange( IBirtViewerContext context, IBirtViewerOp
   operation, ReportDesignHandle designHandle );

// Event handler that runs before a design opens
void beforeDesignOpen( IBirtViewerContext context, String path );

// Event handler that runs before a design is saved
boolean beforeDesignSave( IBirtViewerContext context,
   IBirtViewerOp operation, ReportDesignHandle designHandle,
   String path );

// Event handler before soap response get sent back.
void beforeResponse( IBirtViewerContext context, IBirtViewerOp
   operation, GetUpdatedObjectsResponse response );

// Triggered when session object gets destroyed.
void beforeViewerSessionClose( IBirtViewerSession session );
}
```

The IBirtViewerContext interface defines methods that retrieve information from the HTTP session, as follows:

```
package com.actuate.birtviewer.extension;

public interface IBirtViewerContext {

// Gets the base URL for the viewer
String getAppBaseUrl( );

// Gets reportlet id/bookmark if user is working on a reportlet.
String getReportletId( );
```

```
// Gets the extended session object
IBirtViewerSession getSession( );

// Gets the current user name
String getUserName( );

// Gets the volume profile name
String getVolumeProfile( );

// Gets the resource folder name
String getVolumeResourceFolder( );
}
```

The IBirtViewerOp interface defines methods that retrieve information from the extended session for BIRT Viewer, as follows:

```
package com.actuate.birtviewer.extension;

public interface IBirtViewerOp {

// Gets column ids if target element is a table
String[] getColumnIIds( );

// Gets the operation name
String getName( );

// Gets the target element instance ids
String[] getTargetIIds( );

// Get target element type
String getTargetType( );
}
```

The IBirtViewerSession interface defines methods that retrieve and set a session from the extended session for BIRT Viewer, as follows:

```
package com.actuate.birtviewer.extension;

public interface IBirtViewerSession {

// Check whether key exists.
boolean containsKey( String key );

//Get attribute from session.
Object getAttribute( String key );

//Set attribute into session.
void setAttribute( String key, Object attribute );
}
```

Chapter

6

Configuring BIRT Studio

This chapter contains the following topics:

- Enabling or disabling functionality
- Configuring the application environment

Enabling or disabling functionality

BIRT Studio provides a full range of tools to support the report design process. Administrators can limit BIRT Studio functionality by user roles. For example, if the BIRT Studio users you support have little experience designing reports, you can simplify the design process by disabling more advanced functionality, such as creating calculated columns, aggregating data, and joining multiple information objects. Alternatively, if you want users to format report content only by selecting a corporate-designed theme, you can disable the formatting functionality.

Configuring toolbar and context menu items

You configure the toolbar and context menu functionality that is available to users by editing attributes in the BIRT Studio configuration file, erni_config.xml. This file is located in:

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

Customizations you make to erni_config.xml apply at the application level. If you want different sets of functionality available to different groups of users, you need to create multiple instances of the web application, then customize the functionality of each BIRT Studio instance.

In erni_config.xml, the <actionSets> element defines all the user actions that can be enabled or disabled. The actions are organized by category, for example, file operations, calculations, and formatting. The <actionSet> element defines the category, and the <action> element defines a specific action. Listing 4-1 shows the hierarchy of elements.

Listing 4-1 An example of an <actionSet> element in erni_config.xml

```
<actionSets>
   <actionSet>
      <name>FileOperations</name>
      <visible>true</visible>
      <action>
         <name>New</name>
         <enabled>true</enabled>
      </action>
      <action>
         <name>Open</name>
         <enabled>true</enabled>
      </action>
      <action>
         <name>Save</name>
         <enabled>true</enabled>
      </action>
```

```
    <action>
      <name>SaveAs</name>
      <enabled>true</enabled>
    </action>
  </actionSet>
```

By default, all actions are enabled. You can disable actions in the following ways:

- To disable a particular action, change the action's <enabled> attribute from true to false.
- To disable all actions within a category in one step, change the action set's <visible> attribute from true to false.

For the changes to take effect, restart the appropriate Windows service. When you relaunch BIRT Studio, the toolbar displays different buttons and the context menus display different items, depending on which action or actions you disabled.

Configurable actions

Table 4-1 lists toolbar and context menu actions that you can enable or disable. Some of the actions appear in both the toolbar and context menus, and some appear in context menus only. While you can disable any of the actions defined in erni_config.xml, it does not make sense to disable all the actions. For example, disabling both the New and Open actions under file operations prevents a user from creating or opening reports.

Table 4-1 User actions that you can enable or disable through erni_config.xml

Action set	Action	Description
Calculations	Aggregation	Performs a calculation over a specified set of data rows.
	Calculation	Creates a calculated column, based on a specified expression.
	ChangeSubtotal	Changes the subtotal function, applied to a column in a summary table.
	Chart	Inserts a chart.
	DataFields	Shows the data fields in the report, and supports adding or deleting fields in the report.
	EditCalculation	Changes a calculated column.
	Filter	Filters table rows, based on a specified condition.

(continues)

Table 4-1 User actions that you can enable or disable through erni_config.xml (continued)

Action set	Action	Description
ColumnHeader Operations	DeleteRow	Deletes the row of the selected column header.
	InsertRow	Inserts a row above or below the selected column header.
	Merge	Merges the selected column header with the header on the right, left, or above.
	Split	Splits the selected merged columns.
ColumnOperations	ColumnWidth	Changes the width of the selected column.
	HideColumn	Hides the selected column.
	MergeColumns	Merges the selected columns.
	MoveToDetail	Moves the selected item in the group header row to the table's detail row.
	MoveToGroup	Moves the selected item in the table's detail row to the group header row.
	NoRepeat	If duplicate data values appear in the selected column, displays only the first instance.
	ReorderColumns	Changes the order of the columns in the table.
	RepeatValues	Displays duplicate data values in the selected column.
	ShowColumns	Shows the selected columns.
CrosstabOperations	Analyze	Opens Data Analzyer.
	Delete	Deletes a cross tab.
	Edit	Opens the cross tab builder.
	SwitchView	Switches the cross tab view.
ManageData	ManageData	Opens the Manage Data dialog.
DeleteColumn	DeleteColumn	Deletes a column from a table in the report.
EditText	EditText	Enables editing of the selected static text.
FileOperations	New	Creates a new report design file.
	Open	Opens an existing report design.
	Save	Saves the current report design.
	SaveAs	Saves the current report design file under a different name or in a new location.
Formatting	AlignCenter	Centers the text in the selected column.

Table 4-1 User actions that you can enable or disable through erni_config.xml (continued)

Action set	Action	Description
	AlignLeft	Aligns the left sides of text in the selected column.
	AlignRight	Aligns the right sides of text in the selected column.
	Border	Draws a border around the selected column.
	Conditional Format	Formats data in a selected column, based on a specified condition.
	Data	Formats the display of data in the selected column.
	Font	Formats the font of data in the selected column.
GeneralOperations	Parameter	Displays the parameters, if any, for the current report.
	SwitchSummary Mode	Switches between summary table mode and detail table mode for the selected table.
	TableBuilder	Enables the table builder wizard.
Grouping	CreateSection	Adds a report section, which provides an additional level of data grouping.
	DeleteSection	Removes the selected report section.
	GroupBy	Groups table rows by values in the selected column.
	HideDetail	Hides the detail rows in a report section.
	PageBreak	Adds page breaks before or after a report section.
	ShowDetail	Shows the detail rows in a report section.
	UngroupBy	Removes groups in the selected column.
Help	Help	Shows help information.
PageLayout	PageLayoutIn Toolbar	Displays page layout toggle under toolbar. Disabled by default.
Preview	PreviewHTML	Shows a preview of the report in HTML format.
ReportItemOperations	Bookmark	Assigns a bookmark to a report item.
	Hyperlink	Assigns a hyperlink to a report item.

(continues)

Table 4-1 User actions that you can enable or disable through erni_config.xml (continued)

Action set	Action	Description
SectionOperations	SectionHeading	Shows the data fields in the report, and supports adding fields to the selected section heading.
Sorting	AdvancedSort	Sorts the table rows by the values of multiple columns.
	SortAscending	Sorts, in ascending order, the table rows by the values of the selected column.
	SortDescending	Sorts, in descending order, the table rows by the values of the selected column.
TemplateTable Operations	AutoSummarizeOn	If set to true, creates a summary table by default. If set to false, creates a detail table by default.
UndoRedo	Redo	Redo the last action.
	Undo	Undo the last action.

Configuration examples

This section provides examples of editing attributes in erni_config.xml, and the resulting changes to the BIRT Studio page.

Figure 4-4 shows the default BIRT Studio page with all actions enabled. The formatting actions on the toolbar and context menu are called out, so that you can see the difference in the toolbar and context menu when these actions are disabled.

Listing 4-2 shows a change to the Formatting action set. Its <visible> attribute, shown in bold, is set to false. Note, however, that all the actions under the Formatting action set are still enabled.

Listing 4-2 Visibility of the Formatting action set, changed to false

```
<actionSet>
   <name>Formatting</name>
   <visible>false</visible>
   <action>
      <name>AlignLeft</name>
      <enabled>true</enabled>
   </action>
   <action>
      <name>AlignCenter</name>
      <enabled>true</enabled>
   </action>
```

```
    <action>
       <name>AlignRight</name>
       <enabled>true</enabled>
    </action>
    <action>
       <name>Font</name>
       <enabled>true</enabled>
    </action>
    <action>
       <name>Border</name>
       <enabled>true</enabled>
    </action>
    <action>
       <name>ConditionalFormat</name>
       <enabled>true</enabled>
    </action>
    <action>
       <name>Data</name>
       <enabled>true</enabled>
    </action>
</actionSet>
```

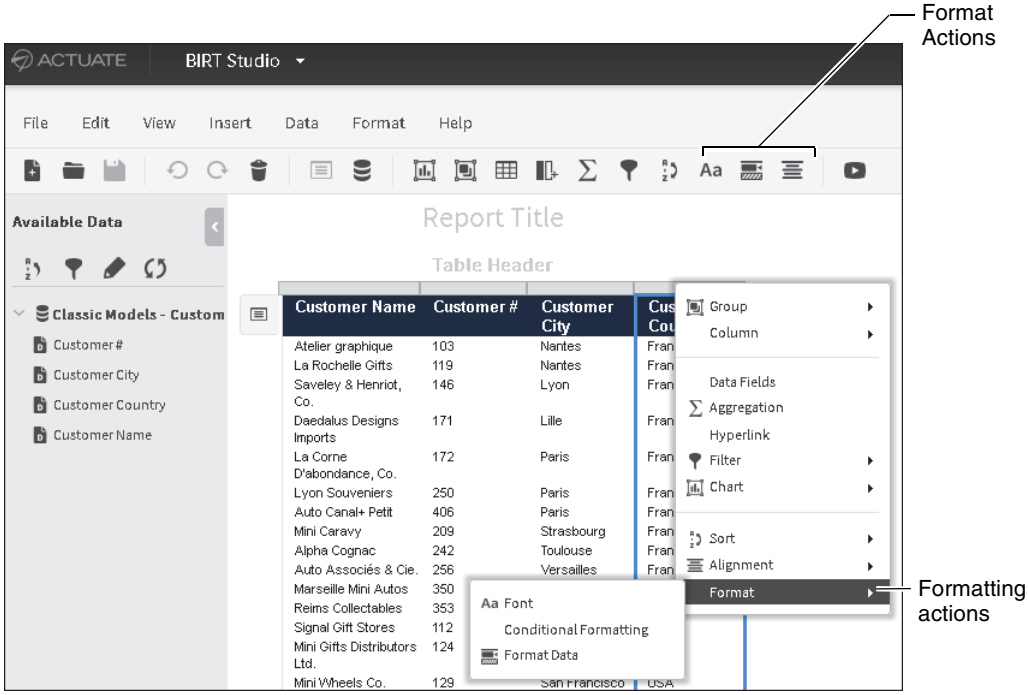


Figure 4-4 Default BIRT Studio page

Figure 4-5 shows the updated BIRT Studio page. None of the formatting actions appear in the toolbar or the context menu. Setting the <visible> attribute of an action set to false disables all actions within the action set.

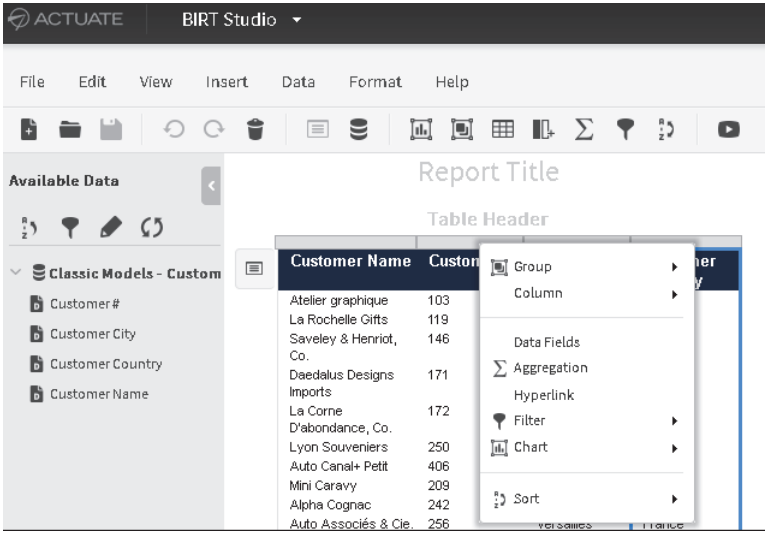


Figure 4-5 Updated toolbar and context menu, without any formatting functions

Listing 4-3 shows changes to the Font and Data actions within the Formatting action set. The Formatting action set's <visible> attribute is set to true. The Font and Data actions are disabled. The other actions in the action set are still enabled.

Listing 4-3 Font and Data (formatting) actions disabled

```
<actionSet>
   <name>Formatting</name>
   <visible>true</visible>
   <action>
      <name>AlignLeft</name>
      <enabled>true</enabled>
   </action>
   <action>
      <name>AlignCenter</name>
      <enabled>true</enabled>
   </action>
   <action>
      <name>AlignRight</name>
      <enabled>true</enabled>
   </action>
```

```
   <action>
      <name>Font</name>
      <enabled>false</enabled>
   </action>
   <action>
      <name>Border</name>
      <enabled>true</enabled>
   </action>
   <action>
      <name>ConditionalFormat</name>
      <enabled>true</enabled>
   </action>
   <action>
      <name>Data</name>
      <enabled>false</enabled>
   </action>
</actionSet>
```

Figure 4-6 shows the updated BIRT Studio page. The alignment actions are available on the toolbar and on the context menu, but not the Font and Data formatting actions.

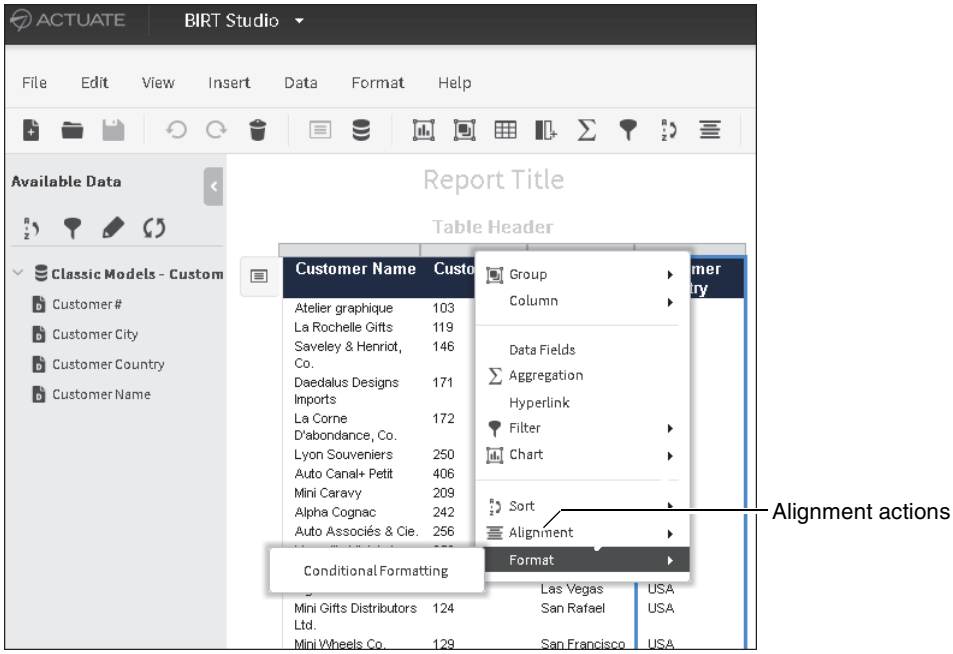


Figure 4-6 Updated toolbar and context menu, without the Font and Data formatting actions

Specifying the default position of aggregate values

The default location of aggregate values is in a group's header. To place aggregate values in a group's footer, set EnableNewAggregationStyle to false in erni_config.xml:

```
<featureConfigs>
   <featureConfig>
      <name>EnableNewAggregationStyle</name>
      <value>false</value>
   </featureConfig>
...
</featureConfigs>
```

Using sample data in a preview

Actual data is used to generate a preview by default. To improve preview performance, you can configure BIRT Studio to use sample data instead, which uses dummy values. To enable sample data in a preview, set EnableSampleDataInPreview to true in erni_config.xml:

```
<featureConfigs>
...
   <featureConfig>
      <name>EnableSampleDataInPreview/name>
      <value>true</value>
   </featureConfig>
...
</featureConfigs>
```

Configuring advanced data operations

You can enable or disable the following advanced data options in Available Data in the report design area of BIRT Studio:

- Modify enables the user to change the data set by joining it with one or more information objects.
- Synchronize Data Sets enables the user to update the data set in the report design with the current data in the information object on the volume.

Figure 4-7 shows the data options in Available Data.

By default, these data options are disabled by default and they are in effect only when the data sources are information objects. The Modify and Synchronize Data Sets buttons do not appear unless enabled by configuration.

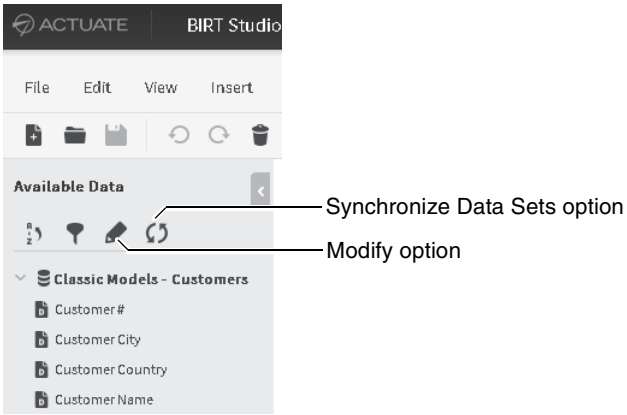


Figure 4-7 Advanced data options

You can enable these options by editing attributes in one of Java Components' configuration files, functionality-level.config. For information about all the functionality options listed in functionality-level.config, see Chapter 6, "Configuring BIRT Studio."

Listing 4-4 highlights in bold the advanced data option that can enable the advanced data option user interface buttons.

Listing 4-4 Intermediate and Advanced levels in functionality-level.config

```
<Level>
   <Name>All</Name>
   ...
   <SubfeatureID>InteractiveViewing</SubfeatureID>
   <SubfeatureID>AdvancedData</SubfeatureID>
   <SubfeatureID>DashboardDeveloper</SubfeatureID>
</Level>
```

Unlike the functionality options you set in erni_config.xml, the options you set in functionality-level.config apply at when a user logs in, not at the application level. The Information Object integration service must be enabled for the buttons to function.

Configuring the application environment

You can change the values of the configuration parameters in the following file:

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

BIRT Studio parameters control how BIRT Studio operates as a web application and how it interoperates with other Java Components. Table 4-2 describes the configuration parameters for BIRT Studio.

Table 4-2 BIRT Studio web.xml parameters

Parameter name	Description
BIRT_REPORT_DESIGN_CACHE_TIMEOUT	Specifies the amount of time, in seconds, before a cached BIRT report design is purged if it has not been used. The default value is 1800, which is 30 minutes.
BIRT_REPORT_DESIGN_CACHE_TOTAL_NUMBER_OF_ENTRIES	Specifies the maximum number of BIRT report designs to cache. The default value is 50.
BIRT_REPORT_DOCUMENT_CACHE_ENABLED	Specifies whether to cache BIRT report documents when they are previewed or generated. The default value is true.
BIRT_REPORT_PAGE_COUNT_CACHE_ENABLED	Specifies whether to cache the number of pages in transient or persistent BIRT report documents when they are previewed or generated. The default value is true.
BIRT_RESOURCE_PATH	Path to Actuate BIRT shared resources, including libraries and templates for Actuate BIRT report designs and BIRT Studio. The default value is <context root>\resources.
BIRT_SCRIPT_LIB_PATH	Path to script libraries. The default value is <context root> \scriptlib.
BRSTUDIO_DESIGN_SESSION_TIMEOUT	The design session time out limit in seconds. If not specified, the design session times out based on the login time out value.
DEFAULT_DATA_CACHE_ROW_COUNT	The number of data rows to display in BIRT Studio when designing a report. The default value is 100.
DEFAULT_LOCALE	The default locale. The default locale is en_US. Users can select a locale when they log in.
DEFAULT_PAGE_BREAK_INTERVAL	The number of rows to display on one page when viewing a report. A value of 0 indicates no page breaks.
DEFAULT_REPORT_TEMPLATE_CATEGORY_NAME	The default BIRT report template category to load when a user opens BIRT Studio. The default value is Standard.
DEFAULT_TIMEZONE	The default time zone. The default time zone is Pacific Standard Time (PST).
MAX_BRSTUDIO_DESIGN_SESSION	The maximum number of designs a user can edit concurrently in BIRT Studio. The default is 10.
MAX_BRSTUDIO_USER_SESSION	The maximum number of concurrent BIRT Studio sessions on the server. The default is 256.
MAX_DATA_CACHE_ROW_COUNT	Limits the number of data rows that a user can choose to display in Actuate BIRT Studio when designing a report. The default value is 200.

Table 4-2 BIRT Studio web.xml parameters

Parameter name	Description
MAX_NUMBER_OF_VALUES_FOR_DYNAMIC_PARAMETER	The number of values shown in the parameter dialog box for a dynamic value parameter in BIRT Studio: ■ A positive number value N means only the first N values appear the parameter dialog box. ■ A value of 0 means all values from the data source appear in the parameter dialog box. The default value is 0. ■ A value of -1 means only the first N values appear where N is the current data cache row count setting for the current design session. MAX_NUMBER_OF_VALUES_FOR_DYNAMIC_PARAMETER only applies to a dynamic value parameter. All the values appear for a static value parameter no matter how many values it has. For a static value parameter, the full list appears in the parameter dialog box when the user chooses Save and View.
MEMORY_DATA_CACHE_ROW_COUNT	Specifies the number of data rows to cache in memory. The default value is 50.
MORE_VALUE_ROW_COUNT	Specifies the number of rows to fetch when a user chooses to filter a report on a column in BIRT Studio. The default value is 200.
PERSISTENT_ARCHIVEFILECACHE_TIMEOUT_SECONDS	Specifies the amount of time, in seconds, before a cached file that was created from a repository file is purged if it has not been used. The default value is 7200, which is 120 minutes.
SEARCH_ENABLE_COLUMN_HEADERS	Indicates whether to include column headings in report search results when the output format is CSV or TSV. Set this parameter to true, the default value, to include column headings.
SEARCH_USE_QUOTE_DELIMITER	Indicates whether to enclose search results in quotation marks when the output format is CSV or TSV. The default value is true, which encloses the results in quotation marks.
TRANSIENT ARCHIVEFILECACHE_TIMEOUT_SECONDS	Specifies the amount of time, in seconds, before a cached file generated without saving it to the repository is archived if it has not been used. The default value is 1200, which is 30 minutes.

Chapter

7

Actuate Java Components URIs

This chapter contains the following topics:

- Actuate Java Components URIs overview
- Actuate Java Components URIs quick reference
- Common URI parameters
- Java Components Struts actions
- Actuate Java Components URIs reference
- Actuate BIRT Viewer URIs reference
- Actuate BIRT Studio URIs Reference

Actuate Java Components URIs overview

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

Java Components 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 Components page and folder names are case-sensitive.

Actuate Java Components URIs quick reference

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

Table 5-1 Actuate Java Components URI pages

Actuate Java Components page	Description
authenticate page	Performs authentication and maintains user, cluster, and volume information.
banner page	Displays a banner at the top of each Actuate Java Components 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 canceling 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.
getfolderitems page	See file and folder index page.
index page	Provides the link from the My Folder button to the Actuate Java Components home page.

Table 5-1 Actuate Java Components URI pages

Actuate Java Components page	Description
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.
viewer page for Actuate BIRT reports	Displays Actuate BIRT documents along with the toolbar.

Common URI parameters

All Actuate Java Components 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 Components authentication using cookies.

Table 5-2 Common Actuate Java Components 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 Components 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 Java Components pages.
locale	The current user's locale, such as U.S. English (en-US). Java Components 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 an Actuate web application, such as http://Services:8000.
timezone	The current user's time zone.

(continues)

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

URI parameter	Description
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 Java Components pages.

The following Java Components 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 Components 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 Components global forwards

Action	Forward
authexpired	/login.do
error	/private/common/errors/errorpage.jsp
executereport	/executereport.do
login	/login.do
logout	/logout.do
viewpage	/servlet/ViewPage

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

Table 5-4 Actuate Java Components actions

Action	Input JSP	Forward name path
/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=error path=/iportal/activePortal/ private/filesfolders/deletefilestatus.jsp
		name=confirm path=/iportal/activePortal/private /filesfolders/confirm.jsp
/executereport	/private/newrequest /newrequest.jsp	name=viewbirt path=/iv
		name=viewreport path=/servlet/DownloadFile
		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

(continues)

Table 5-4 Actuate Java Components actions (continued)

Action	Input JSP	Forward name path
/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
/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

Table 5-4 Actuate Java Components actions (continued)

Action	Input JSP	Forward name path
/treebrowser		name=success path=/iportal/activePortal/private /filesfolders/treebrowser.jsp
/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 Components URIs reference

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

Table 5-5 Actuate Java Components pages

Topic	Java Components file
authenticate page	iportal\activePortal\authenticate.jsp
banner page	iportal\activePortal\private\common\banner.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

(continues)

Table 5-5 Actuate Java Components pages (continued)

Topic	Java Components file
error page	errors\error.jsp iportal\activePortal\private\common\errors\error.jsp
execute report page	executereport.do
index page	
■ file and folder index page	getfolderitems.do iportal\activePortal\private\filesfolders\filefolderlist.jsp
list page	
■ file and folder list page	getfolderitems.do iportal\activePortal\private\filesfolders\filefolderlist.jsp
login page	login.do iportal\activePortal\private\login.jsp
logout page	logout.do
viewer page for Actuate BIRT Reports	IVServlet

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. Java Components 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 Components 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

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 Components perform 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 5-8.

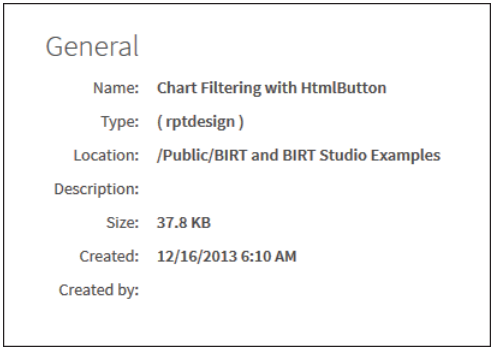


Figure 5-8 Home folder detail page

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 Components use two pages. All Java Components 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 Components 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.
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.

(continues)

Table 5-8 Execute Report URI parameters (continued)

URI parameter	Description
__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 Components wait for the report generation to be completed before displaying it. If "nowait", Java Components display 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

index page

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

file and folder index page

The default entry point to the Deployment Kit 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 Deployment Kit uses to format

and present files and folders data. Page content varies depending on the Actuate Java Components 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 Components call 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.

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, Deployment Kit 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 Components 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 page

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

Actuate BIRT Viewer URIs reference

The BIRT Viewer is a Java servlet that manages binary content and performs tasks such as uploading and downloading binary files. Invoke the BIRT Viewer servlet using the following syntax:

```
http://<application server>:<port>/<context root>/iv
```

- application 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 Components context root.
- iv is the name to which the servlet is mapped in the web application'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.

Actuate BIRT Viewer fully supports the URIs for the open-source BIRT Viewer. After migrating from open-source BIRT to Actuate BIRT, you can use the same URIs in the Actuate BIRT Viewer that you used in open-source BIRT.

BIRT Viewer

The BIRT Viewer servlet provides tools to display and affect BIRT document and design files. This servlet provides both the BIRT Viewer and the BIRT Interactive Viewer. The Interactive Viewer is licensed separately from the BIRT Viewer. To create a link using the URL provided by the Link to this page menu item in the viewer, the HTML page containing the link must use a strict Document Type Definition (DTD). To use the strict DTD, use the following code at the beginning of the HTML page markup:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

The BIRT Viewer provides navigation toolbar options, as shown in Figure 5-9.

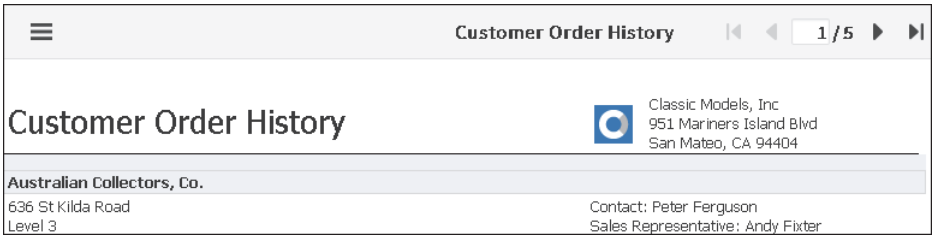


Figure 5-9 BIRT Viewer

The Interactive Viewer displays the report with toolbar options to navigate the report, a menu to perform additional tasks, and provides context menus to edit and format report elements, as shown in Figure 5-10.

The BIRT Viewer servlet supports rptdocument file formats. When an rptdesign files runs, a rptdocument file is generated and displays in the BIRT Viewer.

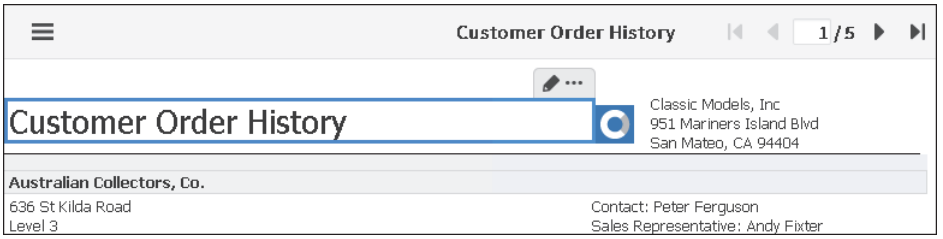


Figure 5-10 BIRT Interactive Viewer

Name com.actuate.iv.servlet.IVServlet

URI parameters Table 5-1 lists and describes the URI parameters for the Interactive Viewer servlet.

Table 5-1 IVServlet 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: Microsoft Excel ■ doc: Microsoft Word ■ ppt: Microsoft PowerPoint ■ ps: PostScript ■ html: HTML ■ flashchartsxml: used to display a fusion chart ■ flashgadgetsxml: used to display Flash gadgets in a fusion chart. ■ reportlet: This is used together with __bookmark to show a particular part/element of the report.
__from_page_range	The page range of a report to display.
__from_page_style	The page style to use for a report in pdf or ps formats: ■ auto: The page size and content size remains the same. ■ actualSize: Change the page size to fit the content. ■ fitToWholePage: Change the content size to fit the page size. Used with the __format parameter.

(continues)

Table 5-1 IVServlet URI parameters (continued)

URI parameter	Description
__imageid	Identifier of the report file image to display.
__instanceid	Identifier of the report file to display.
__launchiv	A Boolean value that enables interactivity.
__locale	Code for a locale. For example FR_fr specifies the French language and the country, France.
__page	The number of a page to render.
__report	Name of the report file to display.
__rtl	Boolean value that specifies right-to-left orientation for the report.

For example, to access the first version of the Top Sales Peformers.rptdocument from the local host, use a URI similar to the following:

```
127.0.0.1:8080/iportal/iv?__report=/Home/administrator/Top Sales
   Performers.rptdocument;1
```

Actuate BIRT Studio URIs Reference

BIRT Studio is a web application that is initiated by a Java servlet. The BIRT Studio servlet manages binary content and performs tasks such as uploading and downloading binary files.

You invoke the BIRT Studio servlet using the following syntax:

```
http://<web server>:<port>/<context root>/wr
```

- web server is the fully qualified domain name or IP address of the machine hosting the web or application server.
- port is the port on which the application server listens for requests.
- context root is the BIRT Report Studio context root.
- wr is the name to which the servlet is mapped in the web application's web.xml file. A typical location for web.xml is <context root>\WEB-INF.

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

BIRT Studio

The BIRT Studio servlet loads the BIRT Studio user interface and establishes a connection to a report repository. A report repository is required in order to use the servlet.

Name com.actuate.erni.servlet.ERNIViewerServlet

Invoke the BIRT Studio servlet as:

```
http://<web server>:<port>/<context root>/wr?<parameters>
```

URI parameters The BIRT Studio servlet requires repository parameters in order to operate. Table 5-2 lists and describes the URI parameters for the BIRT Studio servlet.

Table 5-2 BIRT Studio URI parameters

URI parameter	Description
repositoryType	The repository type. Use Enterprise for a volume.
serverURL	The URL of a web or application server machine.
volume	The name of a volume that is managed by the URL to which you connect.
__vp	The name of a server configured in VolumeProfile.xml. BIRT Studio uses the volume information in a VolumeProfile entry except when a volume parameter specifies a different one.

In addition to the initial BIRT Studio page, you can open BIRT Studio with:

- A specific report design
- A specific template
- A report design that accesses a specific information object
- A report design that accesses a specific information object and a report template

In the example URLs in the following topics, special characters are represented by codes, as shown in Table 5-3.

Table 5-3 Codes for special characters in URLs

Character	Code
Colon (:)	%3a
Slash (/)	%2f
Period (.)	%2e

(continues)

Table 5-3 Codes for special characters in URLs (continued)

Character	Code
Space ( )	%20

To open an existing report design in BIRT Report Studio, use a URL like the one shown in the following example:

```
http://urup.domain.com:8080/ajc/wr?__report=
   %2fApplications%2fBIRT%20Sample%20App%2fCustomer%20Order%20Hist
   ory.rptdesign&pCountry=USA
```

- __report=%2fApplications%2fBIRT%20Sample%20App%2fCustomer%20Order%20History.rptdesign is the path to the report design to use.
- pCountry=USA is a parameter-value pair for the report design.

Chapter

8

Actuate Java Components JavaBeans

This chapter contains the following topics:

- Java Components JavaBeans overview
- Java Components JavaBeans package reference
- Java Components JavaBeans class reference

Java Components JavaBeans overview

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

Java Components JavaBeans package reference

Table 9-1 lists and describes the Actuate packages used in Java Components.

Table 9-1 Java Components 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 Components JavaBeans class reference

Documents

Table 9-2 lists and describes Java Components com.actuate.activeportal.forms classes that support the Document pages.

Table 9-2 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 9-2 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 9-3 describes the Java Components com.actuate.activeportal.beans class that supports general functionality.

Table 9-3 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 9-4 lists and describes Java Components com.actuate.activeportal.forms classes that support general functionality.

Table 9-4 General forms classes

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

Jobs

Table 9-5 lists and describes Java Components com.actuate.activeportal.forms classes that support jobs.

Table 9-5 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 Components security

This chapter contains the following topics:

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

About Actuate Java Components 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 Components 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 Components 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 Components authentication," later in this chapter.

Protecting corporate data

Actuate Java Components provide a structured content generation solution for web applications. Deploying Actuate applications developed for the internet, such as Java Components, 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 Components 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 Components, 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 Components page or by typing an Actuate Java Components URI in a web browser. A Java Components application processes the request.
- If a custom security adapter parameter is set in the web.xml file, the Java Components attempt to load the custom security adapter class. If the class loads successfully, the following steps occur:
  - The Java Components call the custom security adapter's authenticate( ) method with the parameters that the browser sent.
  - The authenticate( ) method performs the custom validation.
  - The Java Components call the getUserName( ), getPassword( ), and getUserHomeFolder( ) methods to retrieve the user information the Actuate web service requires.
  - Optionally, the Java Components call 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 Components authentication

To customize Actuate Java Components 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 Components application. The lib directory for your Java Components 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 Components security adapter is designed so that other applications can authenticate users and log into Java Components using a URL. When a URL activates a custom Java Components security adapter, access is granted based on the security adapter's logic. A Java Components security adapter establishes an additional layer of logic to the existing Java Components, as shown in Figure 10-1.

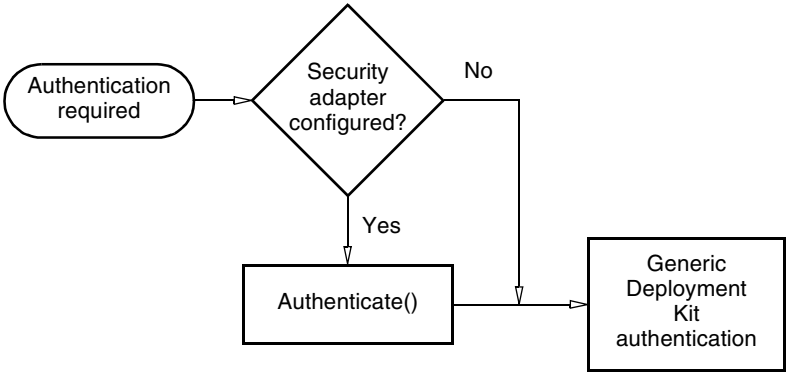


Figure 10-1 Java Components security model with an optional security adapter

The Java Components 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 Components.
- Set the Java Components configuration file web.xml to use the new class.
- Deploy the Custom Security Adapter.

Accessing the IPSE Java classes

The Java Components library, com.actuate.iportal.jar, contains the IPSE Java classes. This library is located in the lib subdirectory in the Java Components 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\org.apache.xerces_<version>.jar
- <context root>\WEB-INF\lib\com.actuate.webcommon.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 using a custom security adapter, the following methods are required:

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

The login module of the Java Components call 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 Components 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 application server 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 a volume. The Login module calls isEnterprise( ) to determine whether to use a repository on the file system.

Returns False.

Part Three

Using Actuate JavaScript API

Chapter

10

Creating a custom web page using the Actuate JavaScript API

This chapter contains:

- About the Actuate JavaScript API
- Accessing the Actuate JavaScript API
- Establishing an HTTP session with an Actuate web application
- About Actuate JavaScript API security integration
- Viewing reports
- Navigating repository content using ReportExplorer
- Using and submitting report parameters
- Retrieving report content as data
- Controlling Interactive Viewer user interface features

About the Actuate JavaScript API

The Actuate JavaScript API enables the creation of custom web pages that use Actuate BIRT report elements. The Actuate JavaScript API handles connections, security, and content. The Actuate JavaScript API classes functionally embed BIRT reports or BIRT report elements into web pages, handle scripted events within BIRT reports or BIRT report elements, package report data for use in web applications, and operate BIRT Viewer and Interactive Crosstabs.

To use the Actuate JavaScript API, connect to Actuate Java Components or Deployment Kit.

The Actuate JavaScript API uses the Prototype JavaScript Framework. The following directory contains the Actuate JavaScript API source files:

```
<Context Root>\iportal\jsapi
```

The base class in the Actuate JavaScript API is actuate. The actuate class is the entry point for all of the Actuate JavaScript API classes. The actuate class establishes connections to the Actuate web application services. The Actuate JavaScript API uses HTTP requests to retrieve reports and report data from an Actuate web service. The subclasses provide functionality that determines the usage of the reports and report data.

Many functions in the Actuate JavaScript API use a callback function. A callback function is a custom function written into the web page that is called immediately after the function that calls it is finished. A callback function does not execute before the required data or connection has been retrieved from the server.

Many of the callback functions in the Actuate JavaScript API use a passback variable. A passback variable contains data that is passed back to the page by the calling function. A callback function that uses an input parameter as a passback variable must declare that input parameter.

Accessing the Actuate JavaScript API

To use the Actuate JavaScript API from a web page, add a script tag that loads the Actuate JavaScript API class libraries from an Actuate application.

Start with a web page that contains standard HTML elements, as shown in the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
```

```
   <meta http-equiv="content-type" content="text/html;
   charset=utf-8" />
   </head>
   <body>
         <div id="viewer1">
            <script type="text/javascript" language="JavaScript"
               src="http://127.0.0.1:8080/iportal/jsapi"></script>
            <script type="text/javascript" language="JavaScript">
            … <!--functionality goes here-->
            </script>
         </div>
      </body>
</html>
```

The <script> element nested in the element imports the Actuate JavaScript API libraries into the web page's context. For example:

```
<script type="text/javascript" src="http://127.0.0.1:8080
   /iportal/jsapi">
</script>
```

where

- 127.0.0.1:8080 is the host name and TCP port for an available Actuate application host.
- /iportal is the context root for the Actuate web service.
- /jsapi is the default location of the Actuate JavaScript API libraries.

Use additional script tags to call JavaScript functions for the page. Use the actuate.load( ) function to enable the components of the Actuate JavaScript API.

The scripts in this section are encapsulated in tags for portability. Encapsulated Actuate JavaScript API functions can be used in any web page.

About the DOCTYPE tag

To render the page in standards compliance mode, specify strict.dtd in the DOCTYPE tag at the top of the page. Standards compliance mode makes the page layout and behaviors significantly more consistent. Pages without this definition render inconsistently.

About UTF8 character encoding

Use a <meta> tag to direct the browser to use UTF8 encoding for rendering and sending data. UTF8 encoding prevents the loss of data when using internationalized strings.

Establishing an HTTP session with an Actuate web application

The actuate class is the general controller for the HTTP session. Call actuate.initialize( ) to establish a connection to an Actuate application. Load the elements that are selected by actuate.load( ) before accessing reports or applications. Initialization establishes a session with an Actuate service. To initialize the actuate object, call the actuate.initialize( ) initialization function. To use actuate.initialize( ), provide connection parameters as shown in the following code:

```
actuate.initialize("http://127.0.0.1:8080/ajc", reqOps, null,
   null, runReport, null);
```

where

- http://127.0.0.1:8080/ajc is a URL for the Actuate report application service. This URL must correspond to an Actuate Java Components application.
- reqOps specifies an actuate.RequestOptions object, which is required for most operations. A default request options object can be generated by calling the constructor without any input parameters, as shown in the following code:

  ```
  var reqOps = new actuate.RequestOptions( );
  ```

  Additional options are required to support specific classes. For example, to use dashboards and gadgets, set the repository type to encyclopedia before calling initialize using code similar to the following.

  ```
  reqOps.setRepositoryType(
          actuate.RequestOptions.REPOSITORY_ENCYCLOPEDIA);
  var reqOps = new actuate.RequestOptions( );
  ```

- The third and fourth parameters are reserved. Leave these parameters as null.
- runReport is the callback function called after the initialization finishes. Specify the callback function on the same page as the initialize function. The callback function cannot take a passback variable.
- null specifies the optional errorCallback parameter. The errorCallback parameter specifies a function to call when an error occurs.

The initialization procedure in this section is the first step in using Actuate JavaScript API objects. Nest the initialization code in the second <script> element in the element of the page.

The runReport( ) function is used as a callback function that executes immediately after actuate.initialize( ) completes. The page must contain runReport( ).

About Actuate JavaScript API security integration

The web service that provides reports also establishes security for a reporting web application. The actuate.initialize( ) function prompts users for authentication information if the web service requires authentication. The Actuate JavaScript API uses a secure session when a secure session already exists. Remove authentication information from the session by using actuate.logout( ).

To integrate an Actuate JavaScript API web page with an Actuate reporting web service, identify the web service from the following list:

- BIRT Viewer Toolkit: Actuate BIRT Viewer Toolkit is a freeware BIRT Viewer that is secured by the web server that runs it. BIRT Viewer Toolkit does not perform an authentication step initially, which enables the Actuate JavaScript API to integrate smoothly.
- Java Components using file-system repositories: Actuate Java Components provide web services that are secured by the application server that runs those services. These applications do not perform an authentication step initially, which enables the Actuate JavaScript API to integrate smoothly.
- Java Components using a volume repository: Volumes are managed by Actuate BIRT iHub. To connect to Java Components that access a volume, an Actuate JavaScript API web page prompts the user for a user name and password if a secure session has not been established. See Chapter 9, "Using Actuate Java Components security" for information about customizing security for Actuate Deployment Kit.
- iHub Information Console: Actuate iHub Information Console connects to a volume and requires authentication. To connect to Information Console, an Actuate JavaScript API web page prompts the user for a user name and password if a secure session has not been established. Actuate iHub provides a login page to establish the secure session. See *Integrating Applications into BIRT iHub* for information about customizing security for Actuate iHub Information Console.

Establishing a secure connection to more than one web service

The actuate.initialize( ) function establishes a session with one Actuate web application service, requesting authentication from the user when the web service requires authentication. Use the actuate.authenticate( ) function for additional secure sessions. Call actuate.authenticate( ) to establish secure sessions with additional web services. Call actuate.initialize( ) before calling actuate.authenticate( ).

Use authenticate( ) as shown in the following code:

```
actuate.authenticate(serviceurl,
                     null,
                     userID,
                     userpassword,
                     null,
                     callback,
                     errorcallback);
```

where

- serviceurl is a URL for the Actuate web application service in use. This URL must correspond to an Actuate Java Components application.
- null specifies the default settings for the RequestOptions object that is provided by the connected Actuate web application. RequestOptions sets custom or additional URL parameters for the request. To use custom or additional URL parameters, construct an actuate.RequestOptions object, assign the specific values to the object, and put the object into the custom or additional URL parameter.
- userID is the userid for authentication when loading Actuate JavaScript API resources. To force a user login, set this parameter to null.
- userpassword is the password for the userid parameter to complete authentication when loading Actuate JavaScript API resources. Use null to force the user to log in.
- null specifies no additional user credentials. This parameter holds information that supports external user credential verification mechanisms, such as LDAP. Add any required credential information with this parameter where additional security mechanisms exist for the application server upon which the web service is deployed.
- callback is a function to call after the authentication completes.
- errorcallback is a function to call when an exception occurs.

After authenticate( ) finishes, access resources from the Actuate web application service at the URL in serviceurl.

Application servers share session authentication information to enable a user to log in to one application context root and have authentication for another. For example, for Apache Tomcat, setting the crossContext parameter to "true" in the server.xml Context entries allows domains to share session information. The entries to share the authentication information from the web application with an Actuate Java Component look like the following example:

```
<Context path="/MyApplication" crossContext="true" />
<Context path="/ActuateJavaComponent" crossContext="true" />
```

Using a login servlet to connect to an Actuate web application

Actuate web applications provide a login servlet, loginservlet, that establishes a secure session with an Actuate web application service. Use the following code to use a form that calls loginservlet explicitly from a login page:

```
<form name="Login"
action="https://myApp/iPortal/loginservlet?" function="post">
   <input type="text" name="userID" />
   <input type="text" name="password" />
   …
</form>
```

This code sets username and password variables in the session. When initialize( ) runs, the Actuate JavaScript API looks up the session map in the current HTTP session, using the service URL as the key. The Actuate JavaScript API finds the session established by login servlet and accepts the authentication for that service URL.

The login servlet authenticates the connection to an Actuate web service. Do not call the actuate.authenticate( ) function to authenticate the connection when using loginservlet.

Using a custom servlet to connect to an Actuate web application

Actuate web applications provide single-sign-on functionality to authenticate users using a custom security adapter. See *Actuate Application Administrator Guide* for details on creating and using a custom security adapter matching a specific deployment scenario.

Unloading authentication information from the session

The Actuate JavaScript API keeps authentication information encrypted in the session. To remove this information from the session, use actuate.logout( ). Use logout( ) as shown in the following code:

```
actuate.logout(serviceurl,
               null,
               callback,
               errorcallback);
```

where

- serviceurl is a URL for the Actuate web application service to log out from. This URL must correspond to an Actuate Java Components application.

- null specifies the default settings for RequestOptions that are provided by the connected Actuate web application. RequestOptions sets custom or additional URL parameters for the request. To use custom or additional URL parameters, construct an actuate.RequestOptions object, assign the specific values to the object, and put the object into the custom or additional URL parameter.
- callback is a function to call after logout( ) completes.
- errorcallback is a function to call when an exception occurs.

After logout( ) finishes, the authentication for the serviceurl is removed. Authenticate again to establish a secure connection.

Viewing reports

The actuate.Viewer class loads and displays reports and report content. Load actuate.Viewer with actuate.load( ) before calling actuate.initialize( ), as shown in the following code:

```
actuate.load("viewer");
```

Load support for dialog boxes from the Actuate JavaScript API using the actuate.load function, as shown in the following code:

```
actuate.load("dialog");
```

Load the viewer and dialog components to use the viewer on the page. Call actuate.Viewer functions to prepare a report, then call the viewer's submit function to display the report in the assigned element.

The actuate.Viewer class is a container for Actuate reports. Create an instance of actuate.Viewer using JavaScript, as shown in the following code:

```
var myViewer = new actuate.Viewer( "viewer1" );
```

The "viewer1" parameter is the name value for the element which holds the report content. The page body must contain a element with the id viewer1 as shown in the following code:

```
<div id="viewer1"></div>
```

Use setReportName( ) to set the report to display in the viewer, as shown in the following code:

```
myViewer.setReportName("/public/customerlist.rptdocument");
```

SetReportName accepts a single parameter, which is the path and name of a report file in the repository. In this example, "/public/customerlist.rptdocument" indicates the Customer List report document in the /public directory.

Call viewer.submit( ) to make the viewer display the report, as shown in the following code:

```
myViewer.submit( );
```

The submit( ) function submits all the asynchronous operations that previous viewer functions prepare and triggers an AJAX request for the report. The Actuate web application returns the report and the page displays the report in the assigned element.

This is an example of calling viewer( ) in a callback function to display a report:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
   <meta http-equiv="content-type" content="text
      /html;charset=utf-8" />
   <title>Viewer Page</title>
</head>
<body onload="init( )">
<div id="viewerpane">
   <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8080/iportal/jsapi"></script>

   <script type="text/javascript" language="JavaScript">

   function init( ){
      actuate.load("viewer");
      var reqOps = new actuate.RequestOptions( );
      actuate.initialize( "http://127.0.0.1:8080/iportal", reqOps,
           null, null, runReport);
   }  function runReport( ) {
      var viewer = new actuate.Viewer("viewerpane");
      viewer.setReportName("/Public/BIRT and BIRT Studio
              Examples/Top 5 Sales Performers.rptdocument");
      viewer.submit(callback);
   }
   </script>
</div>
</body>
</html>
```

The viewer component displays an entire report. If the report is larger than the size of the viewer, the viewer provides scroll bars to navigate the report. To display a specific element of a report instead of the whole report, use viewer.setReportletBookmark( ) prior to calling submit( ), as shown in the following code:

```
function init( ) {
   actuate.load("viewer");
   var reqOps = new actuate.RequestOptions( );
   actuate.initialize( "http://127.0.0.1:8080/iportal", reqOps,
        null, null, runReport);
   function runReport( ) {
   var viewer = new actuate.Viewer("viewerpane");
   viewer.setReportName("/Public/BIRT and BIRT Studio
           Examples/Top 5 Sales Performers.rptdocument");
   viewer.setReportletBookmark("FirstTable");
   viewer.submit(callback);
```

When the FirstTable bookmark is assigned to any table, this code displays that table.

Any changes to the report display must take place after viewer.submit( ) completes. Embed presentation code in a callback class to ensure proper execution.

Controlling viewer user interface features

Control the viewer controls and interface features with the actuate.viewer.UIOptions class. Create an instance of this class using JavaScript, as shown in the following code:

```
var uioptions = new actuate.viewer.UIOptions( );
```

Set the user interface options with the enable functions in the actuate.viewer.UIOptions class. For example, a toolbar appears in the viewer by default, as shown in Figure 11-1.

Figure 11-1 The default toolbar for the JavaScript API viewer

To disable this toolbar, use the following code:

```
uioptions.enableToolBar(false);
```

All of the enable functions take a Boolean value as an argument. To configure the viewer to use these options, use setUIOptions( ) as shown in the following code:

```
viewer.setUIOptions(uioptions);
```

The setUIOptions( ) function accepts one parameter: an actuate.viewer.UIOptions object. The viewer's submit( ) function commits the user interface changes to the viewer when the function sends the object to the HTML container. Set the UI options using setUIOptions( ) before calling submit( ).

Accessing report content

Use the actuate.report subclasses to access report content that is displayed in the viewer. For example, use the actuate.report.Table subclass to manipulate a specific table on a report. To manipulate a specific text element in a report, use the actuate.Viewer.Text subclass. Use viewer.getCurrentPageContent( ) to access specific subclasses of actuate.report as shown in the following code:

```
var myTable= myViewer.getCurrentPageContent( ).
   getTableByBookmark("mytable");
```

Identify report elements by their bookmarks. Set bookmarks in the report design. The viewer subclasses access specific report elements and can change how they are displayed.

To hide a particular data column in the table, use code similar to the following function as the callback function after submitting the viewer:

```
function hideColumn( ){
var myTable=
   myViewer.getCurrentPageContent().getTableByBookmark("mytable");
if ( myTable) {
   myTable.hideColumn("PRODUCTCODE");
   myTable.submit( );
   }
}
```

Hiding the column PRODUCTCODE suppresses the display of the column from the report while keeping the column in the report. Elements that use the PRODUCTCODE column from mytable retain normal access to PRODUCTCODE information and continue to process operations that use PRODUCTCODE information.

Accessing HTML5 Chart features

HTML5 charts are accessed from the viewer using actuate.viewer.getCurrentPageContent( ).getChartByBookmark( ) like other report charts. To access HTML5 chart features, use the actuate.report.HTML5Chart.ClientChart object to handle the chart. For example, to access the HTML5 chart with the HTML5ChartBookmark, use the following code:

```
var bchart = this.getViewer().getCurrentPageContent().
   getChartByBookmark("HTML5ChartBookmark");
var clientChart = bchart.getClientChart();
```

ClientChart provides access to ClientOptions, which can change chart features. For example, to change an HTML5 chart title to Annual Report, use the following code:

```
clientChart.getClientOptions().setTitle('Annual Report');;
clientChart.redraw();
```

Using a filter

Apply a data filter to data or elements in a report, such as a charts or tables, to extract specific subsets of data. For example, the callback function to view only the rows in a table with the CITY value of NYC, uses code similar to the following function:

```
function filterCity(pagecontents){
var myTable = pagecontents.getTableByBookmark("bookmark");

var filters = new Array( );
var city_filter = new actuate.data.Filter("CITY",
   actuate.data.Filter.EQ, "NYC");
filters.push(city_filter);

myTable.setFilters(filters);
myTable.submit(nextStepCallback);
}
```

In this example, the operator constant actuate.data.filter.EQ indicates an equals (=) operator.

Using a sorter

A data sorter can sort rows in a report table or cross tab based on a specific data column. For example, to sort the rows in a table in descending order by quantity ordered, use code similar to the following function as the callback function after submitting the viewer:

```
function sortTable( ){
var btable = this.getViewer( ).getCurrentPageContent( ).
   getTableByBookmark("TableBookmark");

var sorter = new actuate.data.Sorter("QUANTITYORDERED", false);
var sorters = new Array( );
sorters.push(sorter);

btable.setSorters(sorters);
btable.submit( );
}
```

The first line of sortTable( ) uses the this keyword to access the container that contains this code. Use the this keyword when embedding code in a report or report element. The this keyword doesn't provide reliable access to the current viewer when called directly from a web page.

Navigating repository content using ReportExplorer

Use the actuate.ReportExplorer class to navigate and view the contents of a volume in a generic graphical user interface. Load the actuate.ReportExplorer class with actuate.load( ), as shown in the following code:

```
actuate.load("reportexplorer");
```

Call actuate.ReportExplorer functions to identify the root directory to display then call the ReportExplorer's submit function to display the content in the assigned element.

The ReportExplorer class requires the use of a pre-existing actuate.RequestOptions object loaded with initialize. To use the default RequestOptions, use the RequestOptions constructor and provide the object as a parameter to the initialize call, as shown in the following code:

```
requestOpts = new actuate.RequestOptions( );
actuate.initialize( "http://127.0.0.1:8080/iportal", requestOpts,
   null, null, runReportExplorer);
```

Displaying ReportExplorer

The actuate.ReportExplorer class is a GUI that displays repository contents. Create an instance of the actuate.ReportExplorer class using JavaScript, as shown in the following code:

```
var explorer = new actuate.ReportExplorer("explorerpane");
```

The "explorerpane" parameter is the name value for the element which holds the report explorer content. The page body must contain a element with the id explorerpane as shown in the following code:

```
<div id="explorerpane"></div>
```

Use setFolderName( ) to set the directory to display in the explorer, as shown in the following code:

```
explorer.setFolderName("/public");
```

SetFolderName( ) accepts a single parameter, which is the path and name of a directory in the repository. In this example, "/public" indicates the /public directory.

ReportExplorer requires a results definition in order to retrieve data from the repository. The setResultDef( ) accepts an array of strings to define the results definition, as shown in the following code:

```
var resultDef = "Name|FileType|Version|VersionName|Description";
explorer.setResultDef( resultDef.split("|") );
```

The valid string values for the results definition array are "Name", "FileType", "Version", "VersionName", "Description", "Timestamp", "Size", and "PageCount", which correspond to file attributes loaded by ReportExplorer as it displays repository contents.

Call reportexplorer.submit( ) to make the page display the report explorer, as shown in the following code:

```
explorer.submit( );
```

The submit( ) function submits all the asynchronous operations that previous ReportExplorer functions prepare and triggers an AJAX request for the file information. The Actuate web application returns the list according to the results definition and the page displays the report explorer in the assigned element.

This is a complete example of constructing actuate.ReportExplorer( ) in a callback function to display repository contents:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
   <meta http-equiv="content-type" content="text
      /html;charset=utf-8" />
   <title>Report Explorer Page</title>
</head>
<body onload="init( )">
<div id="explorerpane">
   <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8080/iportal/jsapi"></script>

   <script type="text/javascript" language="JavaScript">

   function init( ) {
      actuate.load("reportexplorer");
      var reqOps = new actuate.RequestOptions( );
      actuate.initialize( "http://127.0.0.1:8080/iportal", reqOps,
         null, null, runReportExplorer);
   }

   function runReportExplorer( ) {
      var explorer = new actuate.ReportExplorer("explorerpane");
      explorer.setFolderName( "/Public" );
      var resultDef =
         "Name|FileType|Version|VersionName|Description";
      explorer.setResultDef( resultDef.split("|") );
      explorer.submit( );
   }
```

```
   </script>
</div>
</body>
</html>
```

The report explorer component displays the contents of the set folder, as shown in Figure 11-2.

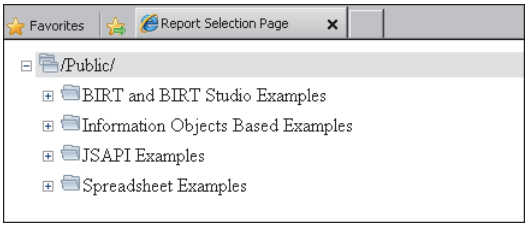


Figure 11-2 Report Explorer page

Use the mouse or arrow keys to navigate the repository tree and expand folders to view their contents.

Opening files from ReportExplorer

The ReportExplorer class generates an actuate.reportexplorer.eventconstants.ON_SELECTION_CHANGED event when the user selects a folder or file in the Report Explorer User Interface. To access the file information in this event, implement an event handler like the one shown in the following code:

```
var file;
...
explorer.registerEventHandler(
   actuate.reportexplorer.EventConstants.ON_SELECTION_CHANGED,
   selectionChanged );
...
function selectionChanged( selectedItem, pathName ){
   file = pathName;
}
```

The event passes the path and name of the file in the second parameter of the handler, pathName. To access the file, the event handler stores the path in a global variable, file.

In this implementation, the file path is updated each time a file selected. To open the file currently selected, implement a button on the page that runs a separate function that opens the file. The following code example shows a button that calls

the custom displayReport( ) function, which attempts to open the file using an actuate.viewer object:

```
<input type="button" style="width: 150pt;" value="View Report"
   onclick="javascript:displayReport( )"/>
...
function displayReport( ){
   var viewer = new actuate.Viewer("explorerpane");
   try {
      viewer.setReportName(file);
      viewer.submit( );
   } catch (e) {
      alert("Selected file is not viewable: " + file);
      runReportExplorer( );
   }
}
```

The try-catch block returns to the report explorer if Viewer is unable to open the file.

This is a complete example of a ReportExplorer page that opens a file in the BIRT Viewer when the user activates the View Report button:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
   <meta http-equiv="content-type" content="text
      /html;charset=utf-8" />
   <title>Report Explorer Page</title>
</head>
<body onload="init( )">
<input type="button" style="width: 150pt;" value="View Report"
   onclick="javascript:displayReport( )"/>
<hr />
<div id="explorerpane">
   <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8080/iportal/jsapi"></script>

   <script type="text/javascript" language="JavaScript">

   var file = "unknown";

   function init( ) {
      actuate.load("reportexplorer");
      actuate.load("viewer");
      actuate.load("dialog");
      var reqOps = new actuate.RequestOptions( );
```

```
      actuate.initialize( "http://127.0.0.1:8080/iportal", reqOps,
        null, null, runReportExplorer);
   }

   function runReportExplorer( ) {
      var explorer = new actuate.ReportExplorer("explorerpane");
      explorer.registerEventHandler( actuate.reportexplorer.
        EventConstants.ON_SELECTION_CHANGED, selectionChanged );
      explorer.setFolderName( "/Public" );
      var resultDef =
        "Name|FileType|Version|VersionName|Description";
      explorer.setResultDef( resultDef.split("|") );
      explorer.submit( );
   }

   function selectionChanged( selectedItem, pathName ){
      file = pathName;
   }

   function displayReport( ){
      var y = document.getElementById('explorerpane'), child;
      while(child=y.firstChild){
        y.removeChild(child);
      }
      var viewer = new actuate.Viewer("explorerpane");
      try {
        viewer.setReportName(file);
        viewer.submit( );
      } catch (e) {
        alert("Selected file is not viewable: " + file);
        runReportExplorer( );
      }
   }
   </script>
</div>
</body>
</html>
```

Using and submitting report parameters

Use the actuate.Viewer class to run report design and executable files. When a report design or executable runs, actuate.Viewer accepts parameters that modify the report output.

The actuate.Parameter class handles parameters and parameter values. The actuate.Parameter class enables a web page to display and gather parameters

from users before processing and downloading a report to the client. Load the actuate.Parameter class with actuate.load( ), as shown in the following code:

```
actuate.load("parameter");
```

Load the parameter component to use it later in the page. Call actuate.Parameters functions to prepare a parameters page, display the parameters in the assigned element, and assign the parameters to the viewer object for processing.

Using a parameter component

The actuate.Parameter class is a container for Actuate report parameters. Create an instance of the actuate.Parameter class using JavaScript, as shown in the following code:

```
var myParameters = new actuate.Parameter( "param1" );
```

The value of the "param1" parameter is the name value for the element that holds the report parameters display. The page body must contain a element with the param1 id, as shown in the following code:

```
<div id="param1"></div>
```

Use setReportName( ) to set the report from which to retrieve parameters, as shown in the following code:

```
myParameters.setReportName("/public/customerlist.rptdesign");
```

The setReportName( ) function takes the path and name of a report file in the repository as the only parameter. In this example, "/public /customerlist.rptdesign" indicates the Customer List report design in the /public directory.

To download the parameters and display them in a form on the page, call parameter.submit( ), as shown in the following code:

```
myParameters.submit(processParameters);
```

The submit( ) function submits all of the asynchronous operations prepared by the calls to parameter functions. The submit function also triggers an AJAX request to download the report parameters to the client. The Actuate web application sends the requested report parameters and the page displays them as a form in the assigned element. The submit( ) function takes a callback function as a parameter, shown above as processParameters.

The following code example calls parameter in the callback function for actuate.initialize( ) to display a parameter:

```
<div id="param1">
   <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8080/iportal/jsapi"></script>
```

```
   <script type="text/javascript" language="JavaScript">
   function init( ){
      actuate.load("viewer");
      actuate.load("parameter");
      var reqOps = new actuate.RequestOptions( );
      actuate.initialize( "http://127.0.0.1:8080/iportal", reqOps,

         null, null, displayParams);
   }
   function displayParams( ) {
      param = new actuate.Parameter("param1");
      param.setReportName("/Public/BIRT and BIRT Studio
         Examples/Customer Order History.rptdesign");
      param.submit(function ( ) { this.run.style.visibility=
         'visible';});
   }function processParameters( ) {
      …
   }
</script></div>
```

The parameter component displays all of the parameters of the report in a form. When the parameters page is larger than the size of the viewer, the viewer provides scroll bars to navigate the parameters page.

To retrieve the parameters, use actuate.Parameter.downloadParameterValues( ). This function takes a callback function as an input parameter. The callback function processes the parameter values, as shown in the following code:

```
function processParameters( ) {
   myParameters.downloadParameterValues(runReport);
}
```

The downloadParameterValues( ) function requires the callback function to accept an array of parameter name and value pairs. The API formats this array properly for the actuate.Viewer class.

Accessing parameter values from the viewer

The actuate.Viewer.setParameterValues( ) function adds the parameters set by the user to the viewer component. The setParameterValues( ) function takes as an input parameter an object composed of variables whose names correspond to parameter names. The downloadParameterValues( ) function returns a properly formatted object for use with actuate.Viewer.setParameterValues( ). The following code example shows how to call downloadParameterValues( ) and move the parameter name and value array into the viewer with actuate.Viewer.setParameterValues( ):

```
function runReport(ParameterValues){
   var viewer = new actuate.Viewer("viewerpane");
   viewer.setReportName("/Public/BIRT and BIRT Studio
      Examples/Customer Order History.rptdesign");
   viewer.setParameterValues(ParameterValues);
   viewer.submit( );
}
```

When the viewer calls submit( ), the client transfers the parameters to the server with the other asynchronous operations for the viewer.

The following code example shows a custom web page that displays parameters and then shows the report in a viewer using those parameters:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
   <meta http-equiv="content-type" content="text/
      html;charset=utf-8" />
   <title>Viewer With Parameters Page</title>
</head>
<body onload="init( )">
   <div id="parampane">
   <script type="text/javascript" language="JavaScript"
      src="http://127.0.0.1:8080/iportal/jsapi"></script>
   <script type="text/javascript" language="JavaScript">
   function init( ){
      actuate.load("viewer");
      actuate.load("parameter");
      var reqOps = new actuate.RequestOptions( );
      actuate.initialize( "http://127.0.0.1:8080/iportal", reqOps,

         null, null, displayParams);
   }
   function displayParams( ) {
      param = new actuate.Parameter("parampane");
      param.setReportName("/Public/BIRT and BIRT Studio
         Examples/Customer Order History.rptdesign");
      param.submit(
            function ( ) {this.run.style.visibility = 'visible';});
   }
   function processParameters( ) {
      param.downloadParameterValues(runReport);
   }
```

```
   </script>
   </div>
   <hr><br />
   <input type="button" class="btn" name="run"
      value="Run Report" onclick="processParameters( )"
      style="visibility: hidden">

   <div id="viewerpane">
   <script type="text/javascript" language="JavaScript"
   src="http://127.0.0.1:8080/iportal/jsapi"></script>
   <script type="text/javascript" language="JavaScript">
   function runReport(paramvalues) {
      var viewer = new actuate.Viewer("viewerpane");
      viewer.setReportName("/Public/BIRT and BIRT Studio
         Examples/Customer Order History.rptdesign");
      viewer.setParameterValues(paramvalues);
      viewer.submit( );
   }
   </script>
   </div>
</body>
</html>
```

The code in the example uses the administrator user credentials and the default report installed with a standard installation of Java Components. The default report is at the following path:

```
/Public/BIRT and BIRT Studio Examples/Customer Order
   History.rptdesign
```

Retrieving report content as data

To retrieve report content as data, use the actuate.DataService class from the Actuate JavaScript API. The DataService is packaged with the actuate.Viewer class. Load the actuate.DataService class with actuate.load( ), as shown in the following code:

```
actuate.load("viewer");
```

Load support for dialog boxes from the Actuate JavaScript API with actuate.load( ), as shown in the following code:

```
actuate.load("dialog");
```

Load the viewer and dialog components to use data services on the page. Call the functions in the actuate.DataService class to prepare report data, then call downloadResultSet( ) from the DataService class to obtain the report data.

Using a data service component

The actuate.DataService class is a container for Actuate report data. Create an instance of the class with JavaScript, as shown in the following code:

```
var dataservice = new actuate.DataService( );
```

Without parameters, the actuate.DataService class uses the Actuate web application service called in actuate.intialize.

To gather data from a report, define a request and send the request to the Actuate web application service for the data. The actuate.data.Request object defines a request. To construct the Request object, use the actuate.data.Request constructor, as shown below:

```
var request = new actuate.data.Request(bookmark, start, end);
```

where

- bookmark is a bookmark that identifies an Actuate report element. The actuate.data.Request object uses the bookmark to identify the report element from which to request information. If bookmark is null, the actuate.data.Request object uses the first bookmark in the report.
- start is the numerical index of the first row to request. The smallest valid value is 1.
- end is the numerical index of the last row to request. A value of 0 indicates all available rows.

To download the data, use dataservice.downloadResultSet( ), as shown in the following code:

```
dataservice.downloadResultSet(filedatasource, request,
   displayData, processError);
```

where

- filedatasource is the path and name of a report file in the repository. For example, "/public/customerlist.rptdesign" indicates the Customer List report design in the /public directory. The dataservice.downloadResultSet( ) function uses the Actuate web application service set with actuate.Initialize( ) by default.
- request is an actuate.data.Request object that contains the details that are sent to the server in order to obtain specific report data.
- displayData is a callback function to perform an action with the downloaded data. This callback function takes an actuate.data.ResultSet object as an input parameter.
- processError is a callback function to use when an exception occurs. This callback function takes an actuate.Exception object as an input parameter.

JSAPI DataService cannot download ResultSets from BIRT report elements with an automatically generated bookmark. When designing a report, report developers can explicitly specify bookmarks for report elements. If a bookmark is not specified, the report generates a generic bookmark name automatically when it executes. The JSAPI DataService class cannot retrieve a result set from these generic bookmarks. To use the JSAPI DataService on a bookmark, the report developer must specify a name value for the bookmark.

To provide a quick alert displaying the column headers for the retrieved data set, use code similar to the following:

```
alert("Column Headers: " + myResultSet.getColumnNames());
```

where myResultSet is the ResultSet object retrieved by downloadResultSet.

Using a result set component

The actuate.data.ResultSet class is the container for the report data obtained with actuate.dataservice.downloadResultSet( ). Because a ResultSet object is not a display element, an application can process or display the data in an arbitrary fashion.

The ResultSet class organizes report data into columns and rows, and maintains an internal address for the current row. To increment through the rows, use the ResultSet's next( ) function as shown in the following code:

```
function displayData(rs)
{
…
   while (rs.next( ))
…
```

In this example, rs is the ResultSet object passed to the displayData callback function. To read the contents of the ResultSet object, a while loop increments through the rows of data with rs.next( ).

Because a web page that loads a DataService object also loads initiates the viewer, the target for displaying a result set must be a separate page or application.

Controlling Interactive Viewer user interface features

The BIRT Interactive Viewer enables users to perform a number of custom operations on a BIRT design or document and save or print changes as a new design or document. The file and print features for Interactive Viewer are available in the main menu of the BIRT viewer, as shown in Figure 11-3.

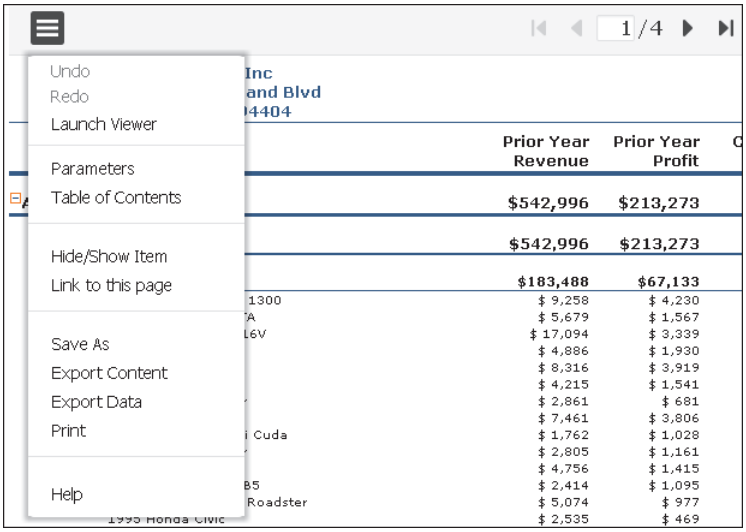


Figure 11-3 Interactive Viewer menu

The actuate.viewer.UIOptions class can enable or disable any of the interactive features, including enabling and disabling interactivity. This restricts access to features that aren't useful to the user or that aren't supported by the Actuate Java Components application.

All actuate.viewer.UIOptions enable functions accept a Boolean input parameter, which if true enables an interactive feature and if false disables an interactive feature. To display the list of currently enabled and disabled features, use actuate.viewer.UIOptions.getFeatureMap( ). Table 11-1 contains a complete list of actuate.viewer.UIOptions enable functions that control features.

Table 11-1 UIOptions enable feature functions

Function	Interactive feature
enableAdvancedSort( )	Enables the advanced sort feature
enableAggregation( )	Enables the aggregation feature
enableCalculatedColumn( )	Enables the calculated column feature
enableChartProperty( )	Enables the chart properties feature.
enableChartSubType( )	Enables the chart subtype selection feature
enableCollapseExpand( )	Enables the collapse/expand feature
enableColumnEdit( )	Enables the column editing feature
enableColumnResize( )	Enables the column resizing feature
enableContentMargin( )	Enables the content margin feature
enableDataAnalyzer( )	Enables the Launch Interactive Crosstab feature

Table 11-1 UIOptions enable feature functions (continued)

Function	Interactive feature
enableDataExtraction( )	Enables the data extraction feature
enableEditReport( )	Enables the report editing/interactivity feature
enableExportReport( )	Enables the export report feature
enableFilter( )	Enables the filter feature
enableFlashGadgetType( )	Enables the flash gadget type change feature
enableFormat( )	Enables the format editing feature
enableGroupEdit( )	Enables the group editing feature
enableHideShowItems( )	Enables the hide/show item feature
enableHighlight( )	Enables the highlight feature
enableHoverHighlight( )	Enables the hover highlight feature
enableLaunchViewer( )	Enables the Launch Viewer feature
enableLinkToThisPage( )	Enables the "link to this page" feature
enableMainMenu( )	Enables the main menu feature
enableMoveColumn( )	Enables the column moving feature
enablePageBreak( )	Enables the page break editing feature
enablePageNavigation( )	Enables the page navigation feature
enableParameterPage( )	Enables the parameter page feature
enablePrint( )	Enables the print feature
enableReorderColumns( )	Enables the column reordering feature
enableRowResize( )	Enables the row resizing feature
enableSaveDesign( )	Enables the report design save feature
enableSaveDocument( )	Enables the report document save feature
enableServerPrint( )	Enables the server-side printing feature
enableShowToolTip( )	Enables the show tooltip feature
enableSort( )	Enables the sort feature
enableSuppressDuplicate( )	Enables the duplication suppression feature
enableSwitchView( )	Enables the switch view feature
enableTextEdit( )	Enables the text editing feature
enableTOC( )	Enables the table of contents feature
enableToolBar( )	Enables the toolbar feature

(continues)

Table 11-1 UIOptions enable feature functions (continued)

Function	Interactive feature
enableToolbarContextMenu( )	Enables the show toolbar features in a context menu
enableToolbarHelp( )	Enables the toolbar help feature
enableTopBottomNFilter( )	Enables the top N and bottom N filter feature
enableUndoRedo( )	Enables the undo and redo feature

The viewer does not accept UIConfig changes after it loads. The only way to reset UIConfig options is to reload the viewer. This is only viable in the context of a web page, as the viewer must always be present for a BIRT design to run scripts.

Disabling UI features in a custom web page

Custom web pages can restrict the viewer's user interface using the actuate.viewer.UIOptions class. For example, if you wanted to create a viewer page that didn't provide access to parameters, create a UIOptions object that disables the parameters page on the display as shown in the following code:

```
var manUIOptions = new actuate.viewer.UIOptions( );
manUIOptions.enableParameterPage(false);
```

To apply this UIConfig settings to the viewer, use the UIConfig object in the viewer's constructor, as shown in the following code:

```
var manViewer = new actuate.Viewer(ManContainer);
manViewer.setUIOptions(manUIOptions);
manViewer.submit();
```

The viewer configured with the parameter page feature disabled does not show the Parameters option in the main menu, as shown in Figure 11-4.

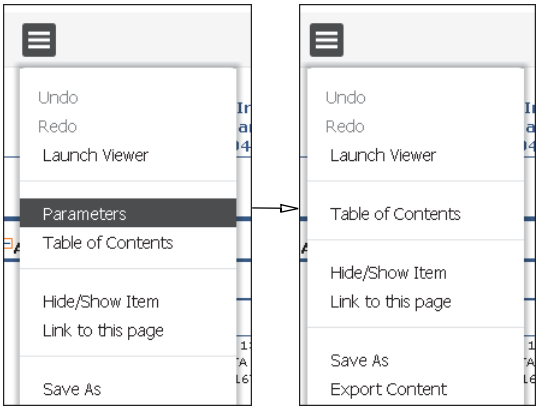


Figure 11-4 The main menu with edit disabled

Control the viewer interface features with the actuate.viewer.UIOptions class. You create an instance of this class using JavaScript, as shown in the following code:

```
var uioptions = new actuate.viewer.UIOptions( );
```

Set the user interface options with the enable functions in the actuate.viewer.UIOptions class. For example, a toolbar appears in the viewer by default, as shown in Figure 11-1.

Figure 11-5 The default toolbar for the JavaScript API viewer

To disable this toolbar, use the following code:

```
uioptions.enableToolBar(false);
```

All of the enable functions take a Boolean value as an argument. To configure the viewer to use these options, use setUIOptions( ) as shown in the following code:

```
viewer.setUIOptions(uioptions);
```

The setUIOptions( ) function accepts one parameter: an actuate.viewer.UIOptions object. The viewer's submit( ) function commits the user interface changes to the viewer when the function sends the object to the HTML container. Set the UI options using setUIOptions( ) before you call submit( ).

Chapter

11

Creating dynamic report content using the Actuate JavaScript API

This chapter contains:

- About Actuate JavaScript API scripting in a BIRT report design
- Using the Actuate JavaScript API in an HTML button
- Using the Actuate JavaScript API in chart interactive features
- Using the Actuate JavaScript API in chart themes

About Actuate JavaScript API scripting in a BIRT report design

The scripting features of the BIRT designers support using the JSAPI for the following operations:

- Using the Actuate JavaScript API in an HTML button
- Using the Actuate JavaScript API in chart interactive features
- Using the Actuate JavaScript API in chart themes

Most Actuate JavaScript API functions run when an event occurs. The report element defines the events that it supports. For example, the onRender event occurs when the report renders in the viewer or on a page.

A BIRT report or Reportlet renders in the following ways:

- In BIRT Viewer or Interactive Viewer
- In BIRT Studio
- In Actuate BIRT Designer
- In an Actuate JavaScript API viewer object on a mashup page

All of these products load the actuate.Viewer and actuate.Dialog classes when they render a report, except for the preview functionality in BIRT Designer. Use the View Report in Web Viewer function to view and test Actuate JavaScript API scripts with BIRT Designer, as shown in Figure 12-1.

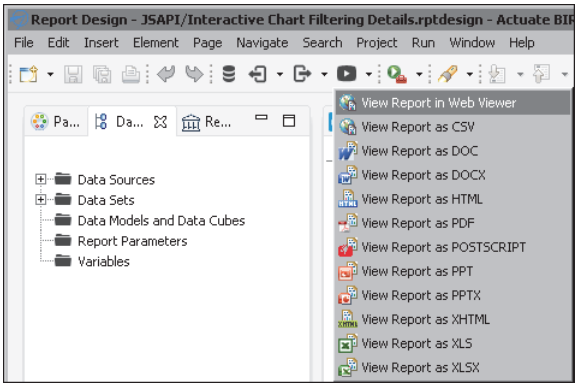


Figure 12-1 Accessing Web Viewer in Actuate BIRT Designer

Most of the classes and functions in the actuate.Viewer class can be used in a BIRT report design without loading or initializing the actuate.Viewer class and. Most of the viewers also load the actuate.Parameters and actuate.DataService classes by default. Define the classes loaded for Actuate JavaScript API mashup page

explicitly. Load the DataService, Parameters, and Viewer classes before the API initializes the connection to the reporting web service.

Using the Actuate JavaScript API in an HTML button

The HTML button element can execute client-side JavaScript code based on button events. Access the HTML button in the BIRT designer by selecting a button element, choosing the script tag, and selecting the event from the event drop-down list, as shown in Figure 12-2.

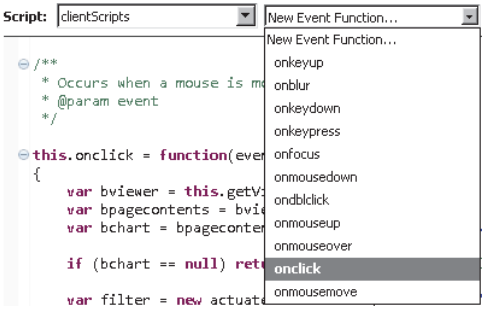


Figure 12-2 Choosing HTML button event handlers

Use event functions to add JavaScript functionality to HTML buttons. For example, a button that swaps columns of data, filters data, sorts data, hides information, or groups the rows of a table by some data column can be created with event functions. The following script groups the rows of a table by the quantity of items in each order when the HTML button is clicked:

```
this.onclick = function(event){
   var btable = this.getViewer( ).getCurrentPageContent( ).
      getTableByBookmark("TableBookmark");
   btable.groupBy("QUANTITYORDERED");
   btable.submit( );
}
```

When the HTML button triggers the example event above, the table grouping changes and the display refreshes, as shown in Figure 12-3.

HTML buttons can be arranged into sets of controls for the user to use once a report runs. For example, when these buttons are used in the header for a table, the header can provide controls similar to those in the header shown in Figure 12-4.

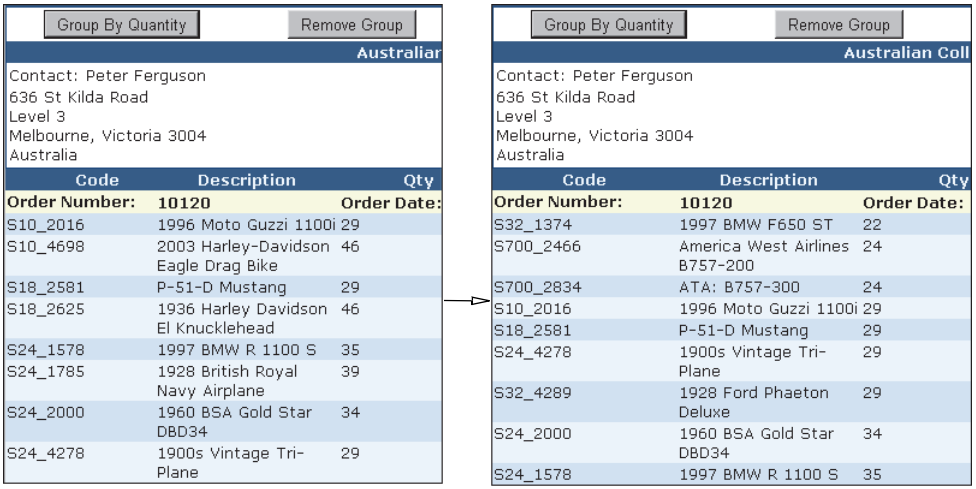


Figure 12-3 Using a GroupBy HTMLButton control

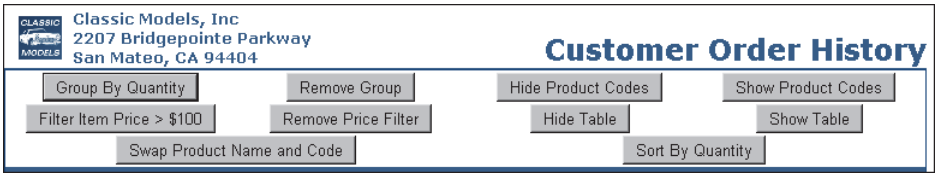


Figure 12-4 HTML button header

Using the Actuate JavaScript API in chart interactive features

BIRT reports support adding interactive features to a chart to enhance the behavior of a chart in the viewer. The interactive chart features are available through the chart builder. Implement Actuate JavaScript API functions within interactive features.

An interactive chart feature supports a response to an event, such as the report user choosing an item or moving the mouse pointer over an item. The response can trigger an action, such as opening a web page, drilling to a detail report, or changing the appearance of the chart. For example, use a tooltip to display the series total when a user places the mouse over a bar in a bar chart, as shown in Figure 12-5.

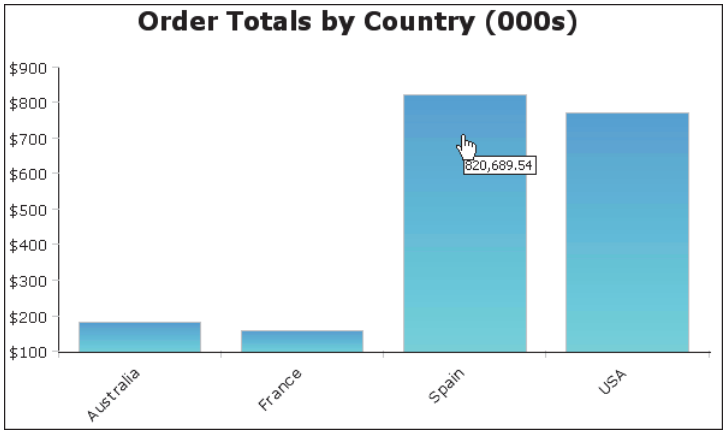


Figure 12-5 Chart showing a tooltip

Interactive features can be added to a value series, the chart area, a legend, marker lines, the *x*- and *y*-axis, or a title. Figure 12-6 identifies these elements.

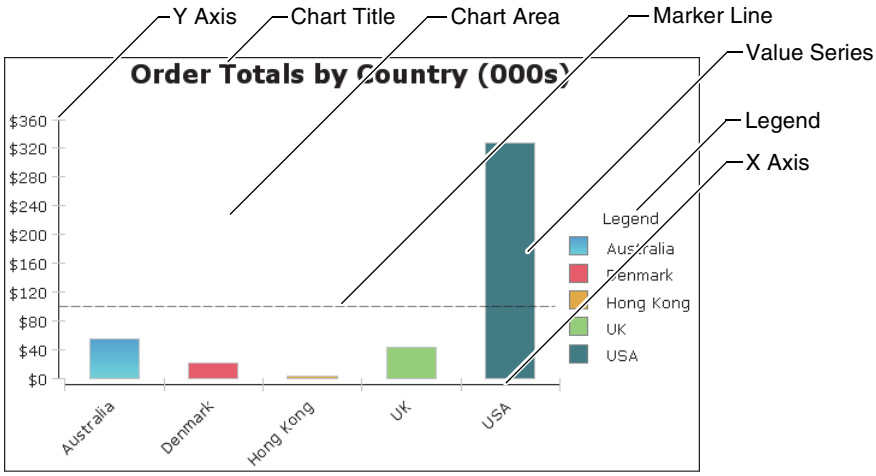


Figure 12-6 Elements selectable for chart interactivity

To add an interactive feature to a chart, either choose Format Chart in the chart builder and select a chart element to make interactive, or choose Script in the chart builder and select the chart element to make interactive. Figure 12-7 shows the location of the Interactivity button for a value series.

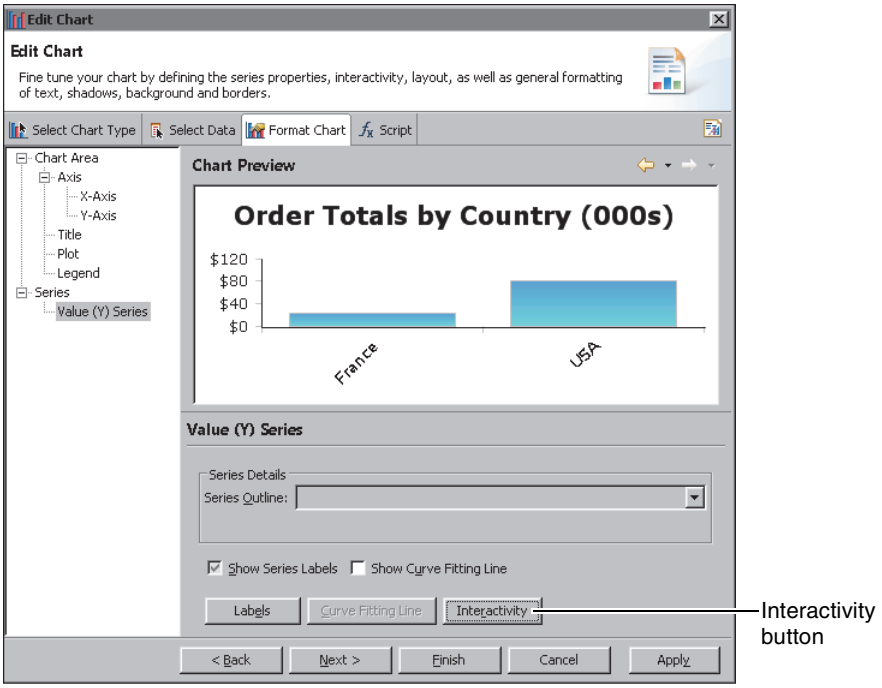


Figure 12-7 Accessing interactivity for a value series

Figure 12-8 shows the elements accessible using the script feature.

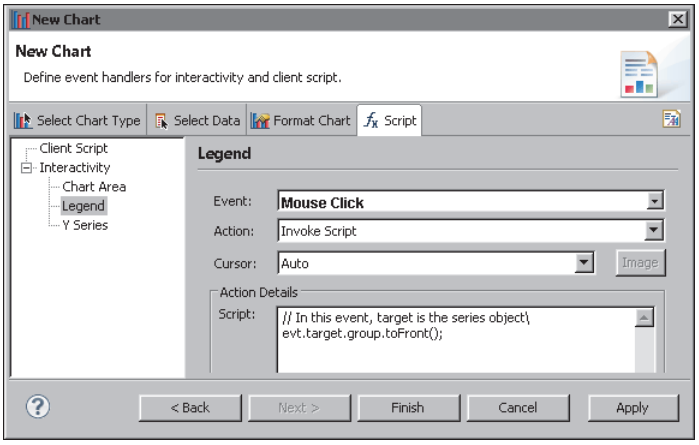


Figure 12-8 Accessing interactivity for a legend

The location of the Interactivity button varies by chart element. Click the Interactivity button to display the interactivity editor. Figure 12-9 shows the interactivity editor.

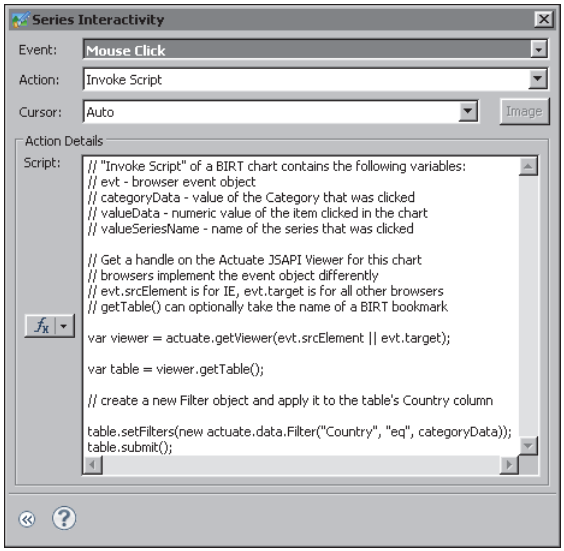


Figure 12-9 Interactivity editor

The Action Details window displays a script that runs when the user clicks an item in the series. The script adds a filter to the table that displays below the chart. The filter restricts the data by the selected element. The code performs the following three tasks to handle this interactivity:

- Obtains the bookmark for the table when the event occurs:

  ```
  var viewer = actuate.getViewer(evt.srcElement ||
     evt.originalTarget)
  var table = viewer.getTable( );
  ```

  The event is taken from the Invoke Script action of a BIRT chart. Set the Invoke Script action in the second field of the interactivity editor. The Invoke Script action contains the following variables:

  - evt: browser event object
  - categoryData: value of the selected category
  - valueData: numeric value of the selected item
  - valueSeriesName: name of the selected series

  The code above uses getViewer and the evt object to obtain a handle for the viewer when an event occurs. The Firefox and Internet Explorer browsers implement the event differently. For Firefox, evt.originalTarget contains the name of the viewer object. For Internet Explorer, evt.srcElement contains the name of the viewer object.

The getTable( ) function retrieves the Table object for the first table in the viewer. To target a different table, use a specific table bookmark as the input parameter for getTableByBookmark( ).

- Performs an operation on the target:

```
table.setFilters(new actuate.data.Filter("Country", "eq",
   categoryData));
```

This code example creates a new filter using the actuate.data.Filter constructor. The constructor takes three arguments:

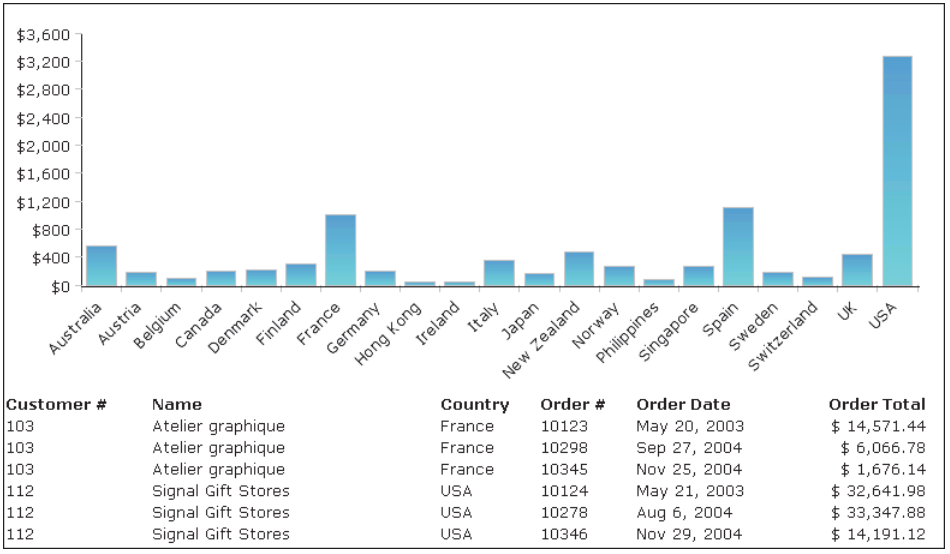
  - column name: The column name is the name of the series. In this case, the *y*-axis is a list of countries, so a mouse click filters the table according to the Country column.
  - operator: eq is the reserved operator for equal to.
  - value: the value of the categoryData object generated by the event, which is a country. The filter returns rows with a Country value that matches the value selected by the user.

- Submits the action for processing:

```
table.submit( );
```

The Actuate JavaScript API processes operations asynchronously. Actions are performed when submit( ) is called.

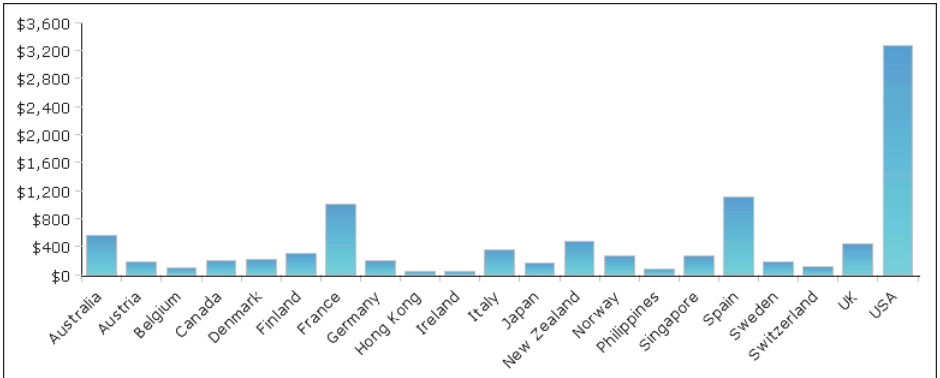
Figure 12-10 shows the chart before interaction.



Customer #	Name	Country	Order #	Order Date	Order Total
103	Atelier graphique	France	10123	May 20, 2003	$ 14,571.44
103	Atelier graphique	France	10298	Sep 27, 2004	$ 6,066.78
103	Atelier graphique	France	10345	Nov 25, 2004	$ 1,676.14
112	Signal Gift Stores	USA	10124	May 21, 2003	$ 32,641.98
112	Signal Gift Stores	USA	10278	Aug 6, 2004	$ 33,347.88
112	Signal Gift Stores	USA	10346	Nov 29, 2004	$ 14,191.12

Figure 12-10 An interactive chart and table before any user action

When the user selects the bar for Australia in the value series, the table is filtered for Australia, as shown in Figure 12-11.



Customer #	Name	Country	Order #	Order Date	Order Total
114	Australian Collectors, Co.	Australia	10120	Apr 29, 2003	$ 45,864.03
114	Australian Collectors, Co.	Australia	10125	May 21, 2003	$ 7,565.08
114	Australian Collectors, Co.	Australia	10223	Feb 20, 2004	$ 44,894.74
114	Australian Collectors, Co.	Australia	10342	Nov 24, 2004	$ 40,265.60
114	Australian Collectors, Co.	Australia	10347	Nov 29, 2004	$ 41,995.62
276	Anna's Decorations, Ltd	Australia	10148	Sep 11, 2003	$ 41,554.73

Figure 12-11 An interactive chart and table after the user selects Australia

Using the Actuate JavaScript API in chart themes

BIRT reports support adding themes to a chart to apply common elements to similar charts. Access chart themes by exporting and then editing a theme or by creating a new theme. Implement Actuate JavaScript API functions within specific theme elements or in the script feature of the theme.

A chart theme supports executing a script before or after certain events, such as before rendering the chart. For example, you can add scripts for beforeGeneration, beforeRendering, beforeDrawAxis, beforeDrawSeries, beforeDrawDataPoint, and afterRendering when editing a chart theme, as shown in Figure 12-12.

In an HTML5 chart, you can use the actuate.report.HTML5Chart classes to alter the report display. For example, to render every data point in the series that is greater than avgValue in a green color, use code similar to the following:

```
beforeDrawSeries: function(series, seriesOptions, tempChart,
   seriesIndex){
   for ( var i = 0; i < series.data.length; i++ ){
      // Find out if this data point is above average
      if ( series.data[i].y <= aveValue ){
```

```
        // The data point is above average.  Color it green
        var pointOptions = seriesOptions.data[i];
        pointOptions.color = 'green';
    }
}
```

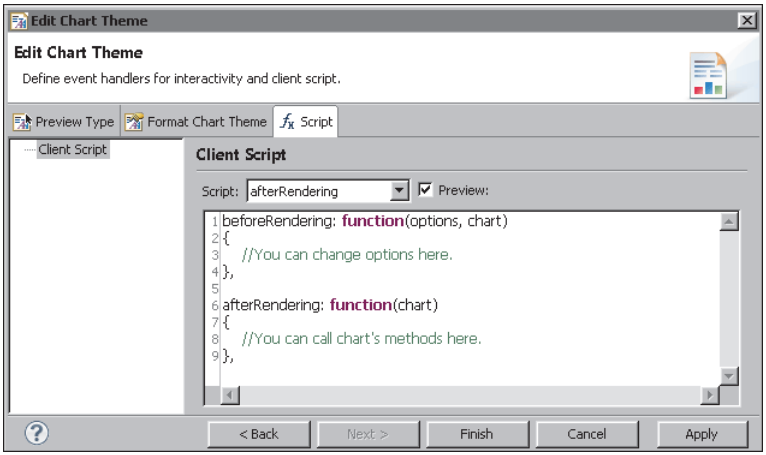


Figure 12-12 Adding script elements in edit chart theme

Chapter

12

Working with Interactive Crosstabs

This chapter contains:

- About cross tabs
- About cubes
- Handling Interactive Crosstabs viewer events
- Working with dimensions, measures, and levels
- Working with totals
- Sorting and filtering cross tab data
- Drilling down within a cross tab
- Controlling the Interactive Crosstabs viewer user interface

About cross tabs

A cross tab, or cross tabulation, displays data in a row-and-column matrix similar to a spreadsheet. A cross tab is ideal for concisely summarizing data. A cross tab displays aggregate values such as averages, counts, or sums in the cross tab's cells.

Figure 13-1 shows a cross tab that organizes state groups in the row area and product line groups in the column area. Aggregate revenue values appear in the cells of the data area.

Row area displays state groups

Column area displays product line groups

	Classic Cars	Motorcycles	Planes	Ships	Trains	Grand Total
	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue
CA	$401,126	$162,711	$108,632	$66,759	$17,965	$757,194
CT	$89,671	$39,700	$41,142	$5,937	$9,549	$185,998
MA	$217,769	$91,024	$51,925	$48,333	$8,070	$417,121
NH	$69,150					$69,150
NJ		$31,103		$4,346		$35,449
NV	$58,719					$58,719
NY	$258,090	$99,515	$24,648	$13,782	$11,010	$407,045
PA	$102,856	$39,025	$15,890	$4,983	$4,862	$167,617
Grand Total	$1,197,382	$463,077	$242,237	$144,141	$51,456	$2,098,293

Data area displays aggregate revenue values

Figure 13-1 Viewing a cross tab

A cell displays a revenue value by product line and by state, as shown in Figure 13-2.

	Classic Cars	Motorcycles
	Revenue	Revenue
CA	$401,126	$162,711
CT	$89,671	$39,700
MA	$217,769	$91,024
NH	$69,150	
NJ		$31,103

The revenue total for Classic Cars for New Hampshire

Figure 13-2 A cell displaying a revenue total

A cross tab uses data from at least three fields. The cross tab in Figure 13-1 uses the following data fields:

- One field provides the values for column headings in the cross tab. The cross tab displays one column for each unique value in the field. In Figure 13-1, the cross tab displays five unique values from the productline field: Classic Cars, Motorcycles, Planes, Ships, and Trains.
- One field provides the values for row headings in the cross tab. The cross tab displays one row for each unique value in the field. In Figure 13-1, the cross tab displays eight unique values from the state field: CA, CT, MA, NH, NJ, NV, NY, and PA.

- Interactive Crosstabs aggregates one field's values, and displays these values in the cross tab cells. In this example, each cell displays a revenue total by product line and state. Interactive Crosstabs calculates the revenue total using the SUM function on the values in the extendedprice field.

About cubes

A cube is a multidimensional data structure that is optimized for analysis. A cube supports applications that perform complex analyses without performing additional queries on the underlying data source. A cube organizes data into the following categories:

- Measures

  Measures are aggregate, or summary, values, such as sales revenues or units of products.

- Dimensions

  Dimensions are groups, such as customers, product lines, or time periods, which aggregate measures. For example, a sales revenue cube contains data that enables viewing sales volume and revenues, both of which are measures, by customers, product lines, and time periods, all of which are dimensions.

Dimensions can contain levels, which organize data into hierarchies. For example, a region dimension can contain a hierarchy of the country, state, and city levels. A time dimension can contain a hierarchy of the year, quarter, month, and day levels. Cubes frequently include time dimensions because displaying measures by time dimensions is useful in data analysis. The time dimension in a cube is a special dimension that supports storing data in developer-defined time periods.

Use Actuate BIRT Designer Professional to create a cube using data from one or more data sources, then create a cross tab that uses the cube data and specifies the cross tab appearance. The initial cross tab that appears in Interactive Crosstabs typically displays a portion of the available cube data in a simple layout. Figure 13-3 shows a cross tab and all of the cube measures and dimensions that are available for analysis.

See *BIRT: A Field Guide* for more information about data cubes and cross tabs.

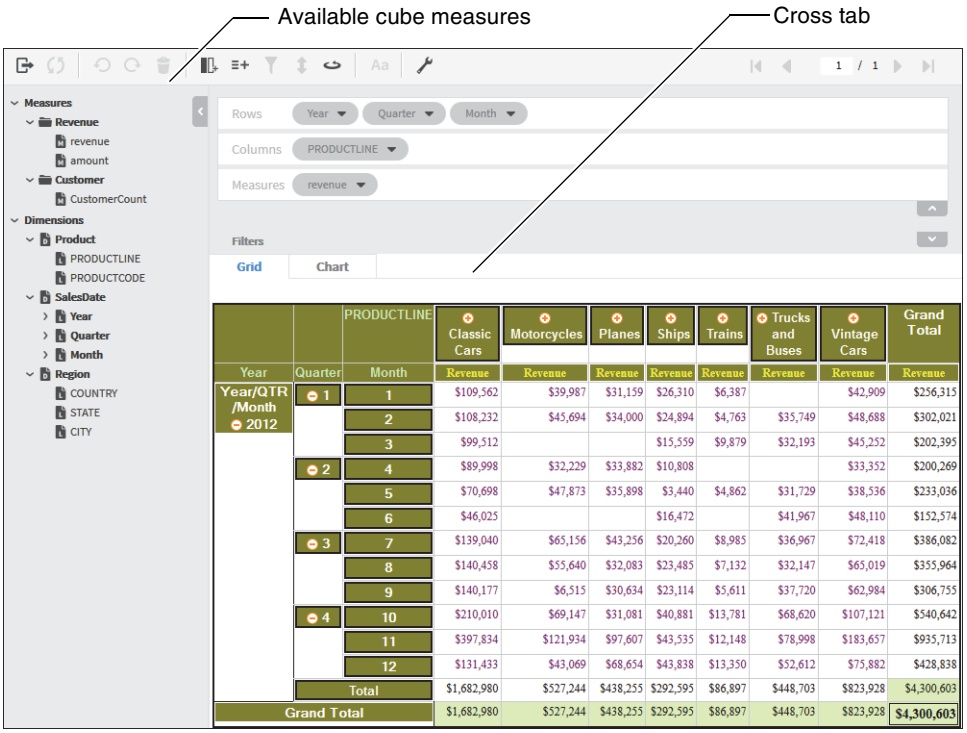


Figure 13-3 Interactive Crosstabs displaying a cross tab and available measures and dimensions

Handling Interactive Crosstabs viewer events

The Interactive Crosstabs viewer triggers events to indicate changes in status. These events include notifications of data changes or errors. Use the registerEventHandler function found in XTabAnalyzer to handle events, as shown in the following code:

```
ctViewer.registerEventHandler(actuate.xtabanalyzer.EventConstants.
                                ON_EXCEPTION,errorHandler);
```

This code registers the event handler errorHandler to be called when an ON_EXCEPTION event occurs.

The XTabAnalyzer class supports the following events:

- ON_CONTENT_CHANGED
- ON_CONTENT_SELECTED
- ON_EXCEPTION

- ON_SESSION_TIMEOUT

To remove an event handler, call removeEventHandler( ).

```
ctViewer.removeEventHandler(actuate.xtabanalyzer.EventConstants.
                            ON_EXCEPTION,errorHandler);
```

The actuate.xtabanalyzer.Exception class handles exceptions. For more information about events, see the section describing the actuate.xtabanalyzer.EventsConstants class.

Working with dimensions, measures, and levels

The actuate.xtabanalyzer.Crosstab class represents the cross tab element. Use this cross tab class when working with Interactive Crosstabs and the XTabAnalyzer viewer. Use the functions in the actuate.xtabanalyzer.Dimension class to add, remove, or modify dimensions. Use the functions in the actuate.xtabanalyzer.Measure class to add, remove, or modify measures. Use the functions in the actuate.xtabanalyzer.Level class to add, remove, or modify levels. These classes contain functions that support the creation and modification of the dimensions, measures, and levels in the cross tab. These functions work with information from a data cube that is created with BIRT Designer Professional.

Adding a dimension with levels

To add a dimension to the cross tab, use Crosstab.addDimension( ) to add an actuate.xtabanalyzer.Dimension object to the cross tab. The following code requires that the dimensions and levels already exist within a data cube:

```
var crosstab = new actuate.xtabanalyzer.Crosstab( );
var dimension = new actuate.xtabanalyzer.Dimension( );

// Set dimension to be in the zero location.
dimension.setIndex(0);
dimension.setAxisType(actuate.xtabanalyzer.Dimension.
                      COLUMN_AXIS_TYPE);
dimension.setDimensionName("dates");
var level = new actuate.xtabanalyzer.Level( );
level.setLevelName("year");
dimension.addLevel(level);
var level = new actuate.xtabanalyzer.Level( );
level.setLevelName("quarter");
dimension.addLevel(level);
var level = new actuate.xtabanalyzer.Level( );
level.setLevelName("month");
dimension.addLevel(level);
crosstab.addDimension(dimension);
crosstab.submit( );
```

Removing a dimension

To remove a dimension from a cross tab, use Crosstab.removeDimension( ). In this example, levelNames is an array of strings containing the names of the levels to remove:

```
crosstab.removeDimension("dates",null,levelNames);
crosstab.submit( );
```

Adding and removing measures

To add a measure to the cross tab, use Crosstab.addMeasure( ). The addMeasure( ) function accepts an actuate.xtabanalyzer.Measure object as a parameter. This example creates a new measure and adds it to a cross tab:

```
var measure = new actuate.xtabanalyzer.Measure( );

measure.setIndex(1);
measure.setMeasureName("Quarter Rate");
measure.setExpression("[revenue]/[revenue_SalesDate/year_Product
                        /PRODUCTLINE]");
crosstab.addMeasure(measure);
crosstab.submit( );
```

The measure.setExpression( ) function dynamically sets the measure to display the revenue received for sales data, organized by year and product line. In this example, the expression is in EasyScript. EasyScript is described in *Using Actuate BIRT Designer Professional*. The expression in the example is the database field that contains the sales revenue value. Interactive Crosstabs aggregates the sales revenue value for each year for each product line. The [revenue_SalesDate /year_Product/PRODUCTLINE] string specifies that the expression applies to the revenue by sales date and then by year for the product line.

The Actuate JavaScript API combined with standard JavaScript functionality enables the creation of web pages that allow for interactive manipulation of cross tabs. In this example, the measure name and the measure expression are retrieved from HTML elements with the names of measureName and measureExpression. As coded, these elements can be an item such as a text entry field. The values of any used elements then go into the new measure for the cross tab.

```
var measureName  = document.getElementById("measureName").value;
var measureExpression  =
   document.getElementById("measureExpression").value;

var measure = new actuate.xtabanalyzer.Measure( );
measure.setIndex(1);
measure.setMeasureName(measureName);
measure.setExpression(measureExpression);

crosstab.addMeasure(measure);
crosstab.submit( );
```

The web page must contain elements with the IDs of measureName and measureExpression. Use the following HTML code to create these elements:

```
<INPUT TYPE="text" SIZE="60" ID="measureName" VALUE="Quarter
   Rate">
<INPUT type="text" SIZE="60" ID="measureExpression"
   VALUE="[revenue]/[revenue_SalesDate/year_Product/PRODUCTLINE]">
```

Use removeMeasure( ) to remove a measure. Pass the name of the measure to remove to removeMeasure( ).

```
crosstab.removeMeasure("Quarter Rate");
crosstab.submit( );
```

Changing measures and dimensions

Edit measures with Crosstab.editMeasure( ). In this example, the measureName measure named measureName takes on a new value:

```
var measure = new actuate.xtabanalyzer.Measure( );

measure.setMeasureName("measureName");
measure.setExpression("measureExpression");
crosstab.editMeasure(measure);
crosstab.submit( );
```

Use Crosstab.changeMeasureDirection( ) to change the measure direction. Pivot the cross tab with Crosstab.pivot( ).

Use Crosstab.reorderDimension( ) to change the order or axis type of a dimension within a cross tab. This example moves the index of a dimension within a cross tab from 1 to 5. The dimension's axis type changes from a row axis to a column axis.

```
var dimIdx = 1;
var newDimIdx = 5
var axis = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
var newAxis = actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE;
crosstab.reorderDimension(dimIdx,axis,newDimIdx,newAxis );
crosstab.submit( );
```

The measure placement order can be altered using Crosstab.reorderMeasure( ). In this example, a measure's index changes from position 1 in the cross tab to position 5:

```
crosstab.reorderMeasure(1,5);
crosstab.submit( );
```

Measures and dimensions can also be changed with the functions in the measure and dimension classes. In this example, a dimension axis changes from column to row:

```
var currentAxis = dimension.getAxisType( )
if (currentAxis ==
   actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE){
   dimension.setNewAxisType(
      actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
}
```

Working with totals

Each dimension within a cross tab and each level within a multilevel dimension can have a total associated with that dimension or level. A row or column with a single dimension can only have a grand total. Each level in a multilevel dimension can have a subtotal. Subtotals are only available for multilevel dimensions.

A total requires a measure and an aggregation function. To add a grand total to a measure, use the actuate.xtabanalyzer.GrandTotal class. Subtotals are added with the actuate.xtabanalyzer.SubTotal class. Both classes use the actuate.xtabanalyzer.Total class. The Total class supports creating aggregated values on a measure, calculated on either a row or a column. This example creates a total and places the SUM aggregation function on the measure located at measure index 0:

```
var grandTotal = new actuate.xtabanalyzer.GrandTotal( );
grandTotal.setAxisType(actuate.xtabanalyzer.Dimension.
                       ROW_AXIS_TYPE );

// Create a total object containing a measure and aggregation.
var total = new actuate.xtabanalyzer.Total( );
total.setMeasureIndex(0);
total.setAggregationFunction("SUM");
total.setEnabled(true);

// Add the total to the cross tab.
grandTotal.addTotal(total);
crosstab.setTotals(grandTotal);
crosstab.submit( );
```

The actuate.xtabanalyzer.Total class uses a measure index and an aggregation function to create a Total object that is added to a SubTotal or GrandTotal object for placement within the cross tab. A total must be enabled for that total to be active on the cross tab.

To remove a total from a cross tab, use setEnabled( ) and pass false as a parameter, as shown in the following code:

```
total.setEnabled(false);
grandTotal.addTotal(total);
crosstab.setTotals(grandTotal);
crosstab.submit( );
```

Sorting and filtering cross tab data

Data within levels can be filtered and sorted. To sort data within a level, use the actuate.xtabanalyzer.Sorter class. Add an instance of the Sorter class to the cross tab with Crosstab.setSorters( ).

```
var sorter = new actuate.xtabanalyzer.Sorter("sortLevelName");
sorter.setAscending(false);

// Add the sort to the cross tab.
crosstab.setSorters(sorter);
crosstab.submit( );
```

Use the actuate.xtabanalyzer.Filter class to filter data within a level. A filter requires an operator and values to filter. Use Crosstab.setFilters( ) to place the filter within the cross tab.

```
var filter = new actuate.xtabanalyzer.Filter
                ("levelName","BETWEEN");

// Filter between the values of 1000 and 2000.
var filterValue = "1000;2000";
filter.setValues(filterValue.split(";"));
crosstab.setFilters(filter);
crosstab.submit( );
```

To remove a filter from a level, use actuate.xtabanalyzer.Crosstab.clearFilters( ).

```
crosstab.clearFilters("levelName");
crosstab.submit( );
```

Drilling down within a cross tab

Drilling supports the ability to expand or collapse a member value within a specific level. Construct a XTabAnalyzer.Driller object as shown in the following code:

```
var driller = new actuate.xtabanalyzer.Driller( );
```

To drill up or down, use actuate.xtabanalyzer.Crosstab.drill( ) with the actuate.xtabanalyzer.Driller and actuate.xtabanalyzer.MemberValue classes. In this example, a cross tab has a dimension named Region with three levels:

Country, State, and City. The actuate.xtabanalyzer.Driller object updates the cross tab to display the requested information, as shown in the following code:

```
driller.setAxisType(
   actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
var levelName1 = "Region/Country";
var levelValue1 = "Australia";
var levelName2 = "Region/State";
var levelValue2 = "NSW";

// Create member value objects, and place them in the driller.
var memberValue1 = new
   actuate.xtabanalyzer.MemberValue(levelName1);
memberValue1.setValue(levelValue1);
var memberValue2 = new
   actuate.xtabanalyzer.MemberValue(levelName2);
memberValue2.setValue(levelValue2);
memberValue1.addMember(memberValue2);
driller.addMember(memberValue1);

crosstab.drill(driller);
crosstab.submit( );
```

To reset the drill, use a Driller object with no level names or member values.

```
var driller = new actuate.xtabanalyzer.Driller( );
driller.setAxisType(actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
crosstab.drill(driller);
crosstab.submit( );
```

Controlling the Interactive Crosstabs viewer user interface

Show or hide Interactive Crosstabs viewer features with the actuate.xtabanalyzer.UIOptions class. The UIOptions class includes functions that support the ability to hide or show different features of the viewer. Figure 13-4 shows what functions affect the viewer display.

Pass true or false values to the UIOptions functions to display or hide the portion of the viewer that is associated with that particular function, as shown in the following code:

```
var uiOptions = new actuate.xtabanalyzer.UIOptions( );
uiOptions.enableToolbar(false);
uiOptions.enableCubeView(false);
uiOptions.enableCrosstabView(false);

// ctViewer is an instance of the XTabAnalyzer class.
ctViewer.setUIOptions( uiOptions );
```

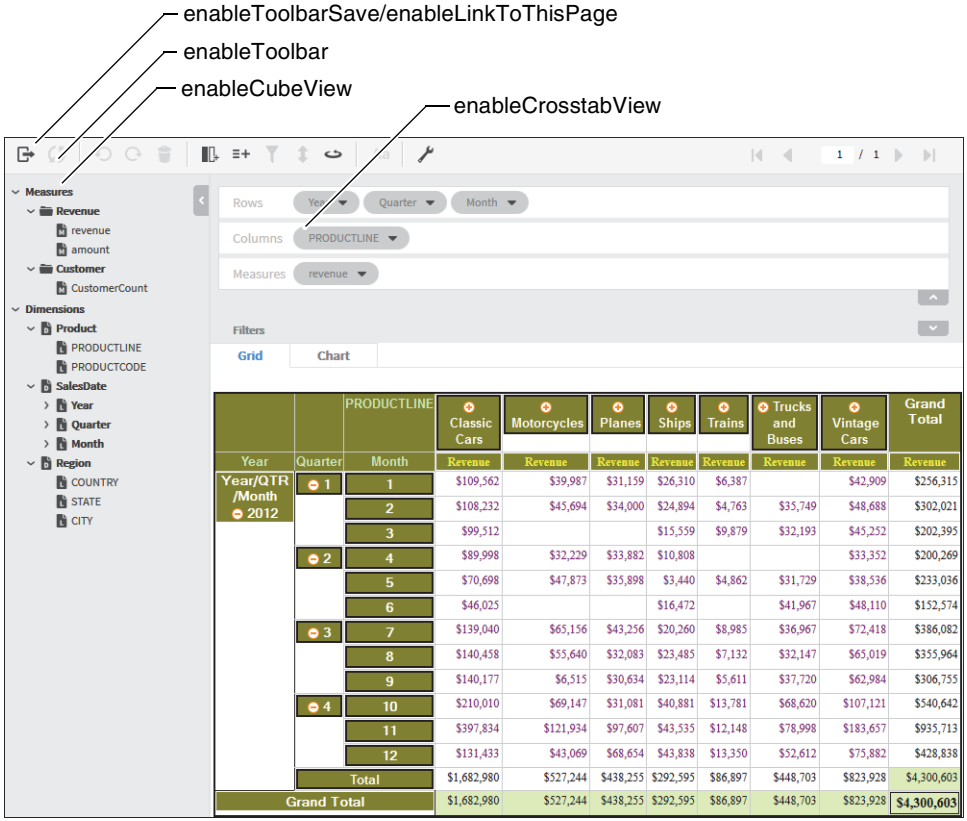


Figure 13-4 Interactive Crosstabs viewer showing areas altered with UIOptions

This code produces a viewer similar to Figure 13-5.

In addition to the UIOptions class, some details shown within the viewer can be hidden with Crosstab.showDetail( ) and Crosstab.hideDetail( ).

For example, the cross tab in Figure 13-5 has a SalesDate dimension consisting of three levels: year, quarter, and month. The following code hides the detail from the quarter level of the dimension. In this example, crosstab is an actuate.xtabanalyzer.Crosstab object:

```
crosstab.hideDetail("SalesDate/quarter");
crosstab.submit( );
```

Grid | Chart

		PRODUCTLINE	Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars	Grand Total
Year	Quarter	Month	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue
Year/QTR /Month 2012	1	1	$109,562	$39,987	$31,159	$26,310	$6,387		$42,909	$256,315
		2	$108,232	$45,694	$34,000	$24,894	$4,763	$35,749	$48,688	$302,021
		3	$99,512			$15,559	$9,879	$32,193	$45,252	$202,395
	2	4	$89,998	$32,229	$33,882	$10,808			$33,352	$200,269
		5	$70,698	$47,873	$35,898	$3,440	$4,862	$31,729	$38,536	$233,036
		6	$46,025			$16,472		$41,967	$48,110	$152,574
	3	7	$139,040	$65,156	$43,256	$20,260	$8,985	$36,967	$72,418	$386,082
		8	$140,458	$55,640	$32,083	$23,485	$7,132	$32,147	$65,019	$355,964
		9	$140,177	$6,515	$30,634	$23,114	$5,611	$37,720	$62,984	$306,755
	4	10	$210,010	$69,147	$31,081	$40,881	$13,781	$68,620	$107,121	$540,642
		11	$397,834	$121,934	$97,607	$43,535	$12,148	$78,998	$183,657	$935,713
		12	$131,433	$43,069	$68,654	$43,838	$13,350	$52,612	$75,882	$428,838
	Total		$1,682,980	$527,244	$438,255	$292,595	$86,897	$448,703	$823,928	$4,300,603
Grand Total			$1,682,980	$527,244	$438,255	$292,595	$86,897	$448,703	$823,928	$4,300,603

Figure 13-5 Interactive Crosstabs viewer with settable UIOptions off

The code in this example modifies the cross tab so it longer shows the month detail level, as shown in Figure 13-6.

		Classic Cars	Motorcycles	Planes	Vintage Cars	Grand Total
Year	Quarter	Revenue	Revenue	Revenue	Revenue	Revenue
2014	1	$317,307	$85,682	$65,159	$136,849	$604,997
	2	$206,722	$80,101	$69,780	$119,998	$476,601
	3	$419,675	$127,311	$105,974	$200,421	$853,380
	4	$739,277	$234,150	$197,342	$366,660	$1,537,430
Grand Total		$1,682,980	$527,244	$438,256	$823,928	$3,472,408

Figure 13-6 Cross tab with level detail hidden

To display the detail again, use showDetail( ).

```
var axisType = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
crosstab.showDetail(axisType,"SalesDate/quarter");
crosstab.submit( );
```

Chapter

13

Actuate JavaScript API classes

This chapter contains:

- Actuate JavaScript API overview
- Actuate JavaScript API classes quick reference
- Actuate JavaScript API reference

Actuate JavaScript API overview

The Actuate JavaScript API is a set of JavaScript classes used to create custom web content that contains Actuate BIRT reports and report elements.

An HTML-based JSDoc JavaScript API class reference is provided for iHub Visualization Platform client and Actuate Java Components in the following file:

```
<Context Root>\help\jsapi\index.html
```

About the actuate namespace

All of the Actuate JavaScript API classes are in the actuate namespace. To use the viewer element, call the actuate.Viewer class.

In addition, the Actuate JavaScript API has a static class:

```
actuate
```

This class handles connections to Actuate web applications and is the only static class in the Actuate JavaScript API.

Using the Actuate library

The Actuate JavaScript library is available from any iHub Visualization Platform client installation or Actuate Java Components. The URL for the library is:

```
http://127.0.0.1:8080/iportal/jsapi
```

- 127.0.0.1:8080 is the host name and TCP port for an available Actuate web application host.
- /iportal is the context root for the web application.
- /jsapi is the default location of the Actuate JavaScript API libraries.

A script tag loads the Actuate JavaScript API library, as shown in the following code:

```
<script type="text/javascript" src="http://127.0.0.1:8080
   /iportal/jsapi">
</script>
```

To call JavaScript functions, use additional script tags after the script tag that adds these libraries for the page.

Actuate JavaScript API classes quick reference

Table 14-1 lists the Actuate JavaScript API classes.

Table 14-1 Actuate JavaScript API classes

JavaScript class	Description
actuate	Entry point to the Actuate JavaScript API library
actuate.AuthenticationException	Exception caused by failed authentication
actuate.ConnectionException	Exception caused by a failed connection
actuate.data.Filter	Conditions to filter data
actuate.data.ReportContent	Represents downloaded content
actuate.data.Request	Represents and processes a request for report data
actuate.data.ResultSet	Results retrieved from a report document in response to a request
actuate.data.Sorter	Sort conditions to sort data
actuate.DataService	Data services to retrieve data from a report document
actuate.Exception	Exception object passed to a callback function or exception handler
actuate.Parameter	Parameters from a report
actuate.parameter.Constants	Global navigation and layout constants used for the Parameter class
actuate.parameter.ConvertUtility	Converts parameters into specific and generic formats
actuate.parameter.EventConstants	Defines the events for parameters this API library supports
actuate.parameter.NameValuePair	Display name and the associated value
actuate.parameter.ParameterData	a high-level wrapper for an actuate.parameter .ParameterDefinition object
actuate.parameter.ParameterDefinition	Qualities, options, name, and format for a parameter as the parameter displays and accepts values

(continues)

Table 14-1 Actuate JavaScript API classes (continued)

JavaScript class	Description
actuate.parameter.ParameterValue	The parameter's value as processed by a report
actuate.report.Chart	A report chart
actuate.report.DataItem	A report data item
actuate.report.FlashObject	A report Flash object
actuate.report.Gadget	A report gadget
actuate.report.HTML5Chart.ClientChart	An HTML5 enabled chart
actuate.report.HTML5Chart.ClientOption	Options for an HTML5 enabled chart
actuate.report.HTML5Chart.ClientPoint	A data point for an HTML5 enabled chart
actuate.report.HTML5Chart.ClientSeries	A data series for an HTML5 enabled chart
actuate.report.HTML5Chart.Highcharts	A Highcharts object
actuate.report.HTML5Chart.Renderer	A Highcharts renderer object
actuate.report.Label	A report label element
actuate.report.Table	A report table element
actuate.report.TextItem	A report text element
actuate.ReportExplorer	The report explorer general container
actuate.reportexplorer.Constants	Global constants used for ReportExplorer class
actuate.reportexplorer.EventConstants	Global EventConstants used for ReportExplorer class
actuate.reportexplorer.File	A file listed in the ReportExplorer and the file's properties
actuate.reportexplorer.FileCondition	A JavaScript version of com.actuate.schemas.FileCondition
actuate.reportexplorer.FileSearch	A JavaScript version of com.actuate.schemas.FileSearch
actuate.reportexplorer.FolderItems	A JavaScript version of com.actuate .schemas.GetFolderItemsResponse
actuate.reportexplorer.PrivilegeFilter	A JavaScript version of com.actuate .schemas.PrivilegeFilter

Table 14-1 Actuate JavaScript API classes (continued)

JavaScript class	Description
actuate.RequestOptions	URL parameters for requests to an iHub volume
actuate.Viewer	A report viewer component that can be embedded in an HTML page
actuate.viewer.BrowserPanel	A non-scrolling panel display
actuate.viewer.EventConstants	Defines the events for the viewer this API library supports
actuate.viewer.PageContent	Content shown on the viewer
actuate.viewer.ParameterValue	Parameter values in the viewer
actuate.viewer.RenderOptions	Options for downloading reports
actuate.viewer.ScrollPanel	A scrolling panel display
actuate.viewer.SelectedContent	Selected report element
actuate.viewer.UIConfig	Enables UI elements of the scrolling panel display
actuate.viewer.UIOptions	Enables UI elements of the viewer
actuate.viewer.ViewerException	Exception constants supported for the viewer

Actuate JavaScript API reference

This section provides an alphabetical listing of the JavaScript API classes.

Class actuate

Description The entry point to the Actuate JavaScript API library. The actuate class uses load( ) to generate data, viewer, cross tab, parameter, explorer, and other components. The actuate class uses initialize( ) and authenticate( ) to connect to an Actuate web application service.

Use actuate.load( ) before calling actuate.initialize( ). The actuate.initialize( ) function loads all of the components added with load( ).

The initialize( ) function connects to an initial Actuate web application service. To connect to additional services simultaneously, use authenticate( ). Call initialize( ) before calling authenticate( ).

Constructor

The static actuate class loads when the a <script> element loads the Actuate JavaScript API.

Function summary

Table 14-2 lists actuate functions.

Table 14-2 actuate functions

Function	Description
authenticate( )	Connects to an Actuate web application service and authenticates
getDefaultIportalUrl( )	Returns the default service URL
getDefaultRequestOptions( )	Returns the default request options
getVersion( )	Returns the Actuate web application version
getViewer( )	Returns a viewer instance containing the given bookmark element
initialize( )	Connects to an initial Actuate web application service, loads an initial component, and invokes a callback function
isConnected( )	Reports whether a given Actuate web application is connected
isInitialized( )	Returns whether a library is initialized
load( )	Loads the library for an additional component
logout( )	Logs a user out of an Actuate web application service

authenticate

Syntax void authenticate(string iPortalURL, actuate.RequestOptions requestOptions, string userid, string password, function callback, string credentials, function errorCallback)

Connects to the Actuate web application service that is addressed by iPortalURL and authenticates the connection.

Parameters

iPortalURL
The iPortalURL parameter is a required string parameter that specifies the target Actuate web application URL.

requestOptions
The requestOptions parameter is an optional actuate.RequestOptions object. The requestOptions parameter defines the URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. Functions in the RequestOptions class enable the addition of custom parameters to the URL. When requestOptions is null, authenticate( ) uses the default parameter values for the target Actuate web application URL. These default parameter values are defined in the Actuate web application's web.xml file.

userid
The userid parameter is an optional string parameter that contains the login user id when the login user id is not provided in the session. You do not need to set this parameter for a default installation of Actuate Java Components.

password
The password parameter is an optional string parameter that contains the login password when the login password is not provided in the session. You do not need to set this parameter for a default installation of Actuate Java Components.

credentials
The credentials parameter is an optional string parameter. This parameter holds information that supports checking user credentials with an externalized system such as LDAP. The credentials parameter supports additional credential information for any additional security systems in place on the application server where the web service is deployed.

callback
The callback parameter is an optional function to call after initialization. The actuate.authenticate( ) function passes the following variables to the callback function:

- iportalURL: The iportal URL passed in from the iPortalURL parameter
- userid: The authenticated user ID
- iserverURL: The BIRT iHub, if applicable
- volume: The volume name, if applicable

errorCallback
The errorCallback parameter is an optional function that specifies a function to call when an error occurs. The possible errors are actuate.ConnectionException, actuate.AuthenticationException, and actuate.Exception. The callback function must take an exception as an argument.

Example To connect to an additional Actuate web service called digits, use code similar to the following:

```
actuate.authenticate("http://digits:8080/iportal", null, null,
   null, null, null, null);
```

getDefaultIportalUrl

Syntax String getDefaultIportalUrl( )

Returns the default service URL.

Returns String. The default service URL.

Example This example calls actuate.getDefaultIportalUrl( ) to return the default service URL:

```
alert ("The default service URL is " +
   actuate.getDefaultIportalUrl( ));
```

getDefaultRequestOptions

Syntax actuate.RequestOptions getDefaultRequestOptions( )

Returns the default request options.

Returns actuate.RequestOptions object that contains the default request options.

Example This example calls actuate.getDefaultRequestOptions( ) to return the default iHub URL:

```
alert ("The default iHub URL is " +
   actuate.getDefaultRequestOptions( ).getServerUrl( ));
```

getVersion

Syntax string getVersion( )

Returns the Actuate web application version.

Returns String. The string contains the Actuate web application version in the format "#version# (Build #buildnumber#)".

Example The following sample code displays the version in an alert box:

```
alert("Version: " + actuate.getVersion( ));
```

getViewer

Syntax actuate.Viewer getViewer(string bookmark)

actuate.Viewer getViewer(htmlelement viewer)

Returns a viewer instance containing the given bookmark element. Load the viewer module before calling actuate.getViewer( ).

Parameters

bookmark
This string parameter contains the name of the bookmark to retrieve or the name of an HTML element.

viewer
This parameter is the DOM htmlelement object for the HTML element that contains a viewer.

Returns An actuate.Viewer object that contains a viewer. When actuate.getViewer( ) does not find a viewer, the function returns null.

Example To retrieve the viewer assigned to the first_viewer element on the page, use code similar to the following:

```
currentViewer = actuate.getViewer("first_viewer");
```

initialize

Syntax void initialize(string iPortalURL, actuate.RequestOptions requestOptions, reserved, reserved, function callback, function errorCallback)

Connects to an initial Actuate web application service, loads all of the components added with load( ), and invokes a callback function.

Authentication is optional in initialize( ).

When using more than one service in one mashup page, use actuate.authenticate( ) to connect to additional services.

Parameters

iPortalURL
String. The target Actuate web application URL.

requestOptions
actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send in the authentication request, such as the iHub URL, volume, or repository type. It can also add custom parameters to the URL. If requestOptions is null, initialize( ) uses the default parameter values for the target Actuate web application URL. These default parameter values are defined in Actuate web application's web.xml file. Loading performance is improved if you create a requestOptions object to pass to initialize().

reserved
Set to null.

reserved
Set to null.

callback
Function. The callback function called after the initialization is done. The following variables are passed to the callback function:

- iportalUrl: The iportal URL passed in from the iPortalURL parameter
- userId: The authenticated user ID
- iserverUrl: The BIRT iHub URL, if applicable
- volume: The volume name, if applicable

errorCallback
Function. The function to call when an error occurs. The possible errors are actuate.ConnectionException, actuate.AuthenticationException, and actuate.Exception. errorCallback must take an exception as an argument.

Example To initialize the client connection to a web service on myhost and then run the init( ) function, use the following code:

```
actuate.initialize("http://myhost:8080/iportal", null, null, null,
   init, null);
```

isConnected

Syntax boolean isConnected(string iportalUrl, actuate.RequestOptions requestOptions)

Returns whether a given Actuate web application URL is connected.

Parameters

iPortalURL
String. The target Actuate web application URL.

requestOptions
actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. It can also add custom parameters to the URL. If requestOptions is null, initialize( ) uses the default parameter values for the target Actuate web application URL. These default parameter values are defined in Actuate web application's web.xml file.

Returns Boolean. True if there is a connection to the given Actuate web application, False if there is no connection or if it is pending.

Example The following sample code connects to the digits service using authenticate if not currently connected:

```
if (!actuate.isConnected("http://digits:8080/iportal", null)){
   actuate.authenticate("http://digits:8080/iportal", null, null,
   null, null, null, null);
}
```

isInitialized

Syntax boolean isInitialized( )

Returns whether the library is already initialized.

Returns Boolean. True if the library is already initialized.

Example The following sample code initializes a connection with the Actuate web service if one is not already initialized:

```
if (!actuate.isInitialized( )){
   actuate.initialize("http://myhost:8080/iportal", null, null,
   null, init, null);
}
```

load

Syntax void load(string componentName)

Specifies a component to be loaded by actuate.initialize( ). The available components are:

- dialog: The dialog component including the actuate.Dialog class
- parameter: The parameter page component including the actuate.Parameter package
- reportexplorer: The report explorer component including the actuate.ReportExplorer package
- viewer: The viewer component including the actuate.Viewer and actuate.DataService packages
- xtabAnalyzer: The interactive crosstab component, including the actuate.XTabAnalyzer package

Parameter **componentName**
String. componentName is a case-sensitive parameter. Valid component names are listed above.

Example To enable a page to use viewer, dialog, and parameters, call actuate.load( ) three times, as shown in the following code:

```
actuate.load("viewer");
actuate.load("dialog");
actuate.load("parameter");
```

logout

Syntax void logout(string iPortalURL, actuate.RequestOptions requestOptions, function callback, function errorCallback)

Logs out from the given Actuate web application URL and removes authentication information from the session. If the application was previously not logged in to this Actuate web application, it generates no errors but still calls the callback function.

Parameters

iPortalURL
String. The target Actuate web application URL.

requestOptions
actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. It can also add custom parameters to the URL. If requestOptions is null, initialize( ) uses the default parameter values for the target Actuate web application URL. These default parameter values are defined in Actuate web application's web.xml file.

callback
Function. Optional. The callback function called after the logout is done.

errorCallback
Function. The function called when an error occurs. The possible errors are actuate.ConnectionException, actuate.AuthenticationException, and actuate.Exception. errorCallback must take an exception as an argument.

Example

The following sample code disconnects to the digits service if currently connected:

```
if (actuate.isConnected("http://digits:8080/iportal", null)){
   actuate.logout("http://digits:8080/iportal", null, null, null);
}
```

Class actuate.AuthenticationException

Description AuthenticationException provides an object to pass to a error callback function when an authentication exception occurs. The AuthenticationException object contains references to the URL, the UserId, and the request options used in the authentication attempt.

Constructor

The AuthenticationException object is constructed when actuate.Authenticate( ) fails.

Function summary

Table 14-3 lists actuate.AuthenticationException functions.

Table 14-3 actuate.AuthenticationException functions

Function	Description
getIportalUrl( )	Returns the web service URL
getRequestOptions( )	Returns the request options
getUserId( )	Returns the user ID

getIportalUrl

Syntax string AuthenticationException.getIportalUrl( )

Returns the Deployment Kit for BIRT reports or iHub Visualization Platform client URL.

Returns String.

Example The following sample code retrieves the URL from an exception:

```
return AuthenticationException.getIportalUrl( );
```

getRequestOptions

Syntax actuate.RequestOptions AuthenticationException.getRequestOptions( )

Returns an instance of the RequestOptions that modified the URL that caused the exception, if applicable.

Returns actuate.RequestOptions object. A RequestOptions object defines URL parameters sent in the authentication request, such as the iHub URL, volume, or repository type. The RequestOptions object can also add custom parameters to the URL.

Example The following sample code retrieves the RequestOptions object that caused the exception:

```
var exceptReqOpts = AuthenticationException.getRequestOptions( );
```

getUserId

Syntax string AuthenticationException.getUserId( )

Returns the UserId used in the failed authentication attempt.

Returns String.

Example The following sample code retrieves the UserId from an exception:

```
return AuthenticationException.getUserId( );
```

Class actuate.ConnectionException

Description A container for a connection exception. ConnectionException provides an object to pass to a error callback function when an exception occurs.

Constructor

The ConnectionException object is constructed when there is a connection issue. For example, actuate.ConnectionException is created when a wrong URL is given in actuate.initialize( ) or actuate.authenticate( ), or if the server was unreachable.

Function summary

Table 14-4 describes actuate.ConnectionException functions.

Table 14-4 actuate.ConnectionException function

Function	Description
getUrl( )	Returns the whole URL

getUrl

Syntax string ConnectionException.getUrl( )

Returns the complete URL sent with the connection request.

Returns String. The complete URL that was sent with the connection request.

Example This example calls ConnectionException.getUrl( ) to return the complete URL from a connection request:

```
alert ("Connection Error at " + ConnectionException.getUrl( ));
```

Class actuate.data.Filter

Description Specifies filter conditions to be used by other classes when processing data. A filter has three components: a column, an operator, and a value or set of values. The condition is expressed as "value1 operator value2". For some operators, like "IN", the expression will be "value1 IN value2" where value2 is an array of strings.

Format numbers and date/time values in a locale neutral format, for example, "2.5" or "09/31/2008 01:02:03 AM".

Constructor

Syntax

```
actuate.data.Filter(string columnName, string operator, string[ ] value1,
    string[ ] value2)
```

Constructs a filter object.

Parameters

columnName
String. The column name.

operator
String. The operator can be any operator. Table 14-5 lists the valid filter operators and the number of arguments to pass to the constructor or setValues( ).

Table 14-5 Filter operators

Operator	Description	Number of arguments
BETWEEN	Between an inclusive range	2
BOTTOM_N	Matches the bottom n values	1
BOTTOM_PERCENT	Matches the bottom percent of the values	1
EQ	Equal	1
FALSE	Matches false Boolean values	0
GREATER_THAN	Greater than	1
GREATER_THAN_OR_EQUAL	Greater than or equal	1
IN	Matches any value in a set of values	1+
LESS_THAN	Less than	1
LESS_THAN_OR_EQUAL	Less than or equal	1
LIKE	Search for a pattern	1
MATCH	Equal	1

Table 14-5 Filter operators

Operator	Description	Number of arguments
NOT_BETWEEN	Not between an inclusive range	2
NOT_EQ	Not equal	1
NOT_IN	Does not match any value in a set of values	1+
NOT_LIKE	Search for values that do not match a pattern	1
NOT_MATCH	Not equal	1
NOT_NULL	Is not null	0
NULL	Is null	0
TOP_N	Matches the top n values	1
TOP_PERCENT	Matches the top percent of the values	1
TRUE	Matches true Boolean values	0

value1
String or array of strings. The first value to compare to the column value for the BETWEEN or NOT_BETWEEN operators.

value2
String or array of strings. This parameter is only required for the BETWEEN or NOT_BETWEEN operators.

Example To select all of the rows matching a list of countries in their country fields, use code similar to the following:

```
var filter = new actuate.data.Filter("COUNTRY",
   actuate.data.Filter.IN,["Canada" , "USA", "UK", "Australia"]);
```

To create a filter to display only entries with a CITY value of NYC, use the following code:

```
var cityfilter = new actuate.data.Filter("CITY",
   actuate.data.Filter.EQ, "NYC");
```

Function summary

Table 14-6 lists actuate.data.Filter functions.

Table 14-6 actuate.data.Filter functions

Function	Description
getColumnName( )	Returns the column name
getOperator( )	Returns the filter operator
getValues( )	Returns the value or values of the filter
setColumnName( )	Sets the name of the column to filter
setOperator( )	Sets the operator for the filter
setValues( )	Sets string values for the filter

getColumnName

Syntax string Filter.getColumnName( )

Returns the column name.

Returns String. The name of the column.

Example This example retrieves the name of the column:

```
function retrieveColumnName(myFilter){
   var colname = myFilter.getColumnName( );
   return colname;
}
```

getOperator

Syntax string Filter.getOperator( )

Returns the filter operator.

Returns String. Table 4-10 lists the legal filter operator values.

Example This example retrieves the name of the filter operator:

```
function retrieveFilterOperator(myFilter){
   var myOp = myFilter.getOperator( );
   return myOp;
}
```

getValues

Syntax string Filter.getValues( )

string[ ] Filter.getValues( )

Returns the evaluated results of this filter. When the filter is constructed or set with a single argument, the returned value corresponds to the single argument.

When two arguments or an array are set in the filter, the return value is an array of values.

Returns String or array of strings. Returns the value or values from the filter.

Example This example retrieves the name of the filter operator:

```
function retrieveValues(myFilter){
   var myVals = myFilter.getValues( );
   return myVals;
}
```

setColumnName

Syntax void Filter.setColumnName(columnName)

Sets the name of the column to filter.

Parameter **columnName**
String. The column name.

Example This example sets the name of the column to filter to Sales:

```
function setFilterSales( myfilter ){
   myfilter.setColumnName("Sales");
}
```

setOperator

Syntax void Filter.setOperator(string operator)

Sets filter operator. The operator determines the comparison made between the data in the column and the value or values set in the filter.

Parameter **operator**
String. The operator can be any valid operator. Table 14-5 lists the valid filter operators and the number of arguments to pass to Filter.setValues( ).

Example This example sets the filter to retrieve the bottom five values:

```
function setFilterBot5( ){
   myfilter.setOperator("BOTTOM_N");
   myfilter.setValues("5");
}
```

setValues

Syntax void Filter.setValues(string value)

void Filter.setValues(string value1, string value2)

void Filter.setValues(string[ ] values)

Sets string values for the filter to compare to the data in the column according to the operator. Table 14-5 lists the valid filter operators and the values they use. Takes either one or two values, or one array of values.

Parameters

value
String. The value to compare to the column value.

value1
String. The first value to compare to the column value for the BETWEEN operator.

value2
String. The second value to compare to the column value for the BETWEEN operator.

values
Array of strings. The values to compare to the column value for the IN operator.

Example

This example sets the filter to retrieve values between 10 and 35:

```
function setFilter( myfilter ){
   myfilter.setOperator("BETWEEN");
   myfilter.setValues("10","35");
}
```

Class actuate.data.ReportContent

Description The ReportContent class is a container for downloadable report content.

Constructor

Syntax actuate.data.ReportContent(data)

Constructs a ReportContent object.

Parameter **data**
String. Content text.

Function summary

Table 14-7 describes actuate.data.ReportContent functions.

Table 14-7 actuate.data.ReportContent function

Function	Description
getTextContent( )	Returns the text in the downloaded content

getTextContent

Syntax string ReportContent.getTextContent( )

Returns the text in the downloaded content.

Returns String. The text in the downloaded content.

Example To make a callback function that prints back the first line of text from some downloaded content back onto the page, use code similar to the following:

```
function callback(data1){
   var rcontent = data1.ReportContent.getTextContent( );
   var contentArray = rcontent.split("\n");
   var items = contentArray.length
   document.write("<P>\n")
   document.write(listItems.arguments[o] + "\n</P>")
}
```

Class actuate.data.Request

Description Specifies a request for retrieving data and the conditions for that request. This class provides the scope for a request by defining a target element and a range of rows. The scope of the request determines what goes into a ResultSet. Functions that use request can only retrieve ResultSets from report elements that have an explicit bookmark.

Constructor

Syntax actuate.data.Request(string bookmark, integer startRow, integer maxRow)

Constructs a request object that other classes use to retrieve data.

Parameters

bookmark
String. A bookmark that identifies an Actuate report element. The actuate.data.Request object uses the bookmark to identify the report element to request information from. If null, Request uses the first bookmark. Functions that use request can only retrieve actuate.data.ResultSet objects from report elements that have an explicit bookmark.

startRow
Integer. The numerical index of the requested first row. The smallest value is 0.

maxRow
Integer. The numerical index of the requested last row. 0 indicates no limit.

Function summary

Table 14-8 lists actuate.data.Request functions.

Table 14-8 actuate.data.Request functions

Function	Description
getBookmark( )	Returns the bookmark name
getColumns( )	Returns the column names
getFilters( )	Returns filters defined in this data condition
getMaxRows( )	Returns the max row number
getSorters( )	Returns sorters defined in this data condition
getStartRow( )	Returns the start row number
setBookmark( )	Sets the bookmark name
setColumns( )	Sets the columns to return
setFilters( )	Sets the filters for the returned data

Table 14-8 actuate.data.Request functions

Function	Description
setMaxRows( )	Sets the max row number
setSorters( )	Sets the sorters for the returned data
setStartRow( )	Sets the start row number

getBookmark

Syntax string Request.getBookmarkName( )

Returns the bookmark name for this request.

Returns String. The bookmark used in the request object's constructor.

Example This example retrieves the bookmark set in the myRequest object:

```
return myRequest.getBookmarkName( );
```

getColumns

Syntax string[ ] Request.getColumns( )

Returns a list of column names that match the request.

Returns Array of strings. The column names.

Example This example retrieves the first, third, and fifth column names from the request object myRequest:

```
function get135Columns(myRequest){
   var columns = myRequest.getColumns( );
   return columns[0];
   return columns[2];
   return columns[4];
}
```

getFilters

Syntax actuate.data.Filter[ ] Request.getfilters( )

Returns filters set for this request.

Returns Array of actuate.data.Filter objects.

getMaxRows

Syntax integer Request.getMaxRows( )

Returns the maximum number of rows to retrieve.

Returns Integer. The index of the last row in the request. 0 means no limit.

getSorters

Syntax actuate.data.Sorter[ ] Request.getSorters( )

Returns sorters assigned to this request.

Returns Array of actuate.data.Sorter objects.

getStartRow

Syntax Integer Request.getStartRow( )

Returns the index of the starting row as an integer.

Returns Integer. The startRow value. The first row in a column has an index of 0.

setBookmark

Syntax void Request.setBookmark(string bookmark)

Sets the bookmark of the element from which to request values.

Parameter **bookmark**
String. A bookmark.

Example This example sets the bookmark for the myRequest object to the string myRequestStart:

```
function setMyRequestBookmark(myRequest){
   myRequest.setBookmark("myRequestStart");
}
```

setColumns

Syntax void Request.setColumns(string[ ] columns)

Sets the request column names.

Parameter **columns**
An array of strings designating the columns of requested data. Use an array for this argument, even if there is only one value.

setFilters

Syntax void Request.setFilters(actuate.data.Filter[ ] filters)

Adds filters to a request. Filters further refine the set of data provided by a request. Using setFilter removes the previous filters from the request object. All of the filters set in a request are applied when the request is used.

Parameter **filters**
An array of actuate.data.Filter objects or a single actuate.data.Filter object to refine the request. Use an array for this argument, even if there is only one value.

setMaxRows

Syntax void Request.setMaxRows(integer maxrow)

Sets the maximum number of rows to retrieve.

Parameter **maxrow**
Integer. The numerical value of the index for the last row to request. 0 indicates no limit.

Example This example sets the index of the last row for the myRequest request object to 50:

```
myRequest.setMaxRows(50);
```

setSorters

Syntax void Request.setSorts(actuate.data.Sorter[ ] sorters)

Adds sorters to a request to sort the set of data that a request provides. Sorting the data increases the effectiveness of requests by providing the data in a relevant order. Using setSorters removes the previous sorter objects from the request object. All of the sorters set in a request are applied when the request is used.

Sorters are applied in the order that they occur in the array. For example, if the first sorter specifies sorting on a state column and the second sorter specifies sorting on a city column, the result set is sorted by city within each state.

Parameter **sorters**
An array of actuate.data.Sorter objects or a single actuate.data.Sorter object to sort the result of the request. Use an array for this argument, even if there is only one value.

Example This example sets the alphaNumericSorterSet array in myRequest:

```
myRequest.setSorters(alphaNumericSorterSet);
```

setStartRow

Syntax void Request.setStartRow(integer startrow)

Sets the requested first row.

Parameter **startrow**
Integer. The numerical value of the index for the first row to request. The first row in a column has an index of 0.

Example This example sets the index of the first row for the myRequest request object to 10:

```
myRequest.setStartRow(10);
```

Class actuate.data.ResultSet

Description The actuate.data.ResultSet class represents the data retrieved from a report document. The functions in the actuate.data.ResultSet class access the data by row. The actuate.data.ResultSet class keeps an internal reference to the current row and increments the current row with next( ).

Constructor

There is no public constructor for actuate.data.ResultSet. The actuate.DataService.downloadResultSet and actuate.Viewer.downloadResultSet functions instantiate the ResultSet object. Set the reference to the ResultSet object in the callback function. For example, when the result set is used as the input parameter for the callback function, result becomes the label for the ResultSet, as shown below:

```
viewer.downloadResultSet(request, parseRS)
function parseRS(resultset){
   // do something with resultset
}
```

Function summary

Table 14-9 lists actuate.data.ResultSet functions.

Table 14-9 actuate.data.ResultSet functions

Function	Description
getColumnNames( )	Returns the column names
getValue( )	Returns the data by the given column index
next( )	Increments the current row

getColumnNames

Syntax string[ ] Request.getColumnNames( )

Returns a list of column names.

Returns Array of strings. The column names.

Example This example retrieves the first, third, and fifth column names from the ResultSet object myResult:

```
function get135Columns(myResult){
   var columns = myResult.getColumns( );
   return columns[0];
   return columns[2];
   return columns[4];
}
```

getValue

Syntax string ResultSet.getValue(integer columnIndex)

Returns the value of the specified column from the current row. Specify the column by its numerical index. Use next( ) before using getValue( ) to set the cursor to the first record.

Parameter **columnIndex**
Integer. The numerical index of the column from which to retrieve data.

Returns String. The field value.

Example This example returns the value for the column with an index value of 4 from the current row in the ResultSet object myResult:

```
return myResult.getValue(4);
```

next

Syntax boolean next( )

Increments the current row for the ResultSet. When no current row is set, next( ) sets the current row to the first row in the ResultSet. When no next row exists, next( ) returns false.

Returns Boolean. True indicates a successful row increment. False indicates that there are no further rows.

Example This example returns the value for the column with an index value of 4 from all of the rows in the ResultSet object myResult:

```
function getColumn4Rows(myResult){
   var nextrow = myResult.next( );
   while (nextrow){
      return myResult.getValue(4);
      nextrow = myResult.next( );
   }
}
```

Class actuate.data.Sorter

Description Specifies the conditions for sorting data as it is returned by a request or stored temporarily in a local ResultSet object. The sort arranges rows based on the value of a specified column.

Constructor

Syntax actuate.data.Sorter(string columnName, boolean ascending)

Constructs a sorter object.

Parameters

columnName
String. The name of the column to sort.

ascending
Boolean. True sets sorting to ascending. False sets sorting to descending.

Function summary

Table 14-10 lists actuate.data.Sorter functions.

Table 14-10 actuate.data.Sorter functions

Function	Description
getColumnName( )	Returns the column name
isAscending( )	Returns true if the current sorting is ascending
setAscending( )	Sets the sort order to ascending or descending
setColumnName( )	Sets the column to which this sorter applies

getColumnName

Syntax string Sorter.getColumnName( )

Returns the name of the column to sort on.

Returns String. The column name.

Example This example displays an alert box that contains the column name currently being sorted on:

```
function showMyColumnName(mySorter){
   var sortColName = mySorter.getColumnName( );
   alert(sortColName);
}
```

isAscending

Syntax boolean Sorter.isAscending( )

Returns true if the current sort order is ascending. Returns false if the current order is descending.

Returns Boolean. True indicates ascending. False indicates descending.

Example This example checks if the current sort order is ascending. When the current sort order is descending, this code sets the order to ascending:

```
function makeAscending(mySort){
   if (mySort.isAscending( )) {
      return;
   } else {
      mySort.setAscending(true);
   }
}
```

setAscending

Syntax void Sorter.setAscending(boolean ascending)

Sets the sort order to ascending or descending.

Parameter **ascending**
Boolean. True sets the sort order to ascending. False sets the sort order to descending.

Example This example checks if the current sort order is descending. When the current sort order is ascending, this code sets the order to descending:

```
function makeAscending(mySort){
   if (mySort.isAscending( )) {
      return;
   } else {
      mySort.setAscending(true);
   }
}
```

setColumnName

Syntax void Sorter.setColumnName(string columnName)

Sets the column to sort on.

Parameter **columnName**
String. The column name.

Example This example makes the current sorter arrange the result set ascending by the Sales column:

```
function makeAscendingOnSales(mySort){
   mySort.setColumnName("Sales");
   if (mySort.isAscending( )) {
      return;
   } else {
      mySort.setAscending(true);
   }
}
```

Class actuate.DataService

Description Connects to an Actuate web application service to retrieve data from Actuate BIRT reports as a ResultSet.

Constructor

Syntax actuate.DataService(string iportalUrl, actuate.RequestOptions requestOptions)

Constructs a DataService object.

Parameters

iportalUrl
String. Optional. The URL of an Actuate web application service. The DataService uses the web application service set in actuate.initialize if one is not specified.

requestOptions
actuate.RequestOptions object. Optional. Specifies the request options for the iportal web service connection. The DataService uses the options set in actuate.initialize if one is not specified.

Function summary

Table 14-11 lists actuate.DataService functions.

Table 14-11 actuate.DataService functions

Function	Description
downloadResultSet( )	Retrieves data from a report in a ResultSet object

downloadResultSet

Syntax void DataService.downloadResultSet(string datasource, actuate.data.Request request, function callback, function errorCallback)

Returns data from an Actuate BIRT report document managed by an Actuate web application. The actuate.data.ResultSet object that downloadResultSet( ) returns is used by the callback function.

Parameters

datasource
String. The repository path and name of the file from which to retrieve data.

request
actuate.data.Request object. Specifies the request for the report.

callback
Function. The callback function to use after the ResultSet finishes downloading. This function must take the returned ResultSet object as an input parameter.

errorCallback
Function. Optional. The function to call when an error occurs. The possible errors are actuate.Exception objects. The errorCallback( ) function must take an exception as an argument.

Example This example retrieves a result set as specified by the myRequest request object, and calls the makeAscendingSales function, which must take a actuate.data.ResultSet object as an input parameter:

```
var myRequest = new actuate.data.Request("Top_5_Customers", 1, 0);
var myDataService =
   new actuate.DataService( "http://127.0.0.1:8080/iportal" );
myDataService.downloadResultSet("/Public
   /BIRT and BIRT Studio Examples/Customer Dashboard.rptdocument",
   myRequest, makeAscendingSales, errorCallback);
```

Class actuate.Exception

Description A container for an uncategorized exceptions that also supports specific exceptions. Exception provides an object to pass to a callback function or event handler when an exception occurs. The Exception object contains references to the exception's origin, description, and messages.

Constructor

The Exception object is constructed when unspecified exceptions occur. The exceptions are divided into three types, which determine the contents of the Exception object. These types are:

- ERR_CLIENT: Exception type for a client-side error
- ERR_SERVER: Exception type for a server error
- ERR_USAGE: Exception type for a JSAPI usage error

Function summary

Table 14-12 lists actuate.Exception functions.

Table 14-12 actuate.Exception functions

Function	Description
getDescription( )	Returns details of the exception
getErrCode( )	Returns error code for server-side exceptions
getMessage( )	Returns a short message about the exception
getType( )	Returns the type of exception error
isExceptionType( )	Confirms exception type

getDescription

Syntax string Exception.getDescription( )

Returns exception details as provided by the Server, Client, and User objects.

Returns String. A detailed description of the error. Information is provided according to the type of exception generated, as shown below:

- Server error: The SOAP string
- Client error: For the Firefox browser, a list comprised of fileName+number+stack
- Usage error: Any values set in the object generating the exception

Example This example displays the server error description in an alert box:

```
alert("Server error: " + Exception.getDescription( ));
```

getErrCode

Syntax string Exception.getErrCode( )

Returns the error code for server exceptions.

Returns String. A server error code.

Example This example displays the server error code in an alert box:

```
alert("Server error: " + Exception.getErrCode( ));
```

getMessage

Syntax string Exception.getMessage( )

Returns a short message about the exception. This message is set for an actuate.Exception object with the actuate.Exception.initJSException( ) function.

Returns String. A server error code.

Example This example displays the error's short message code in an alert box:

```
alert("Error Message: " + Exception.getMessage( ));
```

getType

Syntax string Exception.getType( )

Returns the type of the exception:

- ERR_CLIENT: Exception type for a client-side error
- ERR_SERVER: Exception type for a server error
- ERR_USAGE: Exception type for a Actuate JavaScript API usage error

Returns String. A server error code.

Example This example displays the error type in an alert box:

```
alert("Error type: " + Exception.getType( ));
```

isExceptionType

Syntax boolean Exception.isExceptionType(object exceptionType)

Compares the input object to the exception contained in this actuate.Exception object to the exceptionType object argument.

Parameter **exceptionType**
Object. Either an Exception object, such as an instance of actuate.Viewer.ViewerException, or the name of an Exception class as a string.

Returns Boolean. Returns true if the exception contained in this actuate.Exception object matches the exceptionType object argument.

Example To alert the user when the exception e is a usage error, use code similar to the following:

```
if (e.isExceptionType(actuate.exception.ERR_USAGE)){
   alert('Usage error occurred!');
}
```

Class actuate.Parameter

Description The actuate.Parameter class retrieves and displays Actuate BIRT report parameters in an HTML container. Users can interact with the parameters on the page and pass parameter values to an actuate.Viewer object, but not to the server directly.

The actuate.Parameter class displays the parameters by page. The actuate.parameters.navigate( ) function changes the page display or changes the current position on the page.

Constructor

Syntax actuate.Parameter(string container)

Constructs a parameter object for a page, initializing the parameter component.

Parameter **container**
String. The name of the HTML element that displays the rendered parameter component or a container object. The constructor initializes the parameter component but does not render it.

Function summary

Table 14-13 lists actuate.Parameter functions.

Table 14-13 actuate.Parameter functions

Function	Description
downloadParameters( )	Returns an array of ParameterDefinition objects
downloadParameterValues( )	Returns an array list of ParameterValue objects
getLayout( )	Returns the parameter layout
getParameterGroupNames( )	Returns the names of the groups of parameters
getReportName( )	Returns the name of the report file
getTransientDocumentName( )	Returns the name of the transient document
hideNavBar( )	Hides the navigation bar
hideParameterGroup( )	Hides report parameters by group
hideParameterName( )	Hides parameters by name
navigate( )	Navigates the parameter page

(continues)

Table 14-13 actuate.Parameter functions (continued)

Function	Description
onUnload( )	Unloads unused JavaScript variables
registerEventHandler( )	Registers an event handler
removeEventHandler( )	Removes an event handler
renderContent( )	Renders the parameter content to the container
setAutoSuggestDelay( )	Sets the autosuggest delay time
setAutoSuggestFetchSize( )	Sets the fetch size of the autosuggestion list
setAutoSuggestListSize( )	Sets the size of the autosuggestion list
setExpandedGroups( )	Sets the groups to expand by default
setFont( )	Sets the font of the parameter page
setGroupContainer( )	Sets the HTML container for the group
setLayout( )	Sets the parameter layout type
setReadOnly( )	Sets the parameter UI to read-only
setReportName( )	Sets the remote report path and name
setService( )	Sets the Actuate web application service
setShowDisplayType( )	Sets the parameter page to display localized content
submit( )	Submits all the asynchronous operations that the user has requested on this Parameter object and renders the parameter component on the page

downloadParameters

Syntax void Parameter.downloadParameters(function callback)

Retrieves an array of actuate.parameter.ParameterDefinition objects that contain the report parameters for the report and sends the array to the callback function, which must take the array as an input parameter.

Parameter **callback**
Function. The function to execute after the report parameters finish downloading. Parameter.downloadParameters( ) sends an array of actuate.parameter.ParameterDefinition objects to the callback function as an input argument.

Example This example retrieves a set of report parameters and sends them to a callback function.

```
function getChartParams(myParameter){
   myParameter.downloadParameters(callback( ));
}
```

downloadParameterValues

Syntax void Parameter.downloadParameterValues(function callback)

Returns an array of the actuate.parameter.ParameterValue objects for the parameter object. If no values have been set, the parameter object downloads the default values from the server.

Parameter **callback**
Function. The function to execute after the report parameters finish downloading. Parameter.downloadParameterValues( ) sends an array of actuate.parameter.ParameterValue objects to the callback function as an input argument.

Example To download the parameter values and add them to the viewer, the callback function must use the values as an input parameter, as shown in the following code:

```
paramObj.downloadParameterValues(runNext);
function runNext(values){
   viewer.setParameterValues(values);
}
```

getLayout

Syntax string Parameter.getLayout( )

Returns the parameter layout type.

Returns String. The parameter layout, which will match one of the layout constants in actuate.parameter.Constants:

- actuate.parameter.Constants.LAYOUT_NONE
- actuate.parameter.Constants.LAYOUT_GROUP
- actuate.parameter.Constants.LAYOUT_COLLAPSIBLE

Example This example calls getLayout( ) to display the parameter layout type in an alert box:

```
alert(paramObj.getLayout( ));
```

getParameterGroupNames

Syntax string[ ] Parameter.getParameterGroupNames( )

Returns all the group names for the parameter page as an array of strings.

Returns Array of strings. Each string is a group name.

Example This example displays an alert box with the name of the first group for the parameter page:

```
var groupNames = paramObj.getParameterGroupNames( );
alert("First Group Name: " + groupNames[0]);
```

getReportName

Syntax string Parameter.getReportName( )

Returns the name of the report file currently referenced by this Parameter object.

Returns String. The report file name.

Example This example displays an alert box with the name of the report file:

```
alert("Report file: " + paramObj.getReportName( ));
```

getTransientDocumentName

Syntax string Parameter.getTransientDocumentName( )

Returns the name of the transient document generated by running the report currently referenced by this Parameter object.

Returns String.

Example This example displays an alert box with the name of the transient document:

```
alert("Transient document: " +
   paramObj.getTransientDocumentName( ));
```

hideNavBar

Syntax void Parameter.hideNavBar( )

Hides the navigation bar for the parameter component in the LAYOUT_GROUP layout.

Example This example hides the navigation bar:

```
paramObj.hideNavBar( );
alert("Navigation bar is hidden");
```

hideParameterGroup

Syntax void Parameter.hideParameterGroup(string[ ] groupNames)

Hides all report parameters that belongs to a group or to a list of groups.

Parameter **groupNames**
String or array of strings. Hides any groups listed.

Example This example hides the report parameters that belong to the groups that are listed in the myGroups string array:

```
var myGroups = ["Group1", "Group2", "Group3"];
paramObj.hideParameterGroup(myGroups);
alert("Groups are hidden");
```

hideParameterName

Syntax void Parameter.hideParameterName(string[ ] parameterNames)

Hides report parameters as specified by name.

Parameter **parameterNames**
String or array of strings.

Example This example hides the parameters that are listed in the myParams string array:

```
var myParams = ["Parameter1", "Parameter2", "Parameter3"];
paramObj.hideParameterName(myParams);
alert("Parameters are hidden");
```

navigate

Syntax void Parameter.navigate(string containerId, string navTarget)

Changes the current page of the parameter component. The navTarget determines the new location to display the parameter container.

Parameters **containerId**
String. The value of the id parameter for the HTML element that holds the parameter component.

navTarget
String constant. Which navigation button to trigger. Possible values from actuate.parameter.Constants are NAV_FIRST, NAV_PREV, NAV_NEXT, NAV_LAST.

Example This example displays the last page of the parameter component in the HTML element with the myParams ID:

```
function myParamsLast(myParameter){
   myParameter.navigate("myParams", NAV_LAST);
}
```

onUnload

Syntax void Parameter.onUnload( )

Performs garbage collection for the parameter object and unloads JavaScript variables that are no longer needed by Parameter.

Example This example unloads JavaScript variables and displays an alert box:

```
myParameter.onUnload();
alert("JS variables unloaded.");
```

registerEventHandler

Syntax void Parameter.registerEventHandler(actuate.parameter.EventConstants event, function handler)

Registers an event handler to activate for parameter events. This function can assign several handlers to a single event.

Parameters

event
actuate.parameter.EventConstants. A constant corresponding to a supported event. actuate.Parameter supports the following two events:

- actuate.parameter.EventConstants.ON_CHANGED
- actuate.parameter.EventConstants.ON_SELECTION_CHANGED

handler
Function. The function to execute when the event occurs. The handler must take two arguments: the parameter instance that fired the event and an event object specific to the event type.

Example To register an event handler to catch exceptions, call actuate.Parameter.registerEventHandler using the ON_CHANGED constant after creating the viewer object, as shown in the following example:

```
function initParameter( ){
   parameter = new actuate.Parameter("acparameter");
   parameter.registerEventHandler(actuate.parameter.EventConstants
                                  .ON_CHANGED, errorHandler);
}
```

removeEventHandler

Syntax void Parameter.removeEventHandler(actuate.viewer.EventConstants event, function handler)

Removes an event handler.

Parameters **event**
actuate.parameter.EventConstants. A constant corresponding to a supported event. actuate.Parameter supports the following two events:

- actuate.parameter.EventConstants.ON_CHANGED
- actuate.parameter.EventConstants.ON_SELECTION_CHANGED

handler
Function. A handler function registered for the event.

Example To remove an event handler, call actuate.Parameter.removeEventHandler with a legal event constant, as shown in the following example:

```
function cleanupParameter( ){
   parameter.removeEventHandler(actuate.parameter.EventConstants.
                                ON_CHANGED, errorHandler);
}
```

renderContent

Syntax void Parameter.renderContent(actuate.parameter.ParameterDefinition[ ] paramDefs, function callback)

Renders the parameter component to the container.

Parameters **paramDefs**
Array of actuate.parameter.ParameterDefinition objects.

callback
Function. The function to execute after the rendering is done.

Example This example calls renderContent( ) after hiding parameter groups:

```
function showNoGroups(myParameter){
   myParameter.hideParameterGroup(zipcodes);
   myParameter.renderContent(myParameterArray,
                             cleanupParameter(myParameter));
}
```

setAutoSuggestDelay

Syntax void Parameter.setAutoSuggestDelay(long delay)

Sets the autosuggest delay time.

Parameter **delay**
Long. Interpreted as milliseconds.

Example This example implements a custom autosuggest list. The list is 10 suggestions long and displays 3 suggestions at a time after a delay of 250 milliseconds.

```
function myCustomAutoSuggest(myParameter){
   myParameter.setAutoSuggestFetchSize(10);
   myParameter.setAutoSuggestListSize(3);
   myParameter.setAutoSuggestDelay(250);
}
```

setAutoSuggestFetchSize

Syntax void Parameter.setAutoSuggestFetchSize(integer size)

Sets the fetch size of the autosuggestion list. Autosuggest fetches all suggestions from the server when the fetch size is not set.

Parameter **size**
Integer. The number of suggestions to fetch at a time.

Example This example implements a custom autosuggest list. The list is 10 suggestions long and displays 3 suggestions at a time after a delay of 250 milliseconds.

```
function myCustomAutoSuggest(myParameter){
   myParameter.setAutoSuggestFetchSize(10);
   myParameter.setAutoSuggestListSize(3);
   myParameter.setAutoSuggestDelay(250);
}
```

setAutoSuggestListSize

Syntax void Parameter.setAutoSuggestListSize(integer size)

Sets the length of the autosuggestion list. Autosuggest shows all of the suggestions from the server when the list length is not set.

Parameter **size**
Integer. The number of suggestions to display.

Example This example implements a custom autosuggest list. The list is 10 suggestions long and displays 3 suggestions at a time after a delay of 250 milliseconds.

```
function myCustomAutoSuggest(myParameter){
   myParameter.setAutoSuggestFetchSize(10);
   myParameter.setAutoSuggestListSize(3);
   myParameter.setAutoSuggestDelay(250);
}
```

setExpandedGroups

Syntax void Parameter.setExpandedGroups(groupNames)

Defines a set of groups that are expanded by default.

Parameter **groupNames**
Array of strings. The group names to expand by default.

Example This example sets the "Motorcycles", "Trucks", and "Airplanes" groups as expanded by default:

```
var myGroups = new Array["Motorcycles", "Trucks", "Airplanes"];
paramObj.setExpandedGroups(myGroups);
```

setFont

Syntax void Parameter.setFont(string fontStyleString)

Sets the font of the parameter page content after the page is rendered.

Parameter **fontStyleString**
String. The name of a font.

Example This example sets the font to Arial for the parameters page:

```
paramObj.setFont("arial");
```

setGroupContainer

Syntax void Parameter.setGroupContainer(string[ ] groupNames, string containerId)

Sets the HTML element container for the provided group. All parameter objects listed in groupNames are assigned to the container.

Parameters **groupNames**
Array of strings. The group names to be assigned.

containerID
String. The name of the HTML element that displays the group of rendered parameter components.

Example This example assigns the group names in the myGroups string array to the leftpane HTML element:

```
var myGroups = ["Group1", "Group2", "Group3"];
paramObj.setGroupContainer(myGroups, "leftpane");
```

setLayout

Syntax void Parameter.setLayout(string layoutName)

Sets the parameter layout.

Parameter **layoutName**
String constant. Possible values are:

- actuate.parameter.Constants.LAYOUT_GROUP
- actuate.parameter.Constants.LAYOUT_NONE
- actuate.parameter.Constants.LAYOUT_COLLAPSIBLE

Example This example sets the parameter object's layout type to LAYOUT_COLLAPSIBLE:

```
paramObj.setLayout("LAYOUT_COLLAPSIBLE");
```

setReadOnly

Syntax void Parameter.setReadOnly(boolean readOnly)

Sets the parameters to read-only.

Parameter **readOnly**
Boolean. True indicates that the parameters are read-only.

Example This example makes the parameters read-only:

```
paramObj.setReadOnly(true);
```

setReportName

Syntax void Parameter.setReportName(string reportFile)

Sets the report file from which to get report parameters.

Parameter **reportFile**
String. The report file path and name. The report file can be a report design file or a report document file.

Example To set the name using an HTML input tag with an ID of Selector, use the following code:

```
myViewer.setReportName(document.getElementById("Selector").value);
```

setService

Syntax void Parameter.setService(string iPortalURL, actuate.RequestOptions requestOptions)

Sets the target service URL to which the Parameter object links. If the service URL is not set, this Parameter object links to the default service URL set on the actuate object.

Parameters **iPortalURL**
String. The target Actuate web application URL.

requestOptions
actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. The URL can also include custom parameters.

Example This example sets the URL for the Actuate iPortal web application service:

```
paramObj.setService("http://127.0.0.1:8080
                          /iportal",myRequestOptions);
```

setShowDisplayType

Syntax void Parameter.setShowDisplayType(boolean showDisplayType)

Sets whether localized data is shown or not.

Parameter **showDisplayType**
Boolean. True indicates that localized data is shown.

Example This example hides localized data:

```
paramObj.setShowDisplayType(false);
paramObj.submit(alert("Localized data hidden."));
```

submit

Syntax void Parameter.submit(function callback)

Submits requests to the server for the report parameters. When this function is called, an AJAX request is triggered to submit all the operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the parameter container.

Parameter **callback**
Function. The function to execute after the asynchronous call processing is done.

Example This example calls submit( ) after hiding localized data:

```
paramObj.setShowDisplayType(false);
paramObj.submit(alert("Localized data hidden."));
```

Class actuate.parameter.Constants

Description Global constants used for Parameter class. Table 14-14 lists the constants used for the parameter class.

Table 14-14 Actuate iPortal JavaScript API parameter constants

Event	Description
ERR_CLIENT	Constants used to tell JSAPI user that there was a client-side error
ERR_SERVER	Constants used to tell JSAPI user that there was a server-side error
ERR_USAGE	Constants used to tell JSAPI user that there was a usage API error
LAYOUT_COLLAPSIBLE	Constants to set layout of parameter component to collapsible group
LAYOUT_GROUP	Constants to set layout of parameter component to group
LAYOUT_NONE	Constants to set layout of parameter component to none
NAV_FIRST	Constants to programmatically control the first page navigation link
NAV_LAST	Constants to programmatically control the last page navigation link
NAV_NEXT	Constants to programmatically control the next page navigation link
NAV_PREV	Constants to programmatically control the previous page navigation link

Class actuate.parameter.ConvertUtility

Description actuate.parameter.ConvertUtility encodes multiple actuate.parameter.ParameterValue objects into an array of generic objects. For multi-clue or ad hoc parameters, use the array of generic objects as the input parameter for actuate.Viewer.setParameterValues.

Constructor

Syntax actuate.parameter.ConvertUtility(actuate.parameter.ParameterValue[ ] aParamVals)

Constructs a new ConvertUtility object.

Parameter **aParamVals**
Array of actuate.parameter.ParameterValue objects to convert.

Function summary

Table 14-15 lists actuate.parameter.ConvertUtility functions.

Table 14-15 actuate.parameter.ConvertUtility functions

Function	Description
convert( )	Converts the ParameterValues to an array of generic objects
convertDate( )	Converts locale-neutral parameter values to the user's login locale
getParameterMap( )	Returns the ParameterValues as an associative array
getParameterValues( )	Returns an array of ParameterValues

convert

Syntax void ConvertUtility.convert(function callback)

Converts ParameterValues into an array of generic objects. The callback function takes the array as an argument.

Parameter **callback**
Function. The callback function to call after converting the results. The callback function must take the generic array of objects as an argument.

Example This example stores the name-value pair array for myParamValues in a variable called nameValueArray:

```
var nameValueArray = new Array( );
var converter = new actuate.ConvertUtility(myParamValues)
converter.convert(callback);
function callback (values){
   nameValueArray = values;
}
```

convertDate

Syntax void ConvertUtility.convertDate(function callback)

Converts locale-neutral parameter values to the user's login locale.

Parameter **callback**
Function. An optional function to call when this function completes. The callback function receives an array of actuate.parameter.ParameterValue objects as a parameter.

Example This example converts the name-value pair array for myParamValues and stores the results in a variable called nameValueArray:

```
var nameValueArray = new Array( );
var converter = new actuate.ConvertUtility(myParamValues)
converter.convertDate(callback);
function callback (values){
   nameValueArray = values;
}
```

getParameterMap

Syntax object ConvertUtility.getParameterMap( )

Returns the parameters as an associative array. This function makes the name of each parameter an object property and sets the value of that property to the associated parameter value.

Returns Object.

Example This example stores the associative array for myParamValues in a variable called nameValueArray:

```
var paramMap = new Object( );
var converter = new actuate.ConvertUtility(myParamValues)
paramMap = converter.getParameterMap( );
```

getParameterValues

Syntax actuate.parameter.ParameterValue[ ] ConvertUtility.getParameterValues( )

Returns the array of ParameterValue objects.

Returns Array of actuate.parameter.ParameterValue objects.

Example This example stores the array of ParameterValue objects for myParamValues in a variable called paramValues:

```
var paramValues = new Array( );
var converter = new actuate.ConvertUtility(myParamValues)
paramValues = converter.getParameterMap( );
```

Class actuate.parameter.EventConstants

Description Defines the supported event constants for parameters. Table 14-16 lists the parameter event constants.

Table 14-16 Actuate JavaScript API parameter event constants

Event	Description
ON_CHANGE_COMPLETED	Event name triggered when the action is complete and no internal actions are triggered automatically. For example, when a cascading parameter is changed, its child parameter is changed automatically. This event is triggered when its child parameters are updated. The event handler takes the following arguments: ■ actuate.Parameter: parameter component for which the event occurred
ON_CHANGED	Event triggered when a changed event occurs. For example, this event triggers if the value of a parameter control changes. The event handler takes the following arguments: ■ actuate.Parameter: parameter component for which the event occurred
ON_EXCEPTION	Event triggered when an exception occurs. The event handler must take an actuate.Exception object as an input argument. The Exception object contains the exception information.
ON_SELECTION_CHANGED	Event triggered when a selection change occurs. For example, this event triggers if the value of a parameter list control changes. The event handler must take an actuate.Parameter object as an input argument. This input argument is the parameter component for which the event occurred.
ON_SESSION_TIMEOUT	Session time-out event. Whenever a session time-out event occurs and the user tries to perform any operation on parameter component, a prompt dialog will be shown to ask whether the user wants to log in again or not. If the user clicks yes, the ON_SESSION_TIMEOUT event will be fired. If no handler has been registered for this event, a default built-in login dialog will be displayed. The event handler takes the following arguments: ■ actuate.Parameter: component for which the event occurred

Class actuate.parameter.NameValuePair

Description The NameValuePair object contains a display name associated with a value. The actuate.parameterDefinition.setSelectNameValueList( ) function takes an array of actuate.parameter.NameValuePair objects to use in a selection list. In this way, a ParameterDefinition can display a list of names and map them to values used internally. For example, set the name "My Default Country" for a NameValuePair to display "My Default Country" in the drop-down list in the interface, and set the value to "United States" internally for a US user.

Constructor

Syntax actuate.parameter.NameValuePair(string name, string value)

Constructs a new NameValuePair object.

Parameters

name
String. The name to display in the selection list.

value

String. The value that selecting the name sets internally.

Function summary

Table 14-17 lists actuate.parameter.NameValuePair functions.

Table 14-17 actuate.parameter.NameValuePair functions

Function	Description
getName( )	Gets the name for this NameValuePair
getValue( )	Gets the value for this NameValuePair
setName( )	Sets the name for this NameValuePair
setValue( )	Sets the value for this NameValuePair

getName

Syntax string NameValuePair.getName( )

Returns the name for this NameValuePair.

Returns String.

Example This sample code returns the name component of the myNVPair NameValuePair object:

```
alert("Name component is " + myNVPair.getName( ));
```

getValue

Syntax string NameValuePair.getValue( )

Returns the value for this NameValuePair.

Returns String.

Example This sample code returns the value component of the myNVPair NameValuePair object:

```
alert("Value component is " + myNVPair.getValue( ));
```

setName

Syntax void NameValuePair.setName(string name)

Sets the name for the NameValuePair.

Parameter **name**
String.

Example This sample code sets the name component of the myNVPair NameValuePair object to "My hometown":

```
myNVPair.setName("My hometown");
```

setValue

Syntax void NameValuePair.setValue(string value)

Sets the value for the NameValuePair.

Parameter **value**
String.

Example This sample code sets the value component of the myNVPair NameValuePair object to "Cleveland":

```
myNVPair.setValue("Cleveland");
```

Class actuate.parameter.ParameterData

Description The ParameterData class is a high-level wrapper for an actuate.parameter.ParameterDefinition object.

Constructor

Syntax string actuate.parameter.ParameterData(string reportName, actuate.parameter.ParameterDefinition pd)

Constructs a new ParameterData object.

Parameters

reportName
String. The name of the report where the parameter definition originates.

pd
actuate.parameter.ParameterDefinition object. The parameter definition set for this ParameterData object.

Function summary

Table 14-18 lists the actuate.parameter.ParameterData functions.

Table 14-18 actuateparameter.ParameterData functions

Function	Description
getCascadingParentValues( )	Returns the cascading parent value
getChildData( )	Returns the child ParameterData object
getControlType( )	Returns the controlType UI value
getCurrentValue( )	Returns the current UI value set by the UI control
getDefaultValue( )	Returns the default value for this ParameterData object
getHelpText( )	Returns the help text for this ParameterData object
getNameValueList( )	Returns the list of name-value pairs for this ParameterData object
getParameterName( )	Returns the parameter name for this ParameterData object
getParentData( )	Returns the parent ParameterData object
getPickList( )	Returns the pick list for the child ParameterData object

(continues)

Table 14-18 actuateparameter.ParameterData functions (continued)

Function	Description
getPromptText( )	Returns the prompt text for this ParameterData object
getSuggestionList( )	Returns the filter-based suggestion list for this ParameterData object
isAdhoc( )	Returns true when this parameter is ad hoc
isCascadingParameter( )	Returns true when this parameter is a cascading parameter
isDynamicFilter( )	Returns true when this parameter is a dynamic filter
isMultiList( )	Returns true when this parameter is a multi-list
isRequired( )	Returns true when this parameter is required
setChildData( )	Indicates that the parameter data contains a child
setCurrentValue( )	Sets the UI value of the UI control
setParentData( )	Indicates that the parameter data contains a parent
setWebService( )	Defines a web service to send SOAP messages

getCascadingParentValues

Syntax actuate.parameter.ParameterValue[ ]
ParameterData.getCascadingParentValues(
actuate.parameter.ParameterValue[ ] parentValues)

Returns the cascading parent value.

Parameter **parentValues**
An array of actuate.parameter.ParameterValue objects. This array is the one to be populated.

Returns An array of actuate.parameter.ParameterValue objects. This is the input array populated with the cascading parent values.

Example This sample code returns a storage array of actuate.parameter.ParameterValue objects representing the cascading parent values:

```
var parentValues = new Array( );
parentValues = myParamData.getCascadingParentValues(parentValues);
```

getChildData

Syntax actuate.parameter.ParameterData ParameterData.getChildData( )

Returns the child ParameterData object.

Returns actuate.parameter.ParameterData object.

Example This example assigns the child ParameterData object to a myChildData variable:

```
var myChildData = myParameterData.getChildData( );
```

getControlType

Syntax string ParameterData.getControlType( )

Returns the controlType UI value for this ParameterData object.

Returns String. The controlType UI value. Legal controlType UI values are:

- null
- AutoSuggest
- ControlRadioButton
- ControlList
- ControlListAllowNew
- ControlCheckBox

Example This sample code displays the controlType UI value for the myParamData object in an alert box:

```
alert(myParamData.getControlType( ));
```

getCurrentValue

Syntax actuate.parameter.ParameterValue ParameterData.getCurrentValue( )

Returns the current UI value set by the UI control.

Returns actuate.parameter.ParameterValue. Returns null when the UI control has not set a value.

Example This sample code assigns the current UI value to the myCurrVal variable:

```
var myCurrVal = myParameterData.getCurrentValue( );
```

getDefaultValue

Syntax string ParameterData.getDefaultValue( )

Returns the default value for this ParameterData object.

Returns String. The default value. Returns null when the default value is null.

Example This sample code displays the default value for myParamData in an alert box:

```
alert(myParamData.getDefaultValue( ));
```

getHelpText

Syntax string ParameterData.getHelpText( )

Returns the help text for this ParameterData object.

Returns String. The help text.

Example This example displays the help text for the myParamData object in an alert box:

```
alert(myParamData.getHelpText( ));
```

getNameValueList

Syntax actuate.parameter.NameValuePair[ ] ParameterData.getNameValueList( )

Returns the list of name-value pairs for this ParameterData object.

Returns Array of actuate.parameter.NameValuePair objects.

Example This example stores the array of NameValuePair objects for the myParamValues object in a variable called myNVList:

```
var myNVList = new Array( );
myNVList = myParamValues.getNameValueList( );
```

getParameterName

Syntax string ParameterData.getParameterName( )

Returns the parameter name for this ParameterData object.

Returns String. The parameter name.

Example This sample code displays the parameter name for the myParamData object in an alert box:

```
alert(myParamData.getParameterName( ));
```

getParentData

Syntax actuate.parameter.ParameterData ParameterData.getParentData( )

Returns the parent ParameterData object.

Returns actuate.parameter.ParameterData object.

Example This sample code assigns this ParameterData object's parent ParameterData object to the myParentData variable:

```
var myParentData = myParameterData.getParentData( );
```

getPickList

Syntax actuate.parameter.ParameterValue[ ] ParameterData.getPickList(function callback)

Gets the pick list for the child of this parameter data.

Parameter **callback**
Function. An optional function to call when this function completes. This function receives the following parameters:

- An array of actuate.parameter.NameValuePair objects
- An integer that represents the pick list's total leftover count

Returns An array of actuate.parameter.ParameterValue objects.

Example This sample code uses the callback function runNext( ) to display the pick list's total leftover count in an alert box and assigns the array of NameValuePair objects to the pickListNVPairs variable:

```
paramObj.getPickList(runNext);
function runNext(pairs, leftover){
   alert(leftover);
   var pickListNVPairs = new Array( );
   pickListNVPairs = pairs;
}
```

getPromptText

Syntax string ParameterData.getPromptText( )

Returns the prompt text for this ParameterData object.

Returns String. The prompt text.

Example This sample code displays the prompt text for the myParamData object in an alert box:

```
alert(myParamData.getPromptText( ));
```

getSuggestionList

Syntax string[ ] ParameterData.getSuggestionList(function callback, string filter)

Returns the filter-based suggestion list for this ParameterData object.

Parameters

callback
Function. An optional function to call when this function completes. This function receives an array of actuate.parameter.NameValuePair objects as a parameter.

filter
String. The filter for the suggestion list.

Example

This sample code uses the string "Trucks" to call back function runNext( ) to filter the suggestion list and assigns the filtered NameValuePair objects to the mySuggestions variable:

```
paramObj.getSuggestionList(runNext, "Trucks");
function runNext(suggested){
   var mySuggestions = new Array( );
   mySuggestions = suggested;
}
```

isAdhoc

Syntax

boolean ParameterData.isAdhoc( )

Returns true when this parameter is an ad hoc parameter.

Returns

Boolean. True when this parameter is ad hoc.

Example

This example displays the ad hoc status of a parameter in an alert box:

```
alert(paramData.isAdhoc( ));
```

isCascadingParameter

Syntax

boolean ParameterData.isAdhoc( )

Returns true when this parameter is a cascading parameter.

Returns

Boolean. True when this parameter is a cascading parameter.

Example

This example displays the cascading parameter status of a parameter in an alert box:

```
alert(paramData.isCascadingParameter( ));
```

isDynamicFilter

Syntax

boolean ParameterData.isDynamicFilter( )

Returns true when this parameter is a dynamic filter.

Returns

Boolean. True when this parameter is a dynamic filter.

Example This example displays the dynamic filter status of a parameter in an alert box:

```
alert(paramData.isDynamicFilter( ));
```

isMultiList

Syntax boolean ParameterData.isMultiList( )

Returns true when this parameter is shown as a multi-list UI element.

Returns Boolean. True when this parameter is shown as a multi-list UI element.

Example This example displays the multi-list UI element status of a parameter in an alert box:

```
alert(paramData.isMultiList( ));
```

isRequired

Syntax boolean ParameterData.isRequired( )

Returns true when this parameter is required.

Returns Boolean. True when this parameter is required.

Example This example displays the required status of a parameter in an alert box:

```
alert(paramData.isRequired( ));
```

setChildData

Syntax void ParameterData.setChildData(actuate.parameter.ParameterData childData)

Adds a child parameter to this parameter.

Parameter **childData**
An actuate.parameter.ParameterData object that contains the child for this ParameterData object.

Example This sample code sets the ParameterData object myChildData as the child of the ParameterData object myParamData:

```
myParamData.setChildData(myChildData);
```

setCurrentValue

Syntax void ParameterData.setCurrentValue(actuate.parameter.ParameterValue value)

Sets the UI value of the UI control. When a UI value changes, UIControl calls this method to update the ParameterData object.

Parameter **value**
An actuate.parameter.ParameterValue object set by the UI.

Example This sample code sets the ParameterValue object myValue as the value of the ParameterData object myParamData:

```
myParamData.setCurrentValue(myValue);
```

setParentData

Syntax void ParameterData.setParentData(actuate.parameter.ParameterData parentData)

Sets a parent ParameterData object, making this ParameterData object its child.

Parameter **parentData**
An actuate.parameter.ParameterData object that contains the parent for this ParameterData object.

Example This sample code sets the ParameterData object myParentData as the parent of the ParameterData object myParamData:

```
myParamData.setParentData(myParentData);
```

setWebService

Syntax void ParameterData.setWebService(object webService)

Defines a web service to use to send SOAP messages.

Parameter **webService**
Object. A web service to send SOAP messages.

Class actuate.parameter.ParameterDefinition

Description The ParameterDefinition object contains all of the qualities, values, names, and conditions for a parameter. A ParameterDefinition object can display options to the user and respond to user-generated events. The actuate.Parameter class downloads an array of ParameterDefinition objects with downloadParameters( ). The order of this array is also the order in which the parameters are displayed. Parameters can be grouped to divide the parameters on the page into logical sets under a heading.

This class requires significant memory and bandwidth resources. ParameterValue is much smaller than ParameterDefinition. ParameterValue is the more efficient way to communicate to the server that a parameter value has changed.

Constructor

Syntax actuate.parameter.ParameterDefinition( )

Constructs a new ParameterDefinition object.

Function summary

Table 14-19 lists actuate.parameter.ParameterDefinition functions.

Table 14-19 actuate.parameter.ParameterDefinition functions

Function	Description
getAutoSuggestThreshold( )	Gets the auto suggest threshold value for this ParameterDefinition
getCascadingParentName( )	Gets the cascadingParentName value for this ParameterDefinition
getColumnName( )	Gets the columnName value for this ParameterDefinition
getColumnType( )	Gets the columnType value for this ParameterDefinition
getControlType( )	Gets the controlType value for this ParameterDefinition
getCurrentDisplayName( )	Gets the auto suggest current display name for the current value of this ParameterDefinition
getDataType( )	Gets the dataType value for this ParameterDefinition

(continues)

Table 14-19 actuate.parameter.ParameterDefinition functions (continued)

Function	Description
getDefaultValue( )	Gets the defaultValue value for this ParameterDefinition
getDefaultValueIsNull( )	Gets a flag if the default value is null for this ParameterDefinition
getDisplayName( )	Gets the displayName value for this ParameterDefinition
getGroup( )	Gets the group value for this ParameterDefinition
getHelpText( )	Gets the helpText value for this ParameterDefinition
getName( )	Gets the name value for this ParameterDefinition
getOperatorList( )	Gets the list of valid operators
getPosition( )	Gets the position value for this ParameterDefinition
getSelectNameValueList( )	Gets the selectNameValueList value for this ParameterDefinition
getSelectValueList( )	Gets the selectValueList value for this ParameterDefinition
isAdHoc( )	Gets the isAdHoc value for this ParameterDefinition
isHidden( )	Gets the isHidden value for this ParameterDefinition
isPassword( )	Gets the isPassword value for this ParameterDefinition
isRequired( )	Gets the isRequired value for this ParameterDefinition
isViewParameter( )	Gets the isViewParameter value for this ParameterDefinition
setAutoSuggestThreshold( )	Sets the auto suggest threshold value for this ParameterDefinition
setCascadingParentName( )	Sets the cascadingParentName value for this ParameterDefinition
setColumnName( )	Sets the columnName value for this ParameterDefinition
setColumnType( )	Sets the columnType value for this ParameterDefinition

Table 14-19 actuate.parameter.ParameterDefinition functions (continued)

Function	Description
setControlType( )	Sets the controlType value for this ParameterDefinition
setCurrentDisplayName( )	Sets the current display name for this ParameterDefinition
setDataType( )	Sets the dataType value for this ParameterDefinition
setDefaultValue( )	Sets the defaultValue value for this ParameterDefinition
setDefaultValueIsNull( )	Sets the defaultValue to null for this ParameterDefinition
setDisplayName( )	Sets the displayName value for this ParameterDefinition
setGroup( )	Sets the group value for this ParameterDefinition
setHelpText( )	Sets the helpText value for this ParameterDefinition
setIsAdHoc( )	Sets the isAdHoc value for this ParameterDefinition
setIsHidden( )	Sets the isHidden value for this ParameterDefinition
setIsMultiSelectControl( )	Sets the isMultiSelectControl value for this ParameterDefinition
setIsPassword( )	Sets the isPassword value for this ParameterDefinition
setIsRequired( )	Sets the isRequired value for this ParameterDefinition
setIsViewParameter( )	Sets the isViewParameter value for this ParameterDefinition
setName( )	Sets the name value for this ParameterDefinition
setPosition( )	Sets the position value for this ParameterDefinition
setSelectNameValueList( )	Sets the selectNameValueList value for this ParameterDefinition
setSelectValueList( )	Sets the selectValueList value for this ParameterDefinition

getAutoSuggestThreshold

Syntax integer ParameterDefinition.getAutoSuggestThreshold( )

Gets the autosuggest threshold value for this ParameterDefinition. The autosuggest threshold determines the number of characters a user types in before they are given suggestions from autosuggest.

Returns Integer.

Example To store the autosuggest threshold of the parameter definition paramdef in a variable called threshold, use code similar to the following:

```
var threshold = paramdef.getAutoSuggestThreshold( );
```

getCascadingParentName

Syntax string ParameterDefinition.getCascadingParentName( )

Gets the cascadingParentName value for this ParameterDefinition. A cascading parent parameter is only used when one parameter depends upon another.

Returns String.

Example To store the cascading parent name of the parameter definition paramdef in a variable called parentname, use code similar to the following:

```
var parentname = paramdef.getCascadingParentName( );
```

getColumnName

Syntax string ParameterDefinition.getColumnName( )

Gets the columnName value for this ParameterDefinition. This setting sets the column to retrieve data from for an ad hoc parameter that performs a query.

This setting has no effect on other types of parameters.

Returns String.

Example To store the column name of the parameter definition paramdef in a variable called columnname, use code similar to the following:

```
var columnname = paramdef.getColumnName( );
```

getColumnType

Syntax string ParameterDefinition.getColumnType( )

Gets the columnType value for this ParameterDefinition. This setting sets the data type queried by an ad hoc parameter that performs a query.

This setting has no effect on other types parameters.

Returns String. Possible values are: null, "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To store the column type of the parameter definition paramdef in a variable called columntype, use code similar to the following:

```
var columntype = paramdef.getColumnType( );
```

getControlType

Syntax string ParameterDefinition.getControlType( )

Gets the controlType value for this ParameterDefinition. It determines the form element displayed for the user to set the parameter value.

Returns String. Possible values are: null, "", "ControlRadioButton", "ControlList", "ControlListAllowNew", and "ControlCheckBox".

Example To store the control type string for the parameter definition paramdef in a variable called controltype, use code similar to the following:

```
var controltype = paramdef.getControlType( );
```

getCurrentDisplayName

Syntax string ParameterDefinition.getCurrentDisplayName( )

Gets the current display name for this ParameterDefinition.

Returns String.

Example To store the current display name of the parameter definition paramdef in a variable called displayname, use code similar to the following:

```
var displayname = paramdef.getDisplayName( );
```

getDataType

Syntax string ParameterDefinition.getDataType( )

Gets the dataType value for this ParameterDefinition.

Returns String. Possible values are: "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To store the data type of the parameter definition paramdef in a variable called type, use code similar to the following:

```
var type = paramdef.getDataType( );
```

getDefaultValue

Syntax string ParameterDefinition.getDefaultValue( )

Gets the defaultValue value for this ParameterDefinition, if applicable.

Returns String.

Example To store the default value as a string for the parameter definition paramdef in a variable called default, use code similar to the following:

```
var default = paramdef.getDefaultValue( );
```

getDefaultValueIsNull

Syntax boolean ParameterDefinition.getDefaultValueIsNull( )

Returns true when the parameter's default value is null.

Returns Boolean.

Example To alert the user that the default value is null for the parameter definition paramdef, use code similar to the following:

```
if (paramdef.getDefaultValueIsNull( )){
   alert('Default value is null!');
}
```

getDisplayName

Syntax string ParameterDefinition.getDisplayName( )

Gets the displayName for this ParameterDefinition.

Returns String.

Example To store the displayed name for the parameter definition paramdef in a variable called displayname, use code similar to the following:

```
var displayname = paramdef.getDisplayName( );
```

getGroup

Syntax string ParameterDefinition.getGroup( )

Gets the group for this ParameterDefinition, indicating if it is a member of a group.

Returns String. A group name, or null if there is no group.

Example To print the group name for the parameter definition paramdef to the current document, use code similar to the following:

```
document.write(paramdef.getGroup( ));
```

getHelpText

Syntax string ParameterDefinition.getHelpText( )

Gets the helpText for this ParameterDefinition.

Returns String. The help text.

Example To store the help text for the parameter definition paramdef in a variable called helptext, use code similar to the following:

```
var helptext = paramdef.getHelpText( );
```

getName

Syntax string ParameterDefinition.getName( )

Gets the name for this ParameterDefinition.

Returns String. The parameter name.

Example To store the name for the parameter definition paramdef in a variable called paramname, use code similar to the following:

```
var paramname = paramdef.getName( );
```

getOperatorList

Syntax string[ ] ParameterDefinition.getOperatorList( )

Gets the operator list for this ParameterDefinition.

Returns An array of strings containing the operator list.

Example To store the list of operators for the parameter definition paramdef in a variable called ops, use code similar to the following:

```
var ops = new Array( );
ops = paramdef.getOperatorList( );
```

getPosition

Syntax Integer ParameterDefinition.getPosition( )

Gets the position in the array for this ParameterDefinition.

Returns Integer.

Example To store the position of the parameter definition paramdef in a variable called position, use code similar to the following:

```
var position = paramdef.getPosition( );
```

getSelectNameValueList

Syntax selectNameValueList[ ] ParameterDefinition.getSelectNameValueList( )

Gets the selectNameValueList for this ParameterDefinition. This list applies if the parameter is set with a selection list.

Returns Array of actuate.parameter.NameValuePair objects.

Example To retrieve the name-value pair list for the parameter definition paramdef and put it into a new array, use code similar to the following:

```
var namevalues = new array( );
namevalues = paramdef.getSelectNameValueList( ).slice( );
```

getSelectValueList

Syntax string[ ] ParameterDefinition.getSelectValueList( )

Gets the selectValueList for this ParameterDefinition. This list applies when the parameter is set with a selection list.

Returns An array of strings containing the select value list.

Example To retrieve the list of values selectable for the parameter definition paramdef and put it into a new array, use code similar to the following:

```
var selectvalues = new array( );
selectvalues = paramdef.getSelectValueList( ).slice( );
```

isAdHoc

Syntax boolean ParameterDefinition.isAdHoc( )

Gets the isAdHoc for this ParameterDefinition.

Returns Boolean. True indicates that this parameter is an ad hoc parameter.

Example To set the default value to null for the parameter definition paramdef if it is an ad hoc parameter, use code similar to the following:

```
if (paramdef.isAdHoc( )){
   paramdef.setDefaultValueIsNull(true);
}
```

isHidden

Syntax boolean ParameterDefinition.isHidden( )

Gets the isHidden value for this ParameterDefinition.

Returns Boolean. True indicates that this parameter is hidden.

Example To reveal a parameter with the parameter definition paramdef if it is hidden, use code similar to the following:

```
if (paramdef.isHidden( )){
   paramdef.setIsHidden(false);
}
```

isPassword

Syntax boolean ParameterDefinition.isPassword( )

Gets the isPassword value for this ParameterDefinition.

Returns Boolean. True indicates that the parameter is a password.

Example To set the parameter definition paramdef as required if it is a password parameter, use code similar to the following:

```
if (paramdef.isPassword( )){
   paramdef.setIsRequired(true);
}
```

isRequired

Syntax boolean ParameterDefinition.isRequired( )

Gets the isRequired value for this ParameterDefinition.

Returns Boolean. True indicates that the parameter is required.

Example To set specific help text for the parameter definition paramdef if it is a required parameter, use code similar to the following:

```
if (paramdef.isRequired( )){
   paramdef.setHelpText("This parameter is required.");
}
```

isViewParameter

Syntax boolean ParameterDefinition.isViewParameter( )

Gets the isViewParameter value for this ParameterDefinition.

Returns Boolean. True indicates that the parameter is a view-time parameter. False indicates that the parameter is a run-time parameter.

Example To set specific help text for the parameter definition paramdef if it is a view-time parameter, use code similar to the following:

```
if (paramdef.isViewParameter( )){
   paramdef.setHelpText("This is a view-time parameter.");
}
```

setAutoSuggestThreshold

Syntax void ParameterDefinition.setAutoSuggestThreshold(integer threshold)

Sets the autosuggest threshold for this ParameterDefinition. The autosuggest threshold determines the number of characters a user types in before they are given suggestions from autosuggest.

Parameter **threshold**
Integer.

Example To always show the autosuggest dialog for the parameter definition paramdef, use code similar to the following:

```
paramdef.setAutoSuggestThreshold(0);
```

setCascadingParentName

Syntax void ParameterDefinition.setCascadingParentName(string cascadingParentName)

Sets the cascadingParentName for this ParameterDefinition. This sets another parameter as this parameter's parent.

Parameter **cascadingParentName**
String.

Example To set the parent name of the parameter definition paramdef to "Clark", use code similar to the following:

```
paramdef.setCascadingParentName("Clark");
```

setColumnName

Syntax void ParameterDefinition.setColumnName(string columnName)

Sets the columnName for this ParameterDefinition. Used for queries.

Parameter **columnName**
String.

Example To set the parameter definition paramdef to access the ProductName column, use code similar to the following:

```
paramdef.setColumnName("ProductName");
```

setColumnType

Syntax void ParameterDefinition.setColumnType(string columnType)

Sets the columnType for this ParameterDefinition. Used for queries.

Parameter **columnType**
String. Possible values are null, "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To allow the parameter definition paramdef to interpret a column as untyped data, use code similar to the following:

```
paramdef.setColumnType("Unknown");
```

setControlType

Syntax void ParameterDefinition.setControlType(string controlType)

Sets the control type of this ParameterDefinition.

Parameter **controlType**
String. Possible values are null, "", "AutoSuggest", "ControlRadioButton", "ControlList", "ControlListAllowNew", and "ControlCheckBox".

Example To set the parameter definition paramdef to use a control list, use code similar to the following:

```
paramdef.setControlType("ControlList");
```

setCurrentDisplayName

Syntax void ParameterDefinition.setCurrentDisplayName(string currentDiplayName)

Sets the displayed name for this parameter.

Parameter **currentDisplayName**
String.

Example To set the display name for the parameter definition paramdef to "Year", use code similar to the following:

```
paramdef.setCurrentDisplayName("Year");
```

setDataType

Syntax void ParameterDefinition.setDataType(string dataType)

Sets the dataType for this ParameterDefinition.

Parameter **dataType**
String. Possible values are "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To set the parameter definition paramdef data type to date, use code similar to the following:

```
paramdef.setDataType("Date");
```

setDefaultValue

Syntax void ParameterDefinition.setDefaultValue(string defaultValue)

Sets the default value for this ParameterDefinition.

Parameter **defaultValue**
String.

Example To set the default value of parameter definition paramdef to "2010", use code similar to the following:

```
paramdef.setDefaultValue("2010");
```

setDefaultValueIsNull

Syntax void ParameterDefinition.setDefaultValue(boolean value)

When true, sets the default value for this ParameterDefinition to null. Sets the default value to no value in all other cases.

Parameter **value**
Boolean.

Example To set the default value of parameter definition paramdef to null, use code similar to the following:

```
paramdef.setDefaultValueIsNull(true);
```

setDisplayName

Syntax void ParameterDefinition.setDisplayName(string displayName)

Sets the name to display on the parameter page for this ParameterDefinition.

Parameter **displayName**
String.

Example To set the displayed name of parameter definition paramdef to "Year", use code similar to the following:

```
paramdef.setDisplayName("Year");
```

setGroup

Syntax void ParameterDefinition.setGroup(string group)

Sets the group value for this ParameterDefinition.

Parameter **group**
String.

Example To assign the parameter definition paramdef to the "Customer Details" parameter group, use code similar to the following:

```
paramdef.setGroup("Customer Details");
```

setHelpText

Syntax void ParameterDefinition.setHelpText(string helpText)

Sets the helpText value for this ParameterDefinition.

Parameter **helpText**
String.

Example To set specific help text for the parameter definition paramdef if it is a required parameter, use code similar to the following:

```
if (paramdef.isRequired( )){
   paramdef.setHelpText("This parameter is required.");
}
```

setIsAdHoc

Syntax void ParameterDefinition.setIsAdHoc(boolean isAdHoc)

Sets this parameter as an ad hoc parameter.

Parameter **isAdHoc**
Boolean. True sets this parameter to ad hoc.

Example To enable the parameter definition paramdef to accept an ad hoc value, use code similar to the following:

```
paramdef.setIsAdHoc(true);
```

setIsHidden

Syntax void ParameterDefinition.setIsHidden(boolean isHidden)

Sets the parameter to hidden.

Parameter **isHidden**
Boolean. True hides the parameter.

Example To hide a parameter defined by a parameter definition called paramdef, use code similar to the following:

```
paramdef.setIsHidden(true);
```

setIsMultiSelectControl

Syntax void ParameterDefinition.setIsMultiSelectControl(boolean isMultiSelect)

Sets the parameter to accept multiple selected values.

Parameter **isMultiSelect**
Boolean. True allows multiple selected values to be set for this parameter.

Example To allow a parameter defined by a parameter definition called paramdef to accept multiple selected values, use code similar to the following:

```
paramdef.setIsMultiSelectControl(true);
```

setIsPassword

Syntax void ParameterDefinition.setIsPassword(boolean isPassword)

Sets this parameter to treat its value as a password, which hides the input on the page and encrypts the value.

Parameter **isPassword**
Boolean. True indicates a password value.

Example To set the parameter type accepted by the parameter definition paramdef to password, use code similar to the following:

```
paramdef.setIsPassword(true);
```

setIsRequired

Syntax void ParameterDefinition.setIsRequired(boolean isRequired)

Sets the parameter to required.

Parameter **isRequired**
Boolean. True indicates a mandatory parameter.

Example To make the parameter defined by the parameter definition paramdef mandatory, use code similar to the following:

```
paramdef.setIsRequired(true);
```

setIsViewParameter

Syntax void ParameterDefinition.setIsViewParameter(boolean isViewParameter)

Sets the isViewParameter value for this ParameterDefinition.

Parameter **isViewParameter**
Boolean.

Example To make the parameter defined by the parameter definition paramdef a view-time parameter, use code similar to the following:

```
paramdef.setIsViewParameter(true);
```

setName

Syntax void ParameterDefinition.setName(string name)

Sets the name to use internally for this ParameterDefinition.

Parameter **name**
String.

Example To set the internal name of the parameter definition paramdef to Year, use code similar to the following:

```
paramdef.setName("Year");
```

setPosition

Syntax void ParameterDefinition.setPosition(integer position)

Sets the position value for this ParameterDefinition. The index indicates the position in the array of the ParameterDefinition.

Parameter **position**
Integer.

Example To shift the parameter definition paramdef down on position in the parameter array, use code similar to the following:

```
paramdef.setPosition(++paramdef.getPosition( ));
```

setSelectNameValueList

Syntax void ParameterDefinition.setSelectNameValueList
(actuate.parameter.NameValuePair[ ] selectNameValueList)

Sets the selectNameValueList value for this ParameterDefinition.

Parameter **selectNameValueList**
Array of actuate.parameter.NameValuePair objects.

Example To set the parameter definition paramdef to select the same name-value list as the parameter definition nparam, use code similar to the following:

```
paramdef.setSelectNameValueList(nparam.getSelectNameValueList( ));
```

setSelectValueList

Syntax void ParameterDefinition.setSelectValueList(array[ ] selectValueList)

Sets the selectValueList value for this ParameterDefinition.

Parameter **selectValueList**
Array.

Example To set the parameter definition paramdef to select the values 2007-2009, use code similar to the following:

```
var values = new Array("2007", "2008", "2009");
paramdef.setSelectValueList(values);
```

Class actuate.parameter.ParameterValue

Description ParameterValue is a container for the value of Parameter to be passed to a report for processing. When a user sets a value in the interface, the corresponding ParameterValue must change.

Because ParameterValue is much smaller that ParameterDefinition, it is the recommended means of communicating to the server that a parameter value has changed or passing a parameter value to a viewer element. Sending an entire ParameterDefinition has a larger effect on system performance.

Constructor

Syntax actuate.parameter.ParameterValue( )

Constructs a new ParameterValue object.

Function summary

Table 14-20 lists actuate.parameter.ParameterValue functions.

Table 14-20 actuate.parameter.ParameterValue functions

Function	Description
getColumnName( )	Gets the name of the column in this ParameterValue
getColumnType( )	Gets the data type value of the column for this ParameterValue
getDataType( )	Gets the dataType value for this ParameterValue
getDisplayName( )	Gets the displayed name for this ParameterValue
getGroup( )	Gets the group value for this ParameterValue
getName( )	Gets the name value for this ParameterValue
getPosition( )	Gets the position value for this ParameterValue
getPromptParameter( )	Gets the promptParameter value for this ParameterValue
getValue( )	Gets the value or values for this ParameterValue
getValueIsNull( )	Gets the valueIsNull value for this ParameterValue
isViewParameter( )	Gets the isViewParameter value for this ParameterValue

(continues)

Table 14-20 actuate.parameter.ParameterValue functions (continued)

Function	Description
setColumnName( )	Sets the name of the column in this ParameterValue
setColumnType( )	Sets the data type value of the column for this ParameterValue
setDataType( )	Sets the dataType value for this ParameterValue
setDisplayName( )	Sets the displayed name for this ParameterValue
setGroup( )	Sets the group value for this ParameterValue
setIsViewParameter( )	Sets the isViewParameter value for this ParameterValue
setName( )	Sets the name value for this ParameterValue
setPosition( )	Sets the position value for this ParameterValue
setPromptParameter( )	Sets the promptParameter value for this ParameterValue
setValue( )	Sets the value for this ParameterValue
setValueIsNull( )	Sets the valueIsNull value for this ParameterValue

getColumnName

Syntax string ParameterValue.getColumnName( )

Gets the column name value for this ParameterValue. Columns are supported as part of ad hoc parameters.

Returns String. The name of the column.

Example To store the column name for the parameter value pvalue in a variable called columnname, use code similar to the following:

```
var columnname = pvalue.getColumnName( );
```

getColumnType

Syntax string ParameterValue.getColumnType( )

Gets the data type value of the column for this ParameterValue. Columns are supported as part of ad hoc parameters.

Returns String. Possible values are null, "", "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To store the column type for the parameter value pvalue in a variable called columntype, use code similar to the following:

```
var columntype = pvalue.getColumnType( );
```

getDataType

Syntax string ParameterValue.getDataType( )

Gets the dataType value for this ParameterValue.

Returns String. Possible values are null, "", "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To store the data type for the parameter value pvalue in a variable called type, use code similar to the following:

```
var type = pvalue.getDataType( );
```

getDisplayName

Syntax string ParameterValue.getDisplayName( )

Gets the displayed name for this ParameterValue.

Returns String. The displayed name

Example To store the displayed name of the parameter value pvalue in a variable called displayedName, use code similar to the following:

```
var displayedName = pvalue.getDisplayName( );
```

getGroup

Syntax string ParameterValue.getGroup( )

Gets the group value for this ParameterValue.

Returns String.

Example To store the group that the parameter value pvalue belongs to in a variable called group, use code similar to the following:

```
var group = pvalue.getGroup( );
```

getName

Syntax string ParameterValue.getName( )

Gets the name value for this ParameterValue.

Returns String.

Example To store the name of the parameter value pvalue in a variable called name, use code similar to the following:

```
var name = pvalue.getName( );
```

getPosition

Syntax integer ParameterValue.getPosition( )

Gets the position value for this ParameterValue.

Returns Integer.

Example To save the position of the parameter value pvalue in the parameter list to a variable called pos, use code similar to the following:

```
var pos = pvalue.getPosition( );
```

getPromptParameter

Syntax boolean ParameterValue.getPromptParameter( )

Gets the promptParameter value for this ParameterValue.

Returns Boolean.

Example To store the prompt parameter of the parameter value pvalue in a variable called prompt, use code similar to the following:

```
var prompt = pvalue.getPromptParameter( );
```

getValue

Syntax string[ ] ParameterValue.getValue( )

Gets the value values for this ParameterValue.

Returns String or array of strings. The value or values of this ParameterValue object.

Example To store the value of the parameter value pvalue in a variable called value, use code similar to the following:

```
var value = pvalue.getValue( );
```

getValueIsNull

Syntax boolean ParameterValue.getValueIsNull( )

Gets the valueIsNull value for this ParameterValue.

Returns Boolean. True indicates that this ParameterValue is null.

Example To alert the user that the value of the parameter value pvalue is null, use code similar to the following:

```
if (pavalue.getValueIsNull( )){
   alert('Default value is null!');
}
```

isViewParameter

Syntax boolean ParameterValue.isViewParameter( )

Gets the isViewParameter value for this ParameterValue.

Returns Boolean. True indicates that this ParameterValue is visible.

Example To set specific help text for the parameter value pvalue if it is a view-time parameter, use code similar to the following:

```
if (pvalue.isViewParameter( )){
   pvalue.setHelpText("This is a view-time parameter.");
}
```

setColumnName

Syntax void ParameterValue.setColumnName(string columnName)

Sets the column name value for this ParameterValue.

Parameter **columnName**
String. The name of the column.

Example To set the column name for the parameter value pvalue to Year, use code similar to the following:

```
pvalue.setColumnName("Year");
```

setColumnType

Syntax void ParameterValue.setColumnType(string columnType)

Sets the data type of the column for this ParameterValue. Used for queries.

Parameter **columnType**
String. Possible values are "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To set the column type for the parameter value pvalue to Date, use code similar to the following:

```
pvalue.setColumnType("Date");
```

setDataType

Syntax void ParameterValue.setDataType(string dataType)

Sets the dataType value for this ParameterValue.

Parameter **dataType**
String. Possible values are "Currency", "Date", "DateOnly", "Time", "Double", "Integer", "String", "Boolean", "Structure", "Table", and "Unknown".

Example To set the data type for the parameter value pvalue to Date, use code similar to the following:

```
pvalue.setDataType("Date");
```

setDisplayName

Syntax void ParameterValue.setDisplayName(string name)

Sets the displayed name value for this ParameterValue.

Parameter **name**
String. A displayed parameter name.

Example To set the display name of the parameter value pvalue to Year, use code similar to the following:

```
pvalue.setDisplayName("Year");
```

setGroup

Syntax void ParameterValue.setGroup(string group)

Sets the group value for this ParameterValue.

Parameter **group**
String. The name of the group.

Example To set the group for the parameter value pvalue to Customer Details, use code similar to the following:

```
pvalue.setGroup("Customer Details");
```

setIsViewParameter

Syntax void ParameterValue.setIsViewParameter(boolean isViewParameter)

Sets the isViewParameter value for this ParameterValue.

Parameter **isViewParameter**
Boolean. True indicates a view-time parameter.

Example To make the parameter value pvalue into a view-time parameter, use code similar to the following:

```
pvalue.setIsViewParameter(true);
```

setName

Syntax void ParameterValue.setName(string name)

Sets the name value for this ParameterValue.

Parameter **name**
String. A parameter name.

Example To set the name of the parameter value pvalue to Year, use code similar to the following:

```
pvalue.setName("Year");
```

setPosition

Syntax void ParameterValue.setPosition(integer position)

Sets the position value for this ParameterValue.

Parameter **position**
Integer. The position from the top of the parameter list.

Example To move the parameter value pvalue one place farther down in the parameter list, use code similar to the following:

```
pvalue.setPosition(++pvalue.getPosition( ));
```

setPromptParameter

Syntax void ParameterValue.setPromptParameter(boolean promptParameter)

Sets the promptParameter value for this ParameterValue.

Parameter **promptParameter**
Boolean. True indicates that this parameter prompts the user.

Example To set the parameter value pvalue to not prompt the user, use code similar to the following:

```
pvalue.setPromptParameter(false);
```

setValue

Syntax void ParameterValue.setValue(string[ ] value)

Sets the value or values for this ParameterValue.

Parameter **value**
String or array of strings. The value or values of this ParameterValue object.

Example To set the value of the parameter value pvalue to 2010, use code similar to the following:

```
pvalue.setValue("2010");
```

To set the values of the ParameterValue object pvalues to 2008, 2009, and 2010, use code similar to the following:

```
pvalue.setValue({"2008", "2009", "2010"});
```

setValueIsNull

Syntax void ParameterValue.setValueIsNull(boolean valueIsNull)

Sets the valueIsNull value for this ParameterValue.

Parameter **valueIsNull**
Boolean. True indicates that this ParameterValue is null.

Example To set the value of parameter value pvalue to null, use code similar to the following:

```
pvalue.setValueIsNull(true);
```

Class actuate.report.Chart

Description Provides functions to operate on a chart element, such as changing its format or retrieving data from specific elements.

Constructor

The actuate.report.Chart object is created when actuate.viewer.PageContent.getChartByBookmark( ) is called.

Function summary

Table 14-21 lists actuate.report.Chart functions.

Table 14-21 actuate.report.Chart functions

Function	Description
clearFilters( )	Clears the filters applied to the given column
drillDownCategory( )	Drills down into a chart by category
drillDownSeries( )	Drills down into a chart by series
drillUpCategory( )	Drills up one level by category
drillUpSeries( )	Drills up one level by series
getBookmark( )	Returns the report element bookmark name
getClientChart( )	Returns an HTML5 instance of this chart
getHtmlDom( )	Returns the HTML element DOM object
getInstanceId( )	Returns the report element instance id
getPageContent( )	Returns the page content to which this element belongs
getType( )	Returns the report element type
hide( )	Hides this element
setChartTitle( )	Sets the title for this chart
setDimension( )	Sets the number of dimensions for the chart element
setFilters( )	Applies filters to this chart element
setSize( )	Sets the width and height of the chart element
setSubType( )	Sets a chart subtype to the chart element
show( )	Shows this element

(continues)

Table 14-21 actuate.report.Chart functions (continued)

Function	Description
submit( )	Submits all the asynchronous operations that the user has requested on this report and renders the chart component on the page

clearFilters

Syntax void Chart.clearFilters(string columnName)

Clears the filters for a given column.

Parameter **columnName**
String. The name of the column.

Example This example clears existing filters from the PRODUCTLINE column of a chart and changes the chart title:

```
function resetFilter(bchart){
   bchart.clearFilters("PRODUCTLINE");
   bchart.setChartTitle("Orders By Country");
   bchart.submit( );
}
```

drillDownCategory

Syntax void Chart.drillDownCategory(string categoryData)

Drills down into a chart by category.

Parameter **categoryData**
String. The name of the data category to drill down to.

drillDownSeries

Syntax void Chart.drillDownSeries(string seriesName)

Drills down into a chart by series.

Parameter **seriesName**
String. The name of the data series to drill down to.

drillUpCategory

Syntax void Chart.drillUpCategory( )

Drills up into a chart by one data category level.

drillUpSeries

Syntax void Chart.drillUpSeries( )

Drills up into a chart by one series level.

getBookmark

Syntax string Chart.getBookmark( )

Returns the chart's bookmark name.

Returns String. The chart's bookmark name.

Example This example sets the chart's title to the bookmark name:

```
function titleBookmark(bchart){
   bchart.setChartTitle(bchart.getBookmark( ));
   bchart.submit( );
}
```

getClientChart

Syntax actuate.report.HTML5Chart.ClientChart Chart.getClientChart( )

Returns the HTML5 Chart instance if this chart has an HTML5 Chart output format, otherwise returns null.

Returns actuate.report.HTML5Chart.ClientChart. The HTML5 formatted chart or null.

Example This example displays the chart ID of the HTML5 chart in an alert box:

```
function showHTML5ChartID(myChart){
   var myHTML5Chart = myChart.getClientChart( );
   var HTML5ChartID = myHTML5Chart.getViewerId( );
   alert (HTML5ChartID);
}
```

getHtmlDom

Syntax HTMLElement Chart.getHtmlDom( )

Returns the HTML element for this chart.

Returns HTMLElement. The HTML DOM element.

Example This example displays the HTML DOM element for this chart inside a red border:

```
function showHtmlDom(myChart){
   var domNode = myChart.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string Chart.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myChart){
   var elementID = myChart.getInstanceId( );
   alert (elementID);
}
```

getPageContent

Syntax actuate.viewer.PageContent Chart.getPageContent( )

Returns the content of the page to which this chart belongs.

Returns actuate.report.PageContent. The report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myChart){
   var pageContent = myChart.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getType

Syntax string Chart.getType( )

Returns the chart's report element type.

Returns String. This method returns the string "Chart" when the type is actuate.report.Chart.CHART and the string "Flash Chart" when the type is actuate.report.Chart.FLASH_CHART.

Example This example displays the chart type in an alert box:

```
alert ("Chart is of type " + myChart.getType( ));
```

hide

Syntax void Chart.hide( )

Hides this element.

Example To hide the chart bchart, use code similar to the following:

```
alert("Hiding chart" + bchart.getBookmark( ));
bchart.hide( );
bchart.submit( );
```

setChartTitle

Syntax void Chart.setChartTitle(string title)

Sets the title for this chart element.

Parameter **title**
String. The title for the chart.

Example This example sets the chart's title to the bookmark name:

```
function titleBookmark(bchart){
   bchart.setChartTitle(bchart.getBookmark( ));
   bchart.submit( );
}
```

setDimension

Syntax void Chart.setDimension(actuate.report.Chart dimension)

Sets the number of dimensions for the chart element. The chart dimension only works if supported by the chart's type. A 3D chart does not support multiple value axes. Remove all of the *y*-axes after the first before converting a chart to 3D.

Parameter **dimension**
actuate.report.Chart. The number of dimensions in which to display the chart element. Supported values are 2D and 2D with depth. The constants defined for this argument are:

- actuate.report.Chart.CHART_DIMENSION_2D
- actuate.report.Chart.CHART_DIMENSION_2D_WITH_DEPTH

Example This example changes the chart bchart's dimension to 2D with depth:

```
bchart.setChartTitle(bchart.getBookmark( ) + ": 2D with Depth");
bchart.setDimension(actuate.report.Chart.CHART_DIMENSION_2D_WITH_
   DEPTH );
bchart.submit( );
```

setFilters

Syntax void Chart.setFilters(actuate.data.Filter filter)

void Chart.setFilters(actuate.data.Filter[ ] filters)

Applies filters to this chart element. To apply more than one filter to a chart element, call this function multiple times, once for each filter object.

Parameters

filter
An actuate.data.Filter object. A single filter condition to apply to this chart element.

filters
An array of actuate.data.Filter objects. Filter conditions to apply to this chart element.

Example This example applies a filter to the chart and changes the chart's title to reflect the filter:

```
function chartFilter(bchart){
   var filter = new actuate.data.Filter("PRODUCTLINE", "=",
                                        "Trucks and Buses");
   var filters = new Array( );
   filters.push(filter);

   bchart.setFilters(filters);
   bchart.setChartTitle("Orders By Country (Trucks and Buses)");
   bchart.submit( );
}
```

setSize

Syntax void Chart.setSize(integer width, integer height)

Sets the width and height of the chart element displayed.

Parameters

width
Integer. The width in pixels.

height
Integer. The height in pixels.

Example To set the chart bchart to be 600 pixels wide by 800 pixels high, use code similar to the following:

```
alert("Resizing " + bchart.getBookmark( ) + " to 600x800");
bchart.setSize(600,800);
bchart.submit( );
```

setSubType

Syntax void Chart.setSubType(string chartType)

Sets a subtype for this chart element. When the report calls submit( ), the report redraws the chart element as the requested type.

Parameter **chartType**
String. The format in which to redraw the chart element. The constants that define the chart subtypes are:

- CHART_SUBTYPE_PERCENTSTACKED
- CHART_SUBTYPE_SIDEBYSIDE
- CHART_SUBTYPE_STACKED

Example To change the subtype of the chart bchart to side-by-side, use code similar to the following:

```
bchart.setChartTitle("Side by Side Chart");
bchart.setSubType(actuate.report.Chart.CHART_SUBTYPE_SIDEBYSIDE);
bchart.submit( );
```

show

Syntax void Chart.show( )

Shows this element.

Example To reveal the hidden chart bchart, use code similar to the following:

```
alert("Showing chart" + bchart.getBookmark( ));
bchart.show( );
bchart.submit( );
```

submit

Syntax void Chart.submit(function callback)

Submits all the asynchronous operations for this chart. The submit( ) function triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the chart container.

Parameter **callback**
Function. Optional. A function to execute after the asynchronous call processing is done. Submit passes the current actuate.Viewer object to the callback as an input parameter.

Example This example sets the chart's title to the bookmark name and pops up an alert box after calling submit( ):

```
function titleBookmark(bchart){
   bchart.setChartTitle(bchart.getBookmark( ));
   bchart.submit(alert("Title Changed"));
}
```

Class actuate.report.DataItem

Description A container for a data element in a report. DataItem provides functions to operate on a data element, such as retrieving the data value and getting the HTML DOM element from the report data element.

Constructor

The DataItem object is constructed by actuate.viewer.PageContent.getDataItemByBookmark( ).

Function summary

Table 14-22 lists actuate.report.DataItem functions.

Table 14-22 actuate.report.DataItem functions

Function	Description
getBookmark( )	Returns the bookmark name for this data item
getData( )	Returns the data value on this data element
getHtmlDom( )	Returns the HTML element for this data item
getInstanceId( )	Returns the instance id of this report element.
getPageContent( )	Returns the page content to which this element belongs
getType( )	Returns the report element type
hide( )	Hides this element
show( )	Shows this element
submit( )	Applies the changes made to this DataItem

getBookmark

Syntax string DataItem.getBookmark( )

Returns the bookmark name for this data item.

Returns String.

Example This example displays the data item's bookmark in an alert box:

```
alert(myDataItem.getBookmark( ));
```

getData

Syntax string DataItem.getData( )

Returns the data value of this data element.

Returns String. The data value.

Example This example displays the data element's data value in an alert box:

```
alert(myDataItem.getData( ));
```

getHtmlDom

Syntax HTMLElement DataItem.getHtmlDom( )

Returns the HTML element for this data item.

Returns HTMLElement.

Example This example displays the HTML DOM element for this data item inside a red border:

```
function showHtmlDom(myDataItem){
   var domNode = myDataItem.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string DataItem.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myDataItem){
   var elementID = myDataItem.getInstanceId( );
   alert (elementID);
}
```

getPageContent

Syntax actuate.viewer.PageContent DataItem.getPageContent( )

Returns the page content to which this data item belongs.

Returns actuate.report.PageContent. report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myDataItem){
   var pageContent = myDataItem.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getType

Syntax string DataItem.getType( )

Returns the report element type of this object, which is data.

Returns String. "Data".

Example This example checks the report element type and displays an alert if the type is not "Data":

```
if (myDataItem.getType( ) != "Data"){
   alert("Type mismatch, report element type is not data");
}
```

hide

Syntax void DataItem.hide( )

Hides this element.

Example Use hide( ) to hide a data item object, as shown in the following code:

```
myDataItem.hide( );
```

show

Syntax void DataItem.show( )

Shows this element.

Example Use show( ) to reveal a hidden data item object, as shown in the following code:

```
myDataItem.show( );
```

submit

Syntax void DataItem.submit(function callback)

Submits all the asynchronous operations for this DataItem. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the

processing, it returns a response and the results are rendered on the page in the DataItem container.

Parameter **callback**
Function. The function to execute after the asynchronous call processing is done.

Example Use submit( ) to execute changes on a data item object, as shown in the following code:

```
myDataItem.submit( );
```

Class actuate.report.FlashObject

Description A container for a Flash object in a report. FlashObject provides functions to operate on a Flash object, such as retrieving content and getting the HTML DOM element from the report Flash element.

Constructor

The FlashObject object is constructed by actuate.viewer.PageContent.getFlashObjectByBookmark( ).

Function summary

Table 14-23 lists actuate.report.FlashObject functions.

Table 14-23 actuate.report.FlashObject functions

Function	Description
clearFilters( )	Removes filters from this FlashObject
getBookmark( )	Returns the bookmark name for this FlashObject
getHtmlDom( )	Returns the HTML element for this FlashObject
getInstanceId( )	Returns the report element instance id
getPageContent( )	Returns the page content to which this element belongs
getType( )	Returns the FlashObject's element type
hide( )	Hides this element
setFilters( )	Adds filters to this FlashObject
show( )	Shows this element
submit( )	Applies changes made to this FlashObject

clearFilters

Syntax void FlashObject.clearFilters(string columnName)

Clears the filters of a given column.

Parameter **columnName**
String. The name of the column from which to clear the filters.

Example This example clears existing filters from the PRODUCTLINE column:

```
function resetFilter(flashobj){
   flashobj.clearFilters("PRODUCTLINE");
   flashobj.submit( );
}
```

getBookmark

Syntax string FlashObject.getBookmark( )

Returns the bookmark of this FlashObject element.

Returns String.

Example This example displays the Flash object's bookmark in an alert box:

```
function alertBookmark(myFlashobj){
   alert(myFlashobj.getBookmark( ));
}
```

getHtmlDom

Syntax HTMLElement FlashObject.getHtmlDom( )

Returns the HTML element for this FlashObject.

Returns HTMLElement.

Example This example displays the HTML DOM element for this data item inside a red border:

```
function showHtmlDom(myFlashobj){
   var domNode = myFlashobj.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string FlashObject.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myFlashObject){
   var elementID = myFlashObject.getInstanceId( );
   alert (elementID);
}
```

getPageContent

Syntax actuate.viewer.PageContent FlashObject.getPageContent( )

Returns the page content to which this FlashObject belongs.

Returns actuate.viewer.PageContent. report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myFlashobj){
   var pageContent = myFlashobj.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getType

Syntax string FlashObject.getType( )

Returns the report element type of this object, which is FlashObject.

Returns String. "FlashObject".

Example This example checks the report element type and displays an alert if the type is not "FlashObject":

```
if (myFlashObject.getType( ) != "FlashObject"){
   alert("Type mismatch, report element type is not FlashObject");
}
```

hide

Syntax void FlashObject.hide( )

Hides this element.

Example Use hide( ) to hide the Flash object, as shown in the following code:

```
myFlashobj.hide( );
```

setFilters

Syntax void FlashObject.setFilters(actuate.data.Filter[ ] filters)

Sets the given filters.

Parameter **filters**
An array of actuate.data.Filter objects. The filter conditions to apply to this chart element.

Example This example applies a filter to the Flash object:

```
function newFilter(myFlashobj){
   var filter = new
      actuate.data.Filter("PRODUCTLINE", "=", "Trucks and Buses");
   var filters = new Array( );
   filters.push(filter);
   myFlashobj.setFilters(filters);
}
```

show

Syntax void FlashObject.show( )

Shows this element.

Example Use show( ) to reveal a hidden Flash object, as shown in the following code:

```
myFlashobj.show( );
```

submit

Syntax void FlashObject.submit(function callback)

Submits all the asynchronous operations for this FlashObject. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the FlashObject container.

Parameter **callback**
Function. The function to execute after the asynchronous call processing is done.

Example This example clears existing filters from the PRODUCTLINE column and pops up an alert box:

```
function alertResetFilter(flashobj){
   flashobj.clearFilters("PRODUCTLINE");
   flashobj.submit(alert("Filters Cleared"));
}
```

Class actuate.report.Gadget

Description A container for a Flash gadget object in a report. The Gadget class provides functions to operate on a Flash gadget object, such as retrieving content and getting the HTML DOM element from the report Flash element.

Constructor

The Gadget object is constructed by viewer.PageContent.getGadgetByBookmark( ).

Function summary

Table 14-24 lists actuate.report.Gadget functions.

Table 14-24 actuate.report.Gadget functions

Function	Description
clearFilters( )	Removes filters from this gadget
getBookmark( )	Returns the bookmark name for this gadget
getHtmlDom( )	Returns the HTML element for this gadget
getInstanceId( )	Returns the report element instance id
getPageContent( )	Returns the page content to which this element belongs
getType( )	Returns the gadget's element type, which is gadget
hide( )	Hides this element
setFilters( )	Adds filters to this gadget
setGadgetType( )	Sets the gadget type
setSize( )	Resizes the gadget's width and height
show( )	Shows this element
submit( )	Applies changes made to this gadget

clearFilters

Syntax void Gadget.clearFilters(string columnName)

Clears the filters of a given column.

Parameter **columnName**
String. The name of the column from which to clear the filters.

Example This example clears existing filters from the PRODUCTLINE column:

```
function resetFilter(myGadget){
   myGadget.clearFilters("PRODUCTLINE");
   myGadget.submit( );
}
```

getBookmark

Syntax string Gadget.getBookmark( )

Returns the bookmark of this Gadget element.

Returns String. The gadget's bookmark.

Example This example displays the gadget's bookmark in an alert box:

```
function alertBookmark(myGadget){
   alert(myGadget.getBookmark( ));
}
```

getHtmlDom

Syntax HTMLElement Gadget.getHtmlDom( )

Returns the HTML element for this gadget.

Returns HTMLElement.

Example This example displays the HTML DOM element for this gadget inside a red border:

```
function showHtmlDom(myGadget){
   var domNode = myGadget.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string Gadget.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myGadget){
   var elementID = myGadget.getInstanceId( );
   alert (elementID);
}
```

getPageContent

Syntax actuate.viewer.PageContent Gadget.getPageContent( )

Returns the page content to which this gadget belongs.

Returns actuate.viewer.PageContent. report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myGadget){
   var pageContent = myGadget.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getType

Syntax string Gadget.getType( )

Returns the report element type of this object, which is Gadget.

Returns String. "Gadget".

Example This example checks the report element type and displays an alert if the type is not "Gadget":

```
if (myGadget.getType( ) != "Gadget"){
   alert("Type mismatch, report element type is not Gadget");
}
```

hide

Syntax void Gadget.hide( )

Hides this element.

Example Use hide( ) to hide a gadget, as shown in the following code:

```
myGadget.show( );
```

setFilters

Syntax void Gadget.setFilters(actuate.data.Filter[ ] filters)

Sets the given filters.

Parameter **filters**
An array of actuate.data.Filter objects. The filter conditions to apply to this chart element.

Example This example applies a filter to the gadget:

```
function newFilter(myGadget){
   var filter = new
      actuate.data.Filter("PRODUCTLINE", "=", "Trucks and Buses");
   var filters = new Array( );
   filters.push(filter);
   myGadget.setFilters(filters);
}
```

setGadgetType

Syntax void Gadget.setGadgetType(string chartType)

Specifies the gadget type for the Gadget element. The chart type is a constant.

Parameter **chartType**
String. The possible values are constants as listed below:

- GADGET_TYPE_BULLET: Bullet gadget subtype
- GADGET_TYPE_CYLINDER: Cylinder gadget subtype
- GADGET_TYPE_LINEARGAUGE: LinearGauge gadget subtype
- GADGET_TYPE_METER: Meter gadget subtype
- GADGET_TYPE_SPARK: Spark gadget subtype
- GADGET_TYPE_THERMOMETER: Thermometer gadget subtype

Example To change the gadget type to a meter, use code similar to the following:

```
myGadget.setGadgetType(actuate.report.Gadget.GADGET_TYPE_METER);
```

setSize

Syntax void Gadget.setSize(integer width, integer height)

Specifies the width and height of a gadget in pixels.

Parameters **width**
Integer. The width in pixels.

height
Integer. The height in pixels.

Example To set the gadget to a 300-by-300-pixel square area, use code similar to the following:

```
myGadget.setSize(300, 300);
```

show

Syntax void Gadget.show( )

Shows this element.

Example Use show( ) to reveal a hidden gadget, as shown in the following code:

```
myGadget.show( );
```

submit

Syntax void Gadget.submit(function callback)

Submits all the asynchronous operations for this gadget. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the gadget container.

Parameter **callback**
Function. The function to execute after the asynchronous call processing is done.

Example This example clears existing filters from the PRODUCTLINE column and pops up an alert box:

```
function alertResetFilter(myGadget){
   myGadget.clearFilters("PRODUCTLINE");
   myGadget.submit(alert("Filters Cleared"));
}
```

Class actuate.report.HTML5Chart.ClientChart

Description A container for an HTML5-enabled chart element in a report. ClientChart provides functions to operate on a ClientChart element on the client side only, such as retrieving the chart size or setting the title and series values for the currently displayed chart.

Constructor

The ClientChart object is constructed by actuate.report.Chart.getClientChart( ).

Function summary

Table 14-25 lists actuate.report.HTML5Chart.ClientChart functions.

Table 14-25 actuate.report.HTML5Chart.ClientChart functions

Function	Description
addSeries( )	Adds a series to the chart
getCategoryCount( )	Returns the number of categories in the chart
getChartHeight( )	Returns the height of the chart in pixels
getChartWidth( )	Returns the width of the chart in pixels
getClientOptions( )	Returns the chart options
getCore( )	Returns the core Highcharts object
getSeriesCount( )	Returns the number of run-time series in the chart
getXAxisMax( )	Returns the maximum value of X-axis series
getXAxisMin( )	Returns the minimum value of X-axis series
getYAxisMax( )	Returns the maximum value of Y-axis series
getYAxisMin( )	Returns the minimum value of Y-axis series
isChartWithAxes( )	Returns whether chart has axes
redraw( )	Redraws the chart according to chart options
removeSeries( )	Removes specified series
setSeriesVisible( )	Hides or displays specified series
setTitle( )	Updates chart title
setValues( )	Updates values of specified series
setXAxisRange( )	Changes the minimum and maximum of the X-axis and zooms in on the new data range

Table 14-25 actuate.report.HTML5Chart.ClientChart functions

Function	Description
setYAxisRange( )	Changes the minimum and maximum of the Y-axis and zooms in on the new data range

addSeries

Syntax void ClientChart.addSeries(string seriesName, Array values)

Adds a data series to this ClientChart.

Parameters

seriesName
String. A name for the series.

values
Array. The values for the series, defining X and Y value pairs.

Example This example adds the monthly revenue series as an array of numbers:

```
myClientChart.addSeries('monthly revenue', [1,5.5, 2,4.5, 3,7.8,
   4,7.7, 5,1.2, 6,8.5 7,1.9, 8,4.5, 9,12, 10,9.1, 11,4, 12,6.6]);
```

getCategoryCount

Syntax integer ClientChart.getCategoryCount( )

Returns the number of categories in this ClientChart.

Returns Integer. The number of categories.

Example This example displays the number of categories in myClientChart as an alert:

```
alert("This HTML5 client chart has" +
   myClientChart.getCategoryCount( ) + "categories.");
```

getChartHeight

Syntax integer ClientChart.getChartHeight( )

Returns the height of this ClientChart in pixels.

Returns Integer. The height of the chart in pixels.

Example This example displays the height of myClientChart as an alert:

```
alert("Height: " + myClientChart.getHeight( ));
```

getChartWidth

Syntax integer ClientChart.getChartWidth( )

Returns the width of this ClientChart in pixels.

Returns Integer. The width of the chart in pixels.

Example This example displays the width of myClientChart as an alert:

```
alert("Width: " + myClientChart.getChartWidth( ));
```

getClientOptions

Syntax actuate.report.HTML5Chart.ClientOption ClientChart.getClientOptions( )

Returns the ClientOptions set for this ClientChart.

Returns actuate.report.HTML5Chart.ClientOption object. The client options.

Example This example retrieves the client options for myClientChart and stores them in the myClientOptions variable:

```
var myClientOptions = myClientChart.getClientOptions( );
```

getCore

Syntax actuate.report.HTML5Chart.Highcharts ClientChart.getCore( )

Returns the Highcharts object contained in this ClientChart.

Returns actuate.report.HTML5Chart.Highcharts object. A Highcharts object.

Example This example retrieves the Highcharts object from myClientChart and stores it in the myHighchart variable:

```
var myHighchart = myClientChart.getCore( );
```

getSeriesCount

Syntax integer ClientChart.getSeriesCount( )

Returns the number of run-time series in this ClientChart.

Returns Integer. The number of series.

Example This example displays the number of run-time series in myClientChart as an alert:

```
alert("Runtime Series: " + myClientChart.getSeriesCount( ));
```

getXAxisMax

Syntax float ClientChart.getXAxisMax( )

Returns the maximum value of the series associated with the X-axis in this ClientChart.

Returns Float. The axis series' maximum.

Example This example displays the maximum value of the series associated with the X-axis in myClientChart as an alert:

```
alert("Max for X-axis series: " + myClientChart.getXAxisMax( ));
```

getXAxisMin

Syntax float ClientChart.getXAxisMin( )

Returns the minimum value of the series associated with the X-axis in this ClientChart.

Returns Float. The axis series' minimum.

Example This example displays the minimum value of the series associated with the X-axis in myClientChart as an alert:

```
alert("Min for X-axis series: " + myClientChart.getXAxisMin( ));
```

getYAxisMax

Syntax float ClientChart.getYAxisMax(integer axisIndex)

Returns the maximum value of a series associated with the Y-axis in this ClientChart.

Parameter **axisIndex**
Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

Returns Float. The axis series' maximum.

Example This example displays the maximum value of the series associated with the Y-axis in myClientChart as an alert:

```
alert("Max for Y-axis series: " + myClientChart.getYAxisMax( ));
```

getYAxisMin

Syntax float ClientChart.getYAxisMin(integer axisIndex)

Returns the minimum value of a series associated with the Y-axis in this ClientChart.

Parameter **axisIndex**
Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

Returns Float. The axis series' minimum.

Example This example displays the minimum value of the series associated with the Y-axis in myClientChart as an alert:

```
alert("Min for Y-axis series: " + myClientChart.getYAxisMin( ));
```

isChartWithAxes

Syntax boolean ClientChart.isChartWithAxes( )

Returns whether this chart has axes.

Returns Boolean. True indicates axes, false otherwise.

Example This example displays whether myClientChart has axes:

```
alert("Chart has axes: " + myClientChart.isChartWithAxes( ));
```

redraw

Syntax void ClientChart.redraw(actuate.report.HTML5Chart.ClientOption chartOptions)

Redraws this ClientChart with options.

Parameter **chartOptions**
actuate.report.HTML5Chart.ClientOption object. Optional. The chart options.

Example This example redraws myClientChart with the default options:

```
myClientChart.redraw( );
```

removeSeries

Syntax void ClientChart.removeSeries(string seriesName, boolean redraw)

Removes a series by name.

Parameters **seriesName**
String. The name of the series to remove.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

Example This example removes the series monthly revenue from myClientChart and redraws the chart:

```
myClientChart.removeSeries('monthly revenue', true);
```

setSeriesVisible

Syntax void ClientChart.setSeriesVisible(string seriesName, boolean visible)

Makes a series visible.

Parameters **seriesName**
String. The name of the series to change.

visible
Boolean. Optional. True indicates visible. Default is true.

Example This example sets the series monthly revenue as visible for myClientChart:

```
myClientChart.setSeriesVisible('monthly revenue', true);
```

setTitle

Syntax void ClientChart.setTitle(string title)

Sets the title of this ClientChart.

Parameter **title**
String. Chart title text.

Example This example sets the title of myClientChart to 'Annual Report':

```
myClientChart.setTitle('Annual Report');
```

setValues

Syntax void ClientChart.setValues(string seriesName, float[ ] values, boolean redraw)

Sets the value for a series.

Parameters **seriesName**
String. Name of the series to change.

values
Array of float. The values for the series, defining X and Y value pairs.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

Example This example adds the monthly revenue series as an array of numbers:

```
myClientChart.setValues('monthly revenue', [1,5.5, 2,4.5, 3,7.8,
   4,7.7, 5,1.2, 6,8.5 7,1.9, 8,4.5, 9,12, 10,9.1, 11,4, 12,6.6]);
```

setXAxisRange

Syntax void ClientChart.setXAxisRange(float min, float max, boolean redraw)

Sets the value range for the X-axis.

Parameters **min**
Float. A new minimum value.

max
Float. A new maximum value.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

Example This example sets the X-axis range to 1 through 3 and redraws the chart:

```
myClientChart.setXAxisRange(1,3);
```

setYAxisRange

Syntax void ClientChart.setYAxisRange(float min, float max, boolean redraw, integer axisIndex)

Sets the value range for the Y-axis.

Parameters

min
Float. A new minimum value.

max
Float. A new maximum value.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

axisIndex
Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

Example This example sets the Y-axis range to 0 through 15 and redraws the chart:

```
myClientChart.setYAxisRange(0,15);
```

Class actuate.report.HTML5Chart.ClientOption

Description A container for a ClientOption element in a report. ClientOption provides functions to change ClientChart features, such as orientation, type, and title.

Constructor

Syntax void actuate.report.HTML5Chart.ClientOption( )

Generates a new ClientOption object to manage the chart options for a ClientChart.

Function summary

Table 14-26 lists actuate.report.HTML5Chart.ClientOption functions.

Table 14-26 actuate.report.HTML5Chart.ClientOption functions

Function	Description
addSeries( )	Adds a series to the chart
explodePieSlice( )	Explodes specified pie's slice
isChartWithAxes( )	Checks if current chart is chart with axes
pivotChart( )	Inverts chart
setChartType( )	Updates chart type
setSeriesVisible( )	Hides or shows specified series
setTitle( )	Updates chart title
setXAxisTitle( )	Updates X-axis title
setYAxisTitle( )	Updates Y-axis title

addSeries

Syntax void ClientOption.addSeries(string seriesName, float[ ] values)

Adds a data series to this ClientOption.

Parameters

seriesName
String. A name for the series.

values
Array of float. The values for the series, defining X and Y value pairs.

Example This example adds the monthly revenue series as an array of numbers:

```
myClientOption.addSeries('monthly revenue', [1,5.5, 2,4.5, 3,7.8,
   4,7.7, 5,1.2, 6,8.5 7,1.9, 8,4.5, 9,12, 10,9.1, 11,4, 12,6.6]);
```

explodePieSlice

Syntax void ClientOption.explodePieSlice(string categoryName, boolean sliced)

Explodes the specified pie chart's slice.

Parameters

categoryName
String. The name of a category.

sliced
Boolean. Optional. True means the chart is sliced. Default is true.

Example This example explodes the Q1 category from a chart with myClientOption:

```
myClientOption.explodePieSlice('Q1');
```

isChartWithAxes

Syntax boolean ClientChart.isChartWithAxes( )

Returns whether this chart has axes.

Returns Boolean.

Example This example displays whether myClientOption has axes:

```
alert("Options has axes: " + myClientOption.isChartWithAxes( ));
```

pivotChart

Syntax void ClientChart.pivotChart( )

Switches the axes of the chart, if the chart has axes.

Example This example switches the axes in myClientOption and then redraws myClientChart with the switched axes:

```
var myClientOption = myClientChart.getClientOption( )
myClientOption.pivotChart( );
myClientChart.redraw(myClientOption);
```

setChartType

Syntax void ClientOption.setChartType(string chartType, boolean isCurve)

Sets the chart type in this ClientOption.

Parameters

chartType
String. The chart type. Valid values are line, area, bar, scatter, and pie.

isCurve
Boolean. Optional. Indicates if line or area chart is curve. Default value is false.

Example This example changes the chart type to pie in myClientOption:

```
myClientOption.setChartType('pie');
```

setSeriesVisible

Syntax void ClientOption.setSeriesVisible(string seriesName, boolean visible)

Makes a series visible.

Parameters

seriesName
String. The name of the series to change.

visible
Boolean. Optional. Default is true.

Example This example sets the series months as visible for myClientOption:

```
myClientOption.setSeriesVisible('monthly revenue', true);
```

setTitle

Syntax void ClientOption.setTitle(string title)

Sets the title of this ClientOption.

Parameter

title
String. Chart title text.

Example This example sets the title of myClientOption to 'Annual Report':

```
myClientOption.setTitle('Annual Report');
```

setXAxisTitle

Syntax void ClientOption.setTitle(string title)

Sets the X-axis title of this ClientOption.

Parameter

title
String. X-axis title text.

Example This example sets the title of the X-axis in myClientOption to 'Month':

```
myClientOption.setXAxisTitle('Month');
```

setYAxisTitle

Syntax void ClientOption.setTitle(string title, integer ChartOptions)

Sets the Y-axis title of this ClientOption.

Parameters

title
String. Y-axis title text.

chartOptions
Integer. Optional. Axis index. The minimum value is 0, which is the default value, indicating the first Axis.

Example This example sets the title of the Y-axis in myClientOption to 'Dollars, in millions':

```
myClientOption.setYAxisTitle('Dollars, in millions');
```

Class actuate.report.HTML5Chart.ClientPoint

Description Represents a data point in a chart. ClientPoint provides functions to manage a point in a series on an individual basis, including selections, options, and events. The options for ClientPoint are defined in the Highcharts point class, which is documented at the following URL:

http://api.highcharts.com/highcharts

Constructor

Syntax void actuate.report.HTML5Chart.ClientPoint( )

Generates a new ClientPoint object to manage a data point for a ClientChart.

Function summary

Table 14-27 lists actuate.report.HTML5Chart.ClientPoint functions.

Table 14-27 actuate.report.HTML5Chart.ClientPoint functions

Function	Description
applyOptions( )	Changes the point values or options
destroy( )	Destroys a point to clear memory
remove( )	Removes a point
select( )	Toggles the selection of a point
remove( )	Updates the point with new options

applyOptions

Syntax void ClientPoint.applyOptions({float | object} options)

Applies the options containing the x and y data and possibly some extra properties. This is called on point initialization or from point.update.

Parameter **options**
Float, array of float, or object. The point options. If options is a single number, the point gets that number as the Y value. If options is an array, the point gets the first two numbers as an X and Y value pair. If options is an object, advanced options as outlined in the Highcharts options.point are applied. The fields include color, events, id, marker, legend, Index (pie chart only), name, sliced (pie chart only), x, and y.

Example This example changes the Y value of myClientPoint to 12:

```
myClientPoint.applyOptions(12);
```

destroy

Syntax void ClientPoint.destroy( )

Destroys a point to clear memory. Its reference still stays in series.data.

Example This example destroys the options and values for myClientPoint:

```
myClientPoint.destroy( );
```

remove

Syntax void ClientPoint.remove(boolean redraw, {boolean | object} animation)

Removes this point and optionally redraws the series and axes.

Parameters

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

animation
Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

Example This example removes myClientPoint from a series, and redraws the chart with animation to display the changed series:

```
myClientPoint.remove( );
```

select

Syntax void ClientPoint.select(boolean selected, boolean accumulate)

Selects this point.

Parameters

selected
Boolean. Specifies whether to select or deselect the point.

accumulate
Boolean. Whether to add this point to the previous selection. By default, this is true when the Ctrl (PC) or Cmd (Macintosh) key is held during selection.

Example This example selects MyClientPoint and deselects all other points:

```
myClientPoint.select(true, false);
```

remove

Syntax void ClientPoint.remove(boolean redraw, {boolean | object} animation)

Updates this point and optionally redraws the series and axes.

Parameters

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

animation
Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

Example This example removes myClientPoint and redraws the chart:

```
myClientPoint.remove();
```

update

Syntax void ClientPoint.update({float|float[ ]|object} options, boolean redraw, {boolean | object} animation)

Updates this point and optionally redraws the series and axes.

Parameters

options
Float, array of float, or object. The point options. If options is a single number, the point gets that number as the Y value. If options is an array, the point gets the first two numbers as an X and Y value pair. If options is an object, advanced options as outlined in the Highcharts options.point are applied. The fields include color, events, id, marker, legend, Index (pie chart only), name, sliced (pie chart only), x, and y.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

animation
Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

Example This example updates myClientPoint with an X value of 1 and a Y value of 12, then redraws the point:

```
myClientPoint.update([1,12]);
```

Class actuate.report.HTML5Chart.ClientSeries

Description A container for a ClientSeries in a ClientChart. ClientSeries provides functions to manage a series and the graph of that series. In the ClientSeries object, all the points are accessible from the ClientSeries.data array.

Constructor

Syntax void actuate.report.HTML5Chart.ClientSeries( )

Generates a new ClientSeries object to manage a series for a ClientChart.

Function summary

Table 14-28 lists actuate.report.HTML5Chart.ClientSeries functions.

Table 14-28 actuate.report.HTML5Chart.ClientSeries functions

Function	Description
addPoint( )	Adds a point to the series
cleanData( )	Sorts the data and removes duplicates
destroy( )	Clears DOM objects and frees up memory
hide( )	Hides the series graph
redraw( )	Redraws the series after an update in the axes
remove( )	Removes a series and optionally redraws the chart
render( )	Renders the series graph and markers
select( )	Sets the selected state of the series graph
setData( )	Replaces the series data with a new set of data
setVisible( )	Sets the visibility of the series graph
show( )	Shows the series graph

addPoint

Syntax void ClientSeries.addPoint({float | object} options, boolean redraw, boolean shift, {boolean | object} animation)

Adds a point dynamically to the series.

Parameters **options**
Object. The point options. If options is a single number, the point gets that number as the Y value. If options is an array, the point gets the first two numbers as an X and Y value pair. If options is an object, advanced options as outlined in

the Highcharts options.point are applied. The fields include color, events, id, marker, legend, Index (pie chart only), name, sliced (pie chart only), x, and y.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

shift
Boolean. When shift is true, the graph of the series shifts one point toward the end of the series and a point added to the beginning of the series. Default is false.

animation
Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

Example This example adds a point of value 12 to the end of myClientSeries:

```
myClientSeriesaddPoint(12);
```

cleanData

Syntax void ClientSeries.cleanData( )

Sorts the series and removes duplicate points or values.

Example This example sorts myClientSeries and removes its duplicate points and values:

```
myClientSeries.cleanData( );
```

destroy

Syntax void ClientSeries.destroy( )

Clears DOM series objects and frees memory.

Example This example clears the memory of myClientSeries and its member objects:

```
myClientSeries.destroy( );
```

hide

Syntax void ClientSeries.hide( )

Hides the graph of this series.

Example This example hides myClientSeries graph from the chart:

```
myClientSeries.hide( );
```

redraw

Syntax void ClientSeries.redraw( )

Redraws the graph of this series after updating the data and axes.

Example This example redraws the graph of myClientSeries:

```
myClientSeries.redraw( );
```

remove

Syntax void ClientSeries.remove(boolean redraw, {boolean | object} animation)

Removes this series and optionally redraws the chart.

Parameters

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

animation
Boolean or object. Optional. Whether to apply animation, and optionally animation configuration. Default is true.

Example This example removes the graph of myClientSeries from the chart:

```
myClientSeries.remove( );
```

render

Syntax void ClientSeries.render( )

Renders the graph of this series and its markers.

Example This example renders the graph of myClientSeries to the chart:

```
myClientSeries.render( );
```

select

Syntax void ClientSeries.select(boolean selected)

Selects this series.

Parameter

selected
Boolean. Optional. Specifies whether to select or deselect the series. If undefined, toggles selection.

Example This example selects myClientSeries:

```
myClientSeries.select(true);
```

setData

Syntax void ClientSeries.setData({float | object}[ ] data, boolean redraw)

Replaces the series data with a new set of data.

Parameters

data
Array of float and/or object. An array of data points for the series. The points can be given in three ways:

1. A list of numerical values, which are assigned as Y values, paired with X values starting with 0 and incrementing by 1 for each additional number. For example:

```
[0, 5, 3, 5]
```

2. A list of arrays with two values, which are assigned as X and Y value pairs. If the first value is a string, it is applied as the name of the point, and the x value is incremented following the above rules. For example:

```
[[4, 2], [6, 3], [8, 2]]
```

3. A list of objects with named values, which are assigned to points using the Highcharts point configuration specification options.point. For example:

```
[{ name: 'Point 1',
   color: '#00FF00',
   y: 0
},
{  name: 'Point 2',
   color: '#FF00FF',
   y: 5
}]
```

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

Example

This example replaces the points in myClientSeries with three new points:

```
myClientSeries.setData([[4, 2], [6, 3], [8, 2]]);
```

setVisible

Syntax

void ClientSeries.setVisible(boolean vis, boolean redraw)

Sets the visibility of this series.

Parameters

vis
Boolean. Optional. Specifies whether to display the series. True displays the series, false hides it. If no value is provided, the visibility changes to false if visibility is true, and true if visibility is false.

redraw
Boolean. Optional. Specifies whether to redraw the chart. Default is true.

Example

This example sets myClientSeries to visible and redraws it:

```
myClientSeries.setVisible(true);
```

show

Syntax void ClientSeries.show( )

Displays the graph of this series.

Example This example displays the graph of myClientSeries:

```
myClientSeries.show( );
```

Class actuate.report.HTML5Chart.Highcharts

Description A container for a Highcharts element in a ClientChart. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Constructor

Syntax void actuate.report.HTML5Chart.Highcharts( )

Generates a new Highcharts object to manage the Highcharts for a ClientChart.

Class actuate.report.HTML5Chart.Renderer

Description A container for a Highcharts renderer object. Directly accesses the Highcharts rendering layer to draw primitive shapes like circles, rectangles, paths or text directly. The renderer represents a wrapper object for SVG in modern browsers and VML in older versions of Microsoft Internet Explorer.

Constructor

Syntax void actuate.report.HTML5Chart.Renderer( )

Generates a new Renderer object to manage the Highcharts rendering options for a ClientChart.

Function summary

Table 14-29 lists actuate.report.HTML5Chart.Renderer functions.

Table 14-29 actuate.report.HTML5Chart.Renderer functions

Function	Description
arc( )	Draws and returns an arc
circle( )	Draws a Scalable Vector Graphic circle
clipRect( )	Defines a clipping rectangle
destroy( )	Destroys the renderer and its allocated members
g( )	Creates a group
image( )	Displays an image
path( )	Draws a path
rect( )	Draws and returns a rectangle
setSize( )	Resizes the box and re-aligns all aligned elements
text( )	Adds text to the Scalable Vector Graphic object

arc

Syntax object Renderer.arc(integer x, integer y, integer r, integer innerR, float start, float end)

Generates and draws an arc on the chart.

Parameters **x**
Integer. The X position of the arc's center, measured in pixels from the left edge of the rendering area.

y
Integer. The Y position of the arc's center, measured in pixels from the top edge of the rendering area.

r
Integer. The outer radius, measured in pixels.

innerR
Integer. The inner radius, measure in pixels.

start
Float. The starting angle of the arc, measured in radians, where 0 is directly right and -Math.PI/2 is directly upward. The arc is drawn clockwise from start to end.

end
Float. The ending angle of the arc, measured in radians, where 0 is directly right and -Math.PI/2 is directly upward.

Returns Highcharts element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example draws a 50-pixel wide half-circle arc, concave down, with a center 200 pixels from the left edge and 150 pixels from the top edge of the chart area:

```
myRenderer.arc(200, 150, 100, 50, -Math.PI, 0);
```

circle

Syntax object Renderer.circle(integer x, integer y, integer r)

Generates and draws a Scalable Vector Graphic circle on the chart.

Parameters

x
Integer. The X position of the circle's center, measured in pixels from the left edge of the rendering area.

y
Integer. The Y position of the circle's center, measured in pixels from the top edge of the rendering area.

r
Integer. The radius, measured in pixels.

Returns Highcharts element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example draws a circle with a center 200 pixels from the left edge and 150 pixels from the top edge of the chart area:

```
myRenderer.circle(200, 150, 100);
```

clipRect

Syntax object Renderer.clipRect(string id, integer x, integer y, integer width, integer height)

Generates and draws a clipping rectangle on the chart.

Parameters

id
String. A string to identify the element.

x
Integer. The X position of the rectangle's upper left corner, measured in pixels from the left edge of the rendering area.

y
Integer. The Y position of the rectangle's upper left corner, measured in pixels from the top edge of the rendering area.

width
Integer. The width, in pixels.

height
Integer. The height, in pixels.

Returns Highcharts element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example draws a 100-pixel-by-100-pixel rectangle 100 pixels from the left and top edges of chart area:

```
myRenderer.cliprect('myClipRect', 100, 100, 100, 100);
```

destroy

Syntax void Renderer.destroy( )

Destroys this renderer and its allocated elements.

Example This example destroys the myRenderer object and frees its memory:

```
myRenderer.destroy( );
```

g

Syntax object Renderer.g(string name)

Adds an SVG/VML group to the Renderer object.

Parameter

name
String. The name of the group. Used in the class name, which will be "highcharts-"+ name. Other Element objects are added to the group by using this group as the first parameter in .add( ) for the element wrappers.

Returns Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example creates a new group called myGroup:

```
myRenderer.g('myGroup');
```

image

Syntax object Renderer.image(string src, integer x, integer y, integer width, integer height)

Generates and draws a image on the chart.

Parameters

src
String. A URL for the image.

x
Integer. The X position of the image's upper left corner, measured in pixels from the left edge of the rendering area.

y
Integer. The Y position of the image's upper left corner, measured in pixels from the top edge of the rendering area.

width
Integer. The width, in pixels.

height
integer. The height, in pixels.

Returns Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example adds the sun.png image to the chart 100 pixels from the left and top of the edge of the chart:

```
myRenderer.image('http://highcharts.com/demo/gfx/sun.png', 100,
   100, 30, 30);
```

path

Syntax object Renderer.path(object[ ] path)

Adds a path to the renderer based on SVG's path commands. In SVG-capable browsers, all path commands are supported, but in VML only a subset is supported, including the moveTo, lineTo, and curve commands.

Parameter

path
Array of string and integer objects. An SVG path with attributes split up in array form.

Returns Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example draws a path from the upper left corner of the rendering area (0, 0) to the points (100, 100), (200, 50), and (300, 100), where the first number represents the distance from the left edge of the rendering area and the second number represents the distance from the top edge of the rendering area:

```
myRenderer.path(['M', 0, 0, 'L', 100, 100, 200, 50, 300, 100]);
```

rect

Syntax object Renderer.rect(integer x, integer y, integer width, integer height, integer r, integer strokeWidth)

Generates and draws a rectangle on the chart.

Parameters

x
Integer. The X position of the rectangle's upper left corner, measured in pixels from the left edge of the rendering area.

y
Integer. The Y position of the rectangle's upper left corner, measured in pixels from the top edge of the rendering area.

width
Integer. The width, in pixels.

height
Integer. The height, in pixels.

r
Integer. The corner radius, measured in pixels.

strokeWidth
Integer. Stroke measurement to support crisp drawing.

Returns Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example draws a 100-pixel-by-100-pixel rectangle 100 pixels from the left and top edges of chart area with 5-pixel-radius quarter-circles as edges:

```
myRenderer.rect(100, 100, 100, 100, 5);
```

setSize

Syntax void Renderer.setSize(integer width, integer height, boolean animate)

Resizes the rendering area and re-aligns all aligned elements.

Parameters

width
Integer. The width, in pixels.

height
Integer. The height, in pixels.

animate
Boolean. Optional. Whether to animated the resize. Default is true.

Example This example resizes the renderer area to 500 pixels by 500 pixels:

```
myRenderer.setSize(500, 500);
```

text

Syntax object Renderer.text(string str, integer x, integer y, boolean useHTML)

Adds text to the Scalable Vector Graphic object.

Parameters

str
String. The text in this text element.

x
Integer. The X position of the text's lower left corner, measured in pixels from the left edge of the rendering area.

y
Integer. The Y position of the text's lower left corner, measured in pixels from the top edge of the rendering area.

useHTML
Boolean. Specifies whether to use HTML to render the text.

Returns Highcharts element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highcharts. For reference material for Highcharts, consult the BIRT Designer Professional help or access the Highcharts documentation at the following URL:

http://api.highcharts.com/highcharts

Example This example adds a text graphic that reads “Series 1” 140 pixels from the left edge of the rendering area and 150 pixels from the top edge of the rendering area:

```
myRenderer.text('Series 1', 140, 150, false);
```

Class actuate.report.Label

Description A container for a Label element in a report. Label provides functions to operate on a Label element, such as retrieving the label text and getting the HTML DOM element from the report label.

Constructor

The Label object is constructed by viewer.PageContent.getLabelByBookmark( ).

Function summary

Table 14-30 lists actuate.report.Label functions.

Table 14-30 actuate.report.Label functions

Function	Description
getBookmark( )	Returns the bookmark name for this Label
getHtmlDom( )	Returns the HTML element for this Label
getInstanceId( )	Returns the report element instance id
getLabel( )	Returns the text of this Label element
getPageContent( )	Returns the page content to which this element belongs
getType( )	Returns the Label's element type
hide( )	Hides this element
show( )	Shows this element
submit( )	Applies changes made to this gadget

getBookmark

Syntax string Label.getBookmark( )

Returns the bookmark name for this Label.

Returns String. The Label's bookmark.

Example This example displays the Label's bookmark in an alert box:

```
alert(myLabel.getBookmark( ));
```

getHtmlDom

Syntax HTMLElement Label.getHtmlDom( )

Returns the HTML element for this Label.

Returns HTMLElement.

Example This example displays the HTML DOM element for this Label inside a red border:

```
function showHtmlDom(myLabel){
   var domNode = myLabel.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string Label.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myLabel){
   var elementID = myLabel.getInstanceId( );
   alert (elementID);
}
```

getLabel

Syntax string Label.getLabel( )

Returns the text of this Label element.

Returns String. The Label text.

Example This example displays the text of the myLabel object in an alert box:

```
alert("Label element text is " + myLabel.getLabel( ));
```

getPageContent

Syntax actuate.viewer.PageContent Label.getPageContent( )

Returns the page content to which this Label belongs.

Returns actuate.viewer.PageContent. report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myLabel){
   var pageContent = myLabel.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getType

Syntax string Label.getType( )

Returns the report element type of this object, which is Label.

Returns String. "Label".

Example This example checks the report element type and displays an alert if the type is not "Label":

```
if (myElement.getType( ) != "Label"){
   alert("Type mismatch, report element type is not Label")
}
```

hide

Syntax void Label.hide( )

Hides this element.

Example Use hide( ) to hide a report label, as shown in the following code:

```
myLabel.hide( );
```

show

Syntax void Label.show( )

Shows this element.

Example Use show( ) to reveal a report label, as shown in the following code:

```
myLabel.show( );
```

submit

Syntax void Label.submit(function callback)

Submits all the asynchronous operations for this Label. Submit( ) triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the label container.

Parameter **callback**

Function. The function to execute after the asynchronous call processing is done.

Example Use submit( ) to execute changes on a Label object, as shown in the following code:

```
myLabel.submit( );
```

Class actuate.report.Table

Description A container for a Table element in a report. Table provides functions to operate on a Table element, such as manipulating columns, groups, and data.

Constructor

The Table object is constructed by viewer.PageContent.getTableByBookmark( ).

Function summary

Table 14-31 lists actuate.report.Table functions.

Table 14-31 actuate.report.Table functions

Function	Description
clearFilters( )	Clears the filters from the given column
getBookmark( )	Returns the bookmark name for this Table
getColumn( )	Gets the Table data by column index and returns only the data from the current visible page
getHtmlDom( )	Returns the HTML element for this Table
getInstanceId( )	Returns the report element instance id
getPageContent( )	Returns the page content to which this element belongs
getRow( )	Gets the Table data by row index
getType( )	Returns the report element type
groupBy( )	Adds an inner group to this Table
hide( )	Hides this element
hideColumn( )	Hides a Table column by specifying the column name
hideDetail( )	Hides detailed information for displayed groups
removeGroup( )	Removes an inner group
setFilters( )	Applies filters to this Table
setSorters( )	Adds sorters to this Table
show( )	Shows this element
showColumn( )	Shows a Table column by specifying the column name
showDetail( )	Shows detailed information for displayed groups

Table 14-31 actuate.report.Table functions

Function	Description
submit( )	Submits all the asynchronous operations that the user has requested on this report and renders the Table component on the page
swapColumns( )	Swaps two columns, reordering the columns

clearFilters

Syntax void Table.clearFilters(string columnName)

Clears the filters of a given column.

Parameter **columnName**
String. The name of the column.

Example This example clears existing filters from the PRODUCTLINE column:

```
function resetFilter(myTable){
   myTable.clearFilters("PRODUCTLINE");
   myTable.submit( );
}
```

getBookmark

Syntax string Table.getBookmark( )

Returns the Table's name.

Returns String. The name of the Table.

Example This example displays the Table's bookmark in an alert box:

```
function alertBookmark(myTable){
   alert(myTable.getBookmark( ));
}
```

getColumn

Syntax array[ ] Table.getColumn(integer columnIndex)

Gets the Table data by column index. Returns the data from the current visible page.

Parameter **columnIndex**
Integer. Optional. The numerical index of the column from which to retrieve data. The getColumn( ) function returns the values for the first column when no value is provided for columnIndex.

Returns Array. A list of data in the format of the column.

Example This example returns the first column in myTable:

```
function getMyColumn(myTable) {
   return myTable.getColumn( );
}
```

getHtmlDom

Syntax HTMLElement Table.getHtmlDom( )

Returns the Table's name.

Returns String. The name of the Table.

Example This example displays the HTML DOM element for this Table inside a red border:

```
function showHtmlDom(myTable){
   var domNode = myTable.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string Table.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myTable){
   var elementID = myTable.getInstanceId( );
   alert (elementID);
}
```

getPageContent

Syntax actuate.viewer.PageContent Table.getPageContent( )

Returns the page content to which this Table belongs.

Returns actuate.viewer.PageContent. report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myTable){
   var pageContent = myTable.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getRow

Syntax array[ ] Table.getRow(integer rowIndex)

Gets the Table data by row index. Returns the data from the current visible page.

Parameter **rowIndex**
Integer. Optional. The numerical index of the row from which to retrieve data. The getRow( ) function returns the values for the first row when no value for rowIndex is provided.

Returns Array. A list of data in the format of the columns that cross the row.

Example This example retrieves the first row in myTable:

```
function getMyRow(myTable) {
   return myTable.getRow( );
}
```

getType

Syntax string Table.getType( )

Returns the report element type of this object, which is Table.

Returns String. "Table".

Example This example returns the report element type of this object in an alert box:

```
function getTableType(myTable) {
   alert("Element type is: " + myTable.getType( ));
}
```

groupBy

Syntax void Table.groupBy(string columnName)

Groups the data in a table by the values in a given column. If there is an existing group, this operation will add the new group after the existing group.

Parameter **columnName**
String. The name of the column to use for the innermost group to the Table.

Example This example groups the data in myTable by the values in the TOTAL column:

```
function groupByColumn(myTable) {
   myTable.groupBy("TOTAL");
}
```

hide

Syntax void Table.hide( )

Hides this element.

Example This example hides myTable:

```
myTable.hide( );
```

hideColumn

Syntax void Table.hideColumn(string columnName)

Hides a table column by specifying the column name.

Parameter **columnName**
String. The data binding name for the column to hide.

Example This example hides the TOTAL column from myTable:

```
function myHiddenColumn(myTable) {
   myTable.hideColumn("TOTAL");
   myTable.submit( );
}
```

hideDetail

Syntax void Table.hideDetail(string columnName)

Hides information for a column from the grouped data displayed on the page. If every column is hidden, only the group name is visible.

Parameter **columnName**
String. The data binding name for the column to hide.

Example This example hides the TOTAL column from the grouped data visible for myTable:

```
function hideMyDetail(myTable) {
   myTable.hideDetail("TOTAL");
   myTable.submit( );
}
```

removeGroup

Syntax void Table.removeGroup()

Removes the innermost group.

Example This example removes the innermost group from myTable and displays an alert box after calling submit( ):

```
function removeMyGroup(myTable) {
   myTable.removeGroup();
   myTable.submit(alert("Group removed"));
}
```

setFilters

Syntax void Table.setFilters(actuate.data.Filter filter)

void Table.setFilters(actuate.data.Filter[ ] filters)

Applies a filter or filters to this Table element.

Parameters

filter
actuate.data.Filter object. A single filter condition to apply to this Table.

filters
An array of actuate.data.Filter objects. Filter conditions to apply to this Table.

Example To add a filter to the Table to display only entries with a CITY value of NYC, use the following code:

```
var filters = new Array( );
var city_filter = new actuate.data.Filter("CITY",
   actuate.data.Filter.EQ, "NYC");
filters.push(city_filter);
table.setFilters(filters);
```

setSorters

Syntax void Table.setSorters(actuate.data.Sorter sorter)

void Table.setSorters(actuate.data.Sorter[ ] sorters)

Applies a sorter or sorters to this Table.

Parameters

sorter
actuate.data.Sorter object. A single sort condition to apply to this Table.

sorters
An array of actuate.data.Sorter objects. Sort conditions to apply to this Table.

Example This example adds the myStateSorter and myCitySorter sorters to myTable:

```
function setAllMySorters(myTable) {
   myTable.setSorters(["myStateSorter", "myCitySorter"]);
}
```

show

Syntax void Table.show( )

Shows this element.

Example Use show( ) to reveal a report Table, as shown in the following code:

```
myTable.show( );
```

showColumn

Syntax void Table.showColumn(string columnName)

Shows the Table column by specifying the column name.

Parameter **enabled**
String. The data binding name for the column to display.

Example This example shows the PRODUCTLINE column in myTable:

```
function showMyColumn(myTable) {
   myTable.showColumn("PRODUCTLINE");
   myTable.submit( );
}
```

showDetail

Syntax void Table.showDetail(string columnName)

Displays information for a column from the grouped data displayed on the page. If every column is hidden, only the group name is visible.

Parameter **columnName**
String. The data binding name for the column to display.

Example This example shows the information from the PRODUCTLINE column in the grouped data that is displayed for myTable:

```
function showMyDetail(myTable) {
   myTable.showDetail("PRODUCTLINE");
   myTable.submit( );
}
```

submit

Syntax void Table.submit(function callback)

Submits all the asynchronous operations for this Table element. The submit( ) function triggers an AJAX request to submit all the asynchronous operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the table container.

Parameter

callback
Function. The function called after the asynchronous call processing finishes.

Example This example clears existing filters from the PRODUCTLINE column and pops up an alert box:

```
function alertResetFilter(myTable){
   myTable.clearFilters("PRODUCTLINE");
   myTable.submit(alert("Filters Cleared"));
}
```

swapColumns

Syntax void Table.swapColumns(string columnName1, string columnName2)

Swaps the columns to reorder to column sequence of the Table.

Parameters

columnName1
String. The first column to swap in the column order.

columnName2
String. The second column to swap in the column order.

Example This example swaps the TOTAL and PRODUCTLINE columns in myTable:

```
function swapMyColumns(myTable){
   myTable.swapColumns("TOTAL", "PRODUCTLINE");
   myTable.submit( );
}
```

Class actuate.report.TextItem

Description A container for a Text element in a report. TextItem provides functions to operate on a Text element, such as retrieving the text value and getting the HTML DOM element from the report Text element.

Constructor

The TextItem object is constructed by viewer.PageContent.getTextByBookmark( ).

Function summary

Table 14-32 lists actuate.report.TextItem functions.

Table 14-32 actuate.report.TextItem functions

Function	Description
getBookmark( )	Returns the bookmark name for this Text
getHtmlDom( )	Returns the HTML element for this Text
getInstanceId( )	Returns the report element instance id
getPageContent( )	Returns the page content to which this element belongs
getText( )	Returns the text in this Text element
getType( )	Returns the Text element's type
hide( )	Hides this element
show( )	Shows this element
submit( )	Applies changes made to this element

getBookmark

Syntax string TextItem.getBookmark( )

Returns the bookmark name for this Text item.

Returns String.

Example This example displays the table's bookmark in an alert box:

```
function alertBookmark(myTextItem){
   alert(myTextItem.getBookmark( ));
}
```

getHtmlDom

Syntax HTMLElement TextItem.getHtmlDom( )

Returns the HTML element for this Text.

Returns HTMLElement.

Example This example displays the HTML DOM element for this Text item inside a red border:

```
function showHtmlDom(myTextItem){
   var domNode = myTextItem.getHtmlDom( );
   var box = document.createElement('div');
   box.style.border = '2px solid red';
   var label = document.createElement('h2');
   label.innerHTML = 'The HTML DOM:';
   box.appendChild(label);
   box.appendChild(domNode);
   document.body.appendChild(box);
}
```

getInstanceId

Syntax string TextItem.getInstanceId( )

Returns the instance id of this report element.

Returns String. The instance id.

Example This example displays the instance ID of the report element in an alert box:

```
function showID(myTextItem){
   var elementID = myTextItem.getInstanceId( );
   alert (elementID);
}
```

getPageContent

Syntax actuate.viewer.PageContent TextItem.getPageContent( )

Returns the page content to which this Text belongs.

Returns actuate.viewer.PageContent. report content.

Example This example displays the viewer ID of the page content in an alert box:

```
function showViewID(myTextItem){
   var pageContent = myTextItem.getPageContent( );
   var pageViewerID = pageContent.getViewerId( );
   alert (pageViewerID);
}
```

getText

Syntax string TextItem.getText( )

Returns the text of this Text element.

Returns String. The content text.

Example This example displays the text of the myTextItem object in an alert box:

```
alert("Text content for myTextItem is " + myTextItem.getText( ));
```

getType

Syntax string TextItem.getType( )

Returns the report element type of this object, which is Text.

Returns String. "Text".

Example This example checks the report element type and displays an alert if the type is not "Text":

```
if (myTextItem.getType( ) != "Text"){
   alert("Type mismatch, report element type is not Text");
}
```

hide

Syntax void TextItem.hide( )

Hides this element.

Example This example hides myTextItem:

```
myTextItem.hide( );
myTextItem.submit( );
```

show

Syntax void TextItem.show( )

Shows this element.

Example This example shows myTextItem:

```
myTextItem.show( );
myTextItem.submit( );
```

submit

Syntax void TextItem.submit(function callback)

Submits all the asynchronous operations for this TextItem. The submit( ) function triggers an AJAX request for all asynchronous operations. The server returns a response after processing. The results render on the page in the TextItem container.

Parameter **callback**
Function. The function to execute after the asynchronous call processing is done.

Example This example uses submit( ) after calling show( ) to show myTextItem:

```
myTextItem.show( );
myTextItem.submit( );
```

Class actuate.ReportExplorer

Description The actuate.ReportExplorer class retrieves and displays a navigable repository or file system interface that enables users to navigate folders and select files. This generic user interface enables the user to browse and select repository contents.

Constructor

Syntax actuate.ReportExplorer(string container)

Constructs a ReportExplorer object, initializing the ReportExplorer component.

Parameter **container**
String. The name of the HTML element that displays the rendered ReportExplorer component or a container object. The constructor initializes the ReportExplorer component but does not render it.

Function summary

Table 14-33 lists actuate.ReportExplorer functions.

Table 14-33 actuate.ReportExplorer functions

Function	Description
getFolderName( )	Gets the root folder name
getLatestVersionOnly( )	Gets the latestVersionOnly flag
getResultDef( )	Gets the resultDef value for this GetFolderItems
getSearch( )	Gets the search value for this GetFolderItems
onUnload( )	Unloads unused JavaScript variables
registerEventHandler( )	Registers the event handler
removeEventHandler( )	Removes the event handler
setContainer( )	Sets the div container
setFolderName( )	Sets the root folder name
setLatestVersionOnly( )	Sets the latestVersionOnly flag
setResultDef( )	Sets the resultDef value for this GetFolderItems
setSearch( )	Sets the search value for this GetFolderItems
setService( )	Sets the JSAPI web service
setStartingFolder( )	Sets the path for the initial folder selection
setUseDescriptionAsLabel( )	Sets flag to use descriptions as file/folder labels

Table 14-33 actuate.ReportExplorer functions

Function	Description
showFoldersOnly( )	Sets the flag to only display folders
submit( )	Applies changes made to this element

getFolderName

Syntax string ReportExplorer.getFolderName( )

Returns the name of the root folder for this ReportExplorer.

Returns String. The folder name.

Example This example displays the root folder's name in an alert box:

```
function alertRootFolder(myReportExplorer){
   alert(myReportExplorer.getFolderName( ));
}
```

getLatestVersionOnly

Syntax boolean ReportExplorer.getLatestVersionOnly( )

Returns the latest version only flag for this ReportExplorer.

Returns Boolean. True indicates that ReportExplorer displays only the latest version of each report.

Example This example displays the latest version only flag in an alert box:

```
function alertLatestVersionFlag(myReportExplorer){
   alert(myReportExplorer.getLatestVersionOnly( ));
}
```

getResultDef

Syntax string[ ] ReportExplorer.getResultDef( )

Returns the results definition.

Returns Array of strings. Valid values are: "Name", "FileType", "Version", "VersionName", "Description", "Timestamp", "Size", and "PageCount".

Example This example displays the results definition an alert box:

```
function alertResultsDefinition(myReportExplorer){
   alert(myReportExplorer.getResultDef( ));
}
```

getSearch

Syntax actuate.ReportExplorer.FileSearch ReportExplorer.getSearch( )

Returns the FileSearch object assigned to this ReportExplorer.

Returns actuate.reportexplorer.FileSearch object. The file search settings.

Example This example sets the FileSearch setting for reportexplorer1 to the FileSearch settings of reportexplorer2:

```
reportexplorer1.setSearch(reportexplorer2.getSearch( ));
```

onUnload

Syntax void ReportExplorer.onUnload( )

Unloads JavaScript variables that are no longer needed by ReportExplorer.

Example This example cleans up unused JavaScript variables for myReportExplorer:

```
myReportExplorer.onUnload( );
```

registerEventHandler

Syntax void ReportExplorer.registerEventHandler(string eventName, function handler)

Registers an event handler to activate for parameter eventName. This function can assign several handlers to a single event.

Parameters

eventName
String. Event name to capture.

handler
Function. The function to execute when the event occurs. The handler must take two arguments: the ReportExplorer instance that fired the event and an event object specific to the event type.

Example This example registers the errorHandler( ) function to respond to the ON_EXCEPTION event:

```
myReportExplorer.registerEventHandler(actuate.ReportExplorer.
                     EventConstants.ON_EXCEPTION, errorHandler);
```

removeEventHandler

Syntax void ReportExplorer.removeEventHandler(string eventName, function handler)

Removes an event handler to activate for parameter eventName.

Parameters

eventName
String. Event name to remove from the internal list of registered events.

handler
Function. The function to disable.

Example This example removes the errorHandler( ) function from responding to the ON_EXCEPTION event:

```
myReportExplorer.removeEventHandler(actuate.ReportExplorer.
                     EventConstants.ON_EXCEPTION, errorHandler);
```

setContainer

Syntax void ReportExplorer.setContainer(string containerId)

Sets the HTML element container for the ReportExplorer content.

Parameter **containerID**
String. The name of the HTML element that displays the group of rendered ReportExplorer components.

Example This example sets MyReportExplorer to render the element labeled "History":

```
myReportExplorer.setContainer("History");
```

setFolderName

Syntax void ReportExplorer.setFolderName(string folderName)

Sets the name of the root folder for this ReportExplorer.

Parameter **folderName**
String. The name of the repository folder to use as the root folder. Use a repository path to use subfolders for the root folder. The string '~/' maps to the current user's home folder.

Example This example sets the report explorer root folder to /Public:

```
myReportExplorer.setFolderName("/Public");
```

setLatestVersionOnly

Syntax void ReportExplorer.setLatestVersionOnly(boolean latestVersionOnly)

Sets the latest version only flag for this ReportExplorer.

Parameter **latestVersionOnly**
Boolean. True removes all but the latest versions from the report explorer.

Example This example sets ReportExplorer to display only the latest versions of all files:

```
myReportExplorer.setLatestVersionOnly( true );
```

setResultDef

Syntax void ReportExplorer.setResultDef(string[ ] resultDef)

Sets the results definition.

Parameter **resultDef**
Array of strings. Valid values are: "Name", "FileType", "Version", "VersionName", "Description", "Timestamp", "Size", and "PageCount". iHub requires the Name, FileType, and Version fields in the results definition array to identify all files.

Example This example sets the result set to five columns of data including name, file type, version, version name, and description:

```
var resultDef = "Name|FileType|Version|VersionName|Description";
myReportExplorer.setResultDef( resultDef.split("|") );
```

setSearch

Syntax void ReportExplorer.setSearch(actuate.ReportExplorer.FileSearch search)

Assigns a FileSearch object to this ReportExplorer.

Parameter **search**
actuate.reportexplorer.FileSearch object. The file search settings.

Example This example sets the FileSearch setting for reportexplorer1 to the FileSearch settings of reportexplorer2:

```
reportexplorer1.setSearch(reportexplorer2.getSearch( ));
```

setService

Syntax void ReportExplorer.setService(string iportalURL, actuate.RequestOptions requestOptions)

Sets the target service URL to which this explorer links. When the service URL is not set, this viewer links to the default service URL which is set on the actuate object.

Parameters **iPortalURL**
String. The target Actuate web application URL, either a Java Component or iHub Visualization Platform client.

requestOptions
actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. The URL can also include custom parameters.

Example This example sets the URL for the BIRT Java Component web application service:

```
myExplorer.setService("http://127.0.0.1:8080/ajc",
                          myRequestOptions);
```

setStartingFolder

Syntax void ReportExplorer.setStartingFolder(string strfoldername)

Sets the fully qualified path of the initially selected folder in the explorer tree.

Parameter **strfoldername**
String. The fully qualified path of a folder.

Example This example sets the initially selected folder to Public in the local repository:

```
myExplorer.setStartingFolder("C:\Actuate\JavaComponents\WEB-INF
   \repository\Public");
```

setUseDescriptionAsLabel

Syntax void ReportExplorer.setUseDescriptionAsLabel(boolean useDescription)

Sets the explorer to display the folder description as the folder label instead of the folder name.

Parameter **useDescription**
Boolean. True displays descriptions for folders instead of folder names.

Example This example displays descriptions for folders instead of folder names:

```
myExplorer.setUseDescriptionAsLabel(true);
```

showFoldersOnly

Syntax void ReportExplorer.showFoldersOnly(boolean flag)

Sets ReportExplorer to display folders but not files.

Parameter **flag**
Boolean. True displays folders but not files.

Example This example displays folders in ReportExplorer but not files:

```
myExplorer.showFoldersOnly(true);
```

submit

Syntax void ReportExplorer.submit(function callback)

Submits requests to the server for ReportExplorer. When this function is called, an AJAX request is triggered to submit all the operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the ReportExplorer container.

Parameter **callback**
Function. The function to execute after the asynchronous call processing is done.

Example This example submits ReportExplorer with a root folder that set with setStartingFolder( ) and result definition set with setResultDef( ):

```
myExplorer.setStartingFolder("/Public/Contents");
var resultDef = "Name|FileType|Version|VersionName|Description";
myExplorer.setResultDef( resultDef.split("|") );
myExplorer.submit( );
```

Class actuate.reportexplorer.Constants

Description Global constants used for ReportExplorer class. Table 14-34 lists the constants used for the ReportExplorer class.

Table 14-34 Actuate iPortal JavaScript API ReportExplorer constants

Event	Description
ERR_CLIENT	Constant used to tell JSAPI user that there was a client-side error
ERR_SERVER	Constant used to tell JSAPI user that there was a server-side error
ERR_USAGE	Constant used to tell JSAPI user that there was a usage API error
NAV_FIRST	Constant reference for the first page navigation link
NAV_LAST	Constant reference for the last page navigation link
NAV_NEXT	Constant reference for the next page navigation link
NAV_PREV	Constant reference for the previous page navigation link

Class actuate.reportexplorer.EventConstants

Description Defines the event constants supported by this API for report explorers. Table 14-35 lists the ReportExplorer event constants.

Table 14-35 Actuate JavaScript API ReportExplorer event constants

Event	Description
ON_EXCEPTION	Event triggered when an exception occurs. An event handler registered to this event must take an actuate.Exception object as an input argument. The Exception object contains the exception information.
ON_SELECTION_CHANGED	Event triggered when a selection change occurs. For example, this event triggers if the value of a ReportExplorer list control changes. An event handler registered to this event must take an actuate.ReportExplorer.File object corresponding to the file object in which the selection occurred and a string that contains a repository path as input arguments.
ON_SESSION_TIMEOUT	Event triggered when a user attempts to perform any operation after a session has timed out and chooses yes on a prompt dialog asking whether or not to reload the page content. An event handler registered to this event takes no input arguments.

Class actuate.reportexplorer.File

Description A reference object for displaying and controlling a file reference.

Constructor

Syntax actuate.reportexplorer.File( )

Constructs a new File object.

Function summary

Table 14-36 lists actuate.reportexplorer.File functions.

Table 14-36 actuate.reportexplorer.File functions

Function	Description
getAccessType( )	Gets the accessType value for this File
getDescription( )	Gets the description value for this File
getFileType( )	Gets the fileType value for this File
getId( )	Gets the id value for this File
getName( )	Gets the name value for this File
getOwner( )	Gets the owner value for this File
getPageCount( )	Gets the pageCount value for this File
getSize( )	Gets the size value for this File
getTimeStamp( )	Gets the timeStamp value for this File
getUserPermissions( )	Gets the userPermissions value for this File
getVersion( )	Gets the version value for this File
getVersionName( )	Gets the versionName value for this File
setAccessType( )	Sets the accessType value for this File
setDescription( )	Sets the description value for this File
setFileType( )	Sets the fileType value for this File
setId( )	Sets the id value for this File
setName( )	Sets the name value for this File
setOwner( )	Sets the owner value for this File
setPageCount( )	Sets the pageCount value for this File
setSize( )	Sets the size value for this File

(continues)

Table 14-36 actuate.reportexplorer.File functions (continued)

Function	Description
setTimeStamp( )	Sets the timeStamp value for this File
setUserPermissions( )	Sets the userPermissions value for this File
setVersion( )	Sets the version value for this File
setVersionName( )	Sets the versionName value for this File

getAccessType

Syntax string File.getAccessType( )

Gets the access type.

Returns String. Either "private" or "shared" according to whether the file has been shared or not.

Example To stop a script from running if a file is private, use code similar to the following:

```
if(file.getAccessType( ) == "private"){ return;}
```

getDescription

Syntax string File.getDescription( )

Gets the description from the file.

Returns String. The description.

Example To stop a script from running if a file does not have a description, use code similar to the following:

```
if(file.getDescription( ) == (null || "")){ return;}
```

getFileType

Syntax string File.getFileType( )

Gets the file extension for this File.

Returns String. The file type.

Example To store the file extension of the File object file in a variable called type, use code similar to the following:

```
var type = file.getFileType( );
```

getId

Syntax integer File.getId( )

Gets the file ID value.

Returns Integer. The file ID.

Example To store the file id of the File object file in a variable called id, use code similar to the following:

```
var id = file.getFileId( );
```

getName

Syntax string File.getName( )

Gets the name of the file.

Returns String. The file name.

Example To store the name of the File object file in a variable called name, use code similar to the following:

```
var name = file.getName( );
```

getOwner

Syntax string File.getOwner( )

Gets the name of the File's owner.

Returns String. The owner's name

Example To store the name of the owner of the File object file in a variable called owner, use code similar to the following:

```
var owner = file.getOwner( );
```

getPageCount

Syntax integer File.getPageCount( )

Gets the number pages in the file, if applicable.

Returns Integer. The number of pages.

Example To halt a script if the number of pages exceeds 100 in the file referenced by the File object largefile, use code similar to the following:

```
if (largefile.getPageCount( ) > 100) {return;}
```

getSize

Syntax integer File.getSize( )

Gets the size value for this File.

Returns Integer.

Example To store a File object size in a variable called size, use code similar to the following:

```
var size = file.getSize( );
```

getTimeStamp

Syntax string File.getTimeStamp( )

Gets the time stamp for this file.

Returns String. A date and time of the file's creation or last modification.

Example To store the time stamp for the file referenced by the File object oldfile in a variable called timestamp, use code similar to the following:

```
var timestamp = oldfile.getTimeStamp( );
```

getUserPermissions

Syntax string File.getUserPermissions( )

Gets the user permissions.

Returns String. The current user's permissions for this file.

Example To store a file's permissions in the permissions variable, use code similar to the following:

```
var permissions = file.getUserPermissions( );
```

getVersion

Syntax integer File.getVersion( )

Gets the version of the file.

Returns Integer. The version.

Example To store the file version in the version variable, use code similar to the following:

```
var version = file.getVersion( );
```

getVersionName

Syntax string File.getVersionName( )

Gets the version name.

Returns String. The version name.

Example To store a version name in the version variable, use code similar to the following:

```
var version = file.getVersionName( );
```

setAccessType

Syntax void File.setAccessType(string accessType)

Sets the access type.

Parameter **accessType**
String. "private" or "shared" indicating whether the file has been shared or not.

Example To make a file private, use code similar to the following:

```
file.setAccessType("private")
```

setDescription

Syntax void File.setDescription(string description)

Sets the description from the file.

Parameter **description**
String. The description.

Example To clear a file's description, use code similar to the following:

```
file.setDescription("");
```

setFileType

Syntax void File.setFileType(string fileType)

Sets the file type for this file.

Parameter **fileType**
String. The file type, which is a file extension.

Example To assign a file's type if none is assigned, use code similar to the following:

```
if (file.getFileType == null) {file.setFileType("txt");}
```

setId

Syntax void File.setId(integer id)

Sets the file ID value.

Parameter **id**
Integer. A file ID number.

Example To set a file's ID to 42, use code similar to the following:

```
file.setId("42");
```

setName

Syntax void File.setName(string name)

Sets the name of the file.

Parameter **name**
String. The name.

Example To set a file's name to releasedates, use code similar to the following:

```
file.setName("releasedates");
```

setOwner

Syntax void File.setOwner(string owner)

Sets the name of the owner.

Parameter **owner**
String. A user name.

Example To set a file's owner to Administrator, use code similar to the following:

```
file.setOwner("Administrator");
```

setPageCount

Syntax void File.setPageCount(integer pageCount)

Sets the number pages in the file.

Parameter **pageCount**
Integer. The number of pages, which must be less than the current number of pages.

Example To set a File object's page to 100 if available, use code similar to the following:

```
if(file.getPageCount( ) > 100) {file.setPageCount(100);}
```

setSize

Syntax void File.setSize(integer size)

Sets the size of the file.

Parameter **size**
Integer. File size in bytes.

Example To set a file's size to 0, use code similar to the following:

```
file.setSize(0);
```

setTimeStamp

Syntax void File.setTimeStamp(string timeStamp)

Sets the time stamp.

Parameter **timeStamp**
String. A date and time of the file's creation or last modification.

Example To set a file's time stamp to the current time, use code similar to the following:

```
var currenttime = new Date( );
file.setTimeStamp(currenttime.toLocaleString( ));
```

setUserPermissions

Syntax void File.setUserPermissions(string userPermissions)

Sets the user permissions.

Parameter **userPermissions**
String. The current user's permissions for this file.

Example To apply the user permissions for file1 to file2, use code similar to the following:

```
file2.setUserPermissions(file1.getUserPermissions( ));
```

setVersion

Syntax void File.setVersion(integer version)

Sets the version of the file.

Parameter **version**
Integer. The version.

Example To set the file's version to 1 for the first version, use code similar to the following:

```
file.setVersion(1);
```

setVersionName

Syntax void File.setVersionName(string versionName)

Sets the version name.

Parameter **versionName**
String. A version name.

Example To set a file's version name to 2004, use code similar to the following:

```
file.setVersionName("2004");
```

Class actuate.reportexplorer.FileCondition

Description Used inactuate.reportexplorer.FileSearch objects for comparison. Contains a display field associated with a filter string called a match. This can be used for the purposes of comparing field values for searching, filtering, or batch operations. For example, a file condition can match the FileType field with rptdesign to identify all the rptdesign files for a filter.

Constructor

Syntax actuate.reportexplorer.FileCondition( )

Constructs a new FileCondition object.

Function summary

Table 14-37 lists actuate.reportexplorer.FileCondition functions.

Table 14-37 actuate.reportexplorer.FileCondition functions

Function	Description
getField( )	Gets the field for this FileCondition
getMatch( )	Gets the match value for this FileCondition
setField( )	Sets the field for this FileCondition
setMatch( )	Sets the match value for this FileCondition

getField

Syntax string FileCondition.getField( )

Returns the field for this FileCondition.

Returns String. Possible values are "Name", "FileType", "Description", "PageCount", "Size", "TimeStamp", "Version", "VersionName", and "Owner".

Example To store the display field of fcondition in a variable called field, use code similar to the following:

```
var field = fcondition.getField( );
```

getMatch

Syntax string FileCondition.getMatch( )

Returns the match value for this FileCondition.

Returns String. A string for comparison.

Example To store the matching condition of fcondition in a variable called match, use code similar to the following:

```
var match = fcondition.getMatch( );
```

setField

Syntax void FileCondition.setField(string field)

Sets the field for the FileCondition.

Parameter **field**
String. Possible values are "Name", "FileType", "Description", "PageCount", "Size", "TimeStamp", "Version", "VersionName", and "Owner".

Example To set the display field to FileType for fcondition, use code similar to the following:

```
fcondition.setField("FileType");
```

setMatch

Syntax void FileCondition.setMatch(string match)

Sets the match value for the FileCondition.

Parameter **match**
String. A string for comparison.

Example To set the match value for fcondition to rptdesign, use code similar to the following:

```
fcondition.setField("rptdesign");
```

Class actuate.reportexplorer.FileSearch

Description Searches the contents of files according to one or more file conditions. FileSearch represents a JavaScript version of com.actuate.schemas.FileSearch.

Constructor

Syntax actuate.reportexplorer.FileSearch( )

Constructs a new FileSearch object.

Function summary

Table 14-38 lists actuate.reportexplorer.FileSearch functions.

Table 14-38 actuate.reportexplorer.FileSearch functions

Function	Condition
getAccessType( )	Gets the accessType value for this FileSearch
getCondition( )	Gets the condition value for this FileSearch
getConditionArray( )	Gets the ConditionArray value for this FileSearch
getCountLimit( )	Gets the countLimit value for this FileSearch
getDependentFileId( )	Gets the id value for this FileSearch
getDependentFileName( )	Gets the file name value for this FileSearch
getFetchDirection( )	Gets the fetch direction for this FileSearch
getFetchHandle( )	Gets the fetchHandle value for this FileSearch
getFetchSize( )	Gets the fetchSize value for this FileSearch
getIncludeHiddenObject( )	Gets the includeHiddenObject value for this FileSearch
getOwner( )	Gets the owner
getPrivilegeFilter( )	Gets the privilegeFilter value for this FileSearch
getRequiredFileId( )	Gets the requiredFileId for this FileSearch
getRequiredFileName( )	Gets the requiredFileName value for this FileSearch
setAccessType( )	Sets the accessType value for this FileSearch
setCondition( )	Sets the condition value for this FileSearch
setConditionArray( )	Sets the ConditionArray value for this FileSearch

(continues)

Table 14-38 actuate.reportexplorer.FileSearch functions (continued)

Function	Condition
setCountLimit( )	Sets the id value for this FileSearch
setDependentFileId( )	Sets the id value for this FileSearch
setDependentFileName( )	Sets the file name value for this FileSearch
setFetchDirection( )	Sets the owner value for this FileSearch
setFetchHandle( )	Sets the fetchHandle value for this FileSearch
setFetchSize( )	Sets the fetchSize value for this FileSearch
setIncludeHiddenObject( )	Sets the timeStamp value for this FileSearch
setOwner( )	Sets the Owner
setPrivilegeFilter( )	Sets the PrivilegeFilter value for this FileSearch
setRequiredFileId( )	Sets the requiredFileId for this FileSearch
setRequiredFileName( )	Sets the requiredFileName value for this FileSearch

getAccessType

Syntax string FileSearch.getAccessType( )

Gets the access type.

Returns String. Either "private" or "shared" according to whether the FileSearch has been shared or not.

Example To halt a script if a FileSearch is private, use code similar to the following:

```
if(fsearch.getAccessType( ) == "private"){ return;}
```

getCondition

Syntax actuate.reportexplorer.FileCondition FileSearch.getCondition( )

Gets the condition from the FileSearch.

Returns actuate.reportexplorer.FileCondition object. A condition to apply in a search.

Example To halt a script if a FileSearch does not have a condition, use code similar to the following:

```
if(fsearch.getCondition( ) == null){ return;}
```

getConditionArray

Syntax actuate.reportexplorer.FileCondition[ ] FileSearch.getConditionArray( )

Gets the file condition array for this FileSearch.

Returns Array of actuate.reportexplorer.FileCondition objects. Multiple conditions to apply in a search.

Example To retrieve the array of file conditions from the FileSearch object fsearch, use code similar to the following:

```
var conditions = new Array( );
conditions = fsearch.getConditionArray( );
```

getCountLimit

Syntax integer FileSearch.getCountLimit( )

Gets the maximum number of match results to display set for this file search.

Returns Integer. The maximum number of match results to display. 0 indicates unlimited.

Example To retrieve the count limit from the FileSearch object fsearch, use code similar to the following:

```
var limit = fsearch.getCountLimit( );
```

getDependentFileId

Syntax string FileSearch.getDependentFileId( )

Gets the file ID of the FileSearch, identifying the file it is set to search.

Returns String. The file ID.

Example To retrieve the file Id from the FileSearch object fsearch, use code similar to the following:

```
var id = fsearch.getDependantFileId( );
```

getDependentFileName

Syntax string FileSearch.getDependentFileName( )

Gets the file name of the FileSearch.

Returns String. The file name.

Example To retrieve the file name from the FileSearch object fsearch, use code similar to the following:

```
var name = fsearch.getDependantFileName( );
```

getFetchDirection

Syntax boolean FileSearch.getFetchDirection( )

Gets the fetch direction of the FileSearch.

Returns Boolean. True indicates ascending order.

Example To switch the fetch direction for the FileSearch object fsearch, use code similar to the following:

```
fsearch.setFetchDirection(!fsearch.getFetchDirection( ));
```

getFetchHandle

Syntax string FileSearch.getFetchHandle( )

Gets the fetch handle.

Returns String. The fetch handle.

Example To retrieve the fetch handle from the FileSearch object fsearch, use code similar to the following:

```
var handle = fsearch.getFetchHandle( );
```

getFetchSize

Syntax integer FileSearch.getFetchSize( )

Gets the fetch size.

Returns Integer. The fetch size.

Example To halt a script if a FileSearch has a fetch size of 0, use code similar to the following:

```
if(fsearch.getFetchSize( ) == 0){ return;}
```

getIncludeHiddenObject

Syntax boolean FileSearch.getIncludeHiddenObject( )

Gets the includeHiddenObject value for this FileSearch.

Returns Boolean. True includes hidden object.

Example To alert the user that hidden objects are enabled for a FileSearch, use code similar to the following:

```
if(fsearch.getIncludeHiddenObejct( )){
   alert("Hidden objects are enabled.");
}
```

getOwner

Syntax string FileSearch.getOwner( )

Gets the owner's name.

Returns String. The owner's user name.

Example To retrieve the owner of fsearch, use code similar to the following:

```
var owner = fsearch.getOwner( );
```

getPrivilegeFilter

Syntax actuate.reportexplorer.PrivilegeFilter FileSearch.getPrivilegeFilter( )

Gets the privilege filter.

Returns actuate.reportexplorer.PrivilegeFilter object. A privilege filter.

Example To retrieve the privilege filter for fsearch, use code similar to the following:

```
var privileges = fsearch.getPrivilegeFilter( );
```

getRequiredFileId

Syntax integer FileSearch.getRequiredFileId( )

Gets the requiredFileId of FileSearch.

Returns Integer. A field ID.

Example To retrieve the required field ID assigned to fsearch, use code similar to the following:

```
var id = fsearch.getRequiredFileId( );
```

getRequiredFileName

Syntax string FileSearch.getRequiredFileName( )

Gets the requiredFileName name.

Returns String. A file name.

Example To retrieve the file name assigned to fsearch, use code similar to the following:

```
var id = fsearch.getRequiredFileName( );
```

setAccessType

Syntax void FileSearch.setAccessType(string accessType)

Sets the access type.

Parameter **accessType**
String. Either "private" or "shared" according to whether FileSearch has been shared or not.

Example To make a FileSearch fsearch private, use code similar to the following:

```
fsearch.setAccessType("private");
```

setCondition

Syntax void FileSearch.setCondition(actuate.reportExplorer.FileCondition condition)

Sets a search condition for this FileSearch.

Parameter **condition**
actuate.reportexplorer.FileCondition object. A condition to apply to this search.

Example To clear FileSearch fsearch's condition, use code similar to the following:

```
fsearch.setCondition(null);
```

setConditionArray

Syntax void FileSearch.setConditionArray(actuate.reportExplorer.FileCondition[ ] ConditionArray)

Sets multiple search conditions for this FileSearch.

Parameter **ConditionArray**
Array of actuate.reportexplorer.FileCondition objects. Conditions to apply to this search.

Example To clear FileSearch fsearch's condition array, use code similar to the following:

```
fsearch.setConditionArray(null);
```

setCountLimit

Syntax void FileSearch.setCountLimit(integer countlimit)

Sets the maximum number of match results to display.

Parameter **countlimit**
Integer. The maximum number of match results to display. 0 indicates unlimited.

Example To set FileSearch fsearch to stop searching after finding 100 matches, use code similar to the following:

```
fsearch.setCountLimit(100);
```

setDependentFileId

Syntax void FileSearch.setDependentFileId(string dependentFileId)

Sets the file ID of the FileSearch.

Parameter **dependentFileId**
String. A file ID.

Example To set FileSearch fsearch's File ID to current, use code similar to the following:

```
fsearch.setDependentFileId("current");
```

setDependentFileName

Syntax void FileSearch.setDependentFileName(string dependentFileName)

Sets the file name of FileSearch.

Parameter **dependentFileName**
String. A file name.

Example To set FileSearch fsearch's file name to current, use code similar to the following:

```
fsearch.setDependentFileName("current");
```

setFetchDirection

Syntax void FileSearch.setFetchDirection(boolean fetchDirection)

Sets the fetch direction for this FileSearch.

Parameter **fetchDirection**
Boolean. True indicates ascending order.

Example To switch the fetch direction for the FileSearch object fsearch, use code similar to the following:

```
fsearch.setFetchDirection(!fsearch.getFetchDirection( ));
```

setFetchHandle

Syntax void FileSearch.setFetchHandle(string fetchHandle)

Sets the fetch handle for FileSearch.

Parameter **fetchHandle**
String. A fetch handle.

Example To set FileSearch fsearch's fetch handle to ezsearch, use code similar to the following:

```
fsearch.setFetchHandle("ezsearch");
```

setFetchSize

Syntax void FileSearch.setFetchSize(integer fetchSize)

Sets the fetch size.

Parameter **fetchSize**
Integer. The fetch size.

Example To set FileSearch fsearch's fetch size to 12, use code similar to the following:

```
fsearch.setFetchSize(12);
```

setIncludeHiddenObject

Syntax void FileSearch.setIncludeHiddenObject(boolean includeHiddenObject)

Sets the includeHiddenObject value for this FileSearch.

Parameter **includeHiddenObject**
Boolean. True includes hidden object.

Example To prohibit FileSearch fsearch from including hidden objects, use code similar to the following:

```
fsearch.setIncludeHiddenObject(false);
```

setOwner

Syntax void FileSearch.setOwner(string owner)

Sets the owner for this FileSearch.

Parameter **owner**
String. The owner's user name.

Example To set FileSearch fsearch's owner to administrator, use code similar to the following:

```
fsearch.setOwner("administrator");
```

setPrivilegeFilter

Syntax void FileSearch.setPrivilegeFilter(actuate.reportexplorer.PrivilegeFilter privilegeFilter)

Sets the privilege filter.

Parameter **privilegeFilter**
actuate.reportexplorer.PrivilegeFilter object. The privilege filter.

Example To assign the privilege filter pfilter to the FileSearch fsearch, use code similar to the following:

```
fsearch.setPrivilegeFilter(pfilter);
```

setRequiredFileId

Syntax void FileSearch.setRequiredFileId(string requiredFileId)

Sets the requiredFileId for this FileSearch.

Parameter **requiredFileId**
String. A file ID.

Example To set FileSearch fsearch's File ID to permanent, use code similar to the following:

```
fsearch.setRequiredFileId("permanent");
```

setRequiredFileName

Syntax void FileSearch.setRequiredFileName(string requiredFileName)

Sets the required file name.

Parameter **requiredFileName**
String. A file name.

Example To set FileSearch fsearch's file name to permanent, use code similar to the following:

```
fsearch.setRequiredFileName("permanent");
```

Class actuate.reportexplorer.FolderItems

Description A container for the contents of a folder. FolderItems represents a JavaScript version of com.actuate.schemas.GetFolderItemsResponse.

Constructor

Syntax actuate.reportexplorer.FolderItems( )

Constructs a new FolderItems object.

Function summary

Table 14-39 lists actuate.reportexplorer.FolderItems functions.

Table 14-39 actuate.reportexplorer.FolderItems functions

Function	Description
getFetchHandle( )	Gets the fetchHandle value for GetFolderItemsResponse
getItemList( )	Gets the itemList value for GetFolderItemsResponse
getTotalCount( )	Gets the totalCount value for GetFolderItemsResponse
setFetchHandle( )	Sets the fetchHandle value for GetFolderItemsResponse
setItemList( )	Sets the itemList value for GetFolderItemsResponse
setTotalCount( )	Sets the totalCount value for GetFolderItemsResponse

getFetchHandle

Syntax string FolderItems.getFetchHandle( )

Retrieves the fetch handle for this folder's contents.

Returns String. The fetch handle.

Example To retrieve the fetch handle from fitems, use code similar to the following:

```
var handle = fitems.getFetchHandle( );
```

getItemList

Syntax actuate.reportexplorer.File[ ] FolderItems.getItemList( )

Gets the list of file contents for the folder.

Returns Array of actuate.reportexplorer.File objects. A list of the folder contents.

Example To store fitems' item list in the files variable, use code similar to the following:

```
files = fitems.getItemList( );
```

getTotalCount

Syntax string FolderItems.getTotalCount( )

Returns the maximum number of list items to retrieve from this folder.

Returns String. The total count.

Example To retrieve the total count from fitems, use code similar to the following:

```
var count = fitems.getTotalCount( );
```

setFetchHandle

Syntax void FolderItems.setFetchHandle(string fetchHandle)

Sets the fetch handle value for this FolderItems object.

Parameter **fetchHandle**
String. The fetch handle.

Example To set FolderItems fitems' fetch handle to dir, use code similar to the following:

```
fitems.setFetchHandle("dir");
```

setItemList

Syntax void FolderItems.setItemList(ctuate.reportexplorer.File[ ] itemList)

Sets the list of contents for this folder.

Parameter **itemList**
Array of actuate.reportexplorer.File objects. A list of the folder contents.

Example To assign the item list from fitems1 to fitems2, use code similar to the following:

```
fitems2.setItemList(fitems1.getItemList( ));
```

setTotalCount

Syntax void FolderItems.setDataType(string totalCount)

Sets the maximum number of list items to retrieve from this folder.

Parameter **totalCount**
String. The total count.

Example To reset the count total for fitems, use code similar to the following:

```
fitems.setTotalCount("0");
```

Class actuate.reportexplorer.PrivilegeFilter

Description The PrivilegeFilter class contains a set of user-identifying information and access rights that are associated with identified users. PrivilegeFilter represents a JavaScript version of com.actuate.schemas.PrivilegeFilter.

Constructor

Syntax actuate.reportexplorer.PrivilegeFilter( )

Constructs a new PrivilegeFilter object.

Function summary

Table 14-40 lists actuate.reportexplorer.PrivilegeFilter functions.

Table 14-40 actuate.reportexplorer.PrivilegeFilter functions

Function	Description
getAccessRights( )	Gets the accessRights value for this PrivilegeFilter
getGrantedRoleId( )	Gets the grantedRoleId value for this PrivilegeFilter
getGrantedRoleName( )	Gets the grantedRoleName value for this PrivilegeFilter
getGrantedUserId( )	Gets the grantedUserId value for this PrivilegeFilter
getGrantedUserName( )	Gets the grantedUserName value for this PrivilegeFilter
setAccessRights( )	Sets the accessRights value for this PrivilegeFilter
setGrantedRoleId( )	Sets the grantedRoleId value for this PrivilegeFilter
setGrantedRoleName( )	Sets the grantedRoleName value for this PrivilegeFilter
setGrantedUserId( )	Sets the grantedUserId value for this PrivilegeFilter
setGrantedUserName( )	Sets the grantedUserName value for this PrivilegeFilter

getAccessRights

Syntax string privilegeFilter.getAccessRights( )

Gets the repository access rights value for this PrivilegeFilter.

Returns String. Repository access rights.

Example To halt a script if a PrivilegeFilter pfilter's access rights are null, use code similar to the following:

```
if(pfilter.getAccessRights( ) == null){ return;}
```

getGrantedRoleId

Syntax string PrivilegeFilter.getGrantedRoleId( )

Gets the grantedRoleId value for this PrivilegeFilter.

Returns String. A role ID.

Example To retrieve the granted role ID for a PrivilegeFilter pfilter, use code similar to the following:

```
var roleid = pfilter.getGrantedRoleId( );
```

getGrantedRoleName

Syntax string PrivilegeFilter.getGrantedRoleName( )

Gets the grantedRoleName value for this PrivilegeFilter.

Returns String. A role name.

Example To retrieve the granted role name for a PrivilegeFilter pfilter, use code similar to the following:

```
var rolename = pfilter.getGrantedRoleName( );
```

getGrantedUserId

Syntax string PrivilegeFilter.getGrantedUserId( )

Gets the grantedUserId value for this PrivilegeFilter.

Returns String. A user ID.

Example To retrieve the granted user ID for a PrivilegeFilter pfilter, use code similar to the following:

```
var userid = pfilter.getGrantedUserId( );
```

getGrantedUserName

Syntax string PrivilegeFilter.getGrantedUserName( )

Gets the grantedUserName value for this PrivilegeFilter.

Returns String. A user name.

Example To retrieve the granted user name for a PrivilegeFilter pfilter, use code similar to the following:

```
var username = pfilter.getGrantedUserName( );
```

setAccessRights

Syntax void PrivilegeFilter.setAccessRights(string accessRights)

Sets the repository access rights value for this PrivilegeFilter.

Parameter **accessRights**
String. The access rights.

Example To copy the set of access rights from PrivilegeFilter pfilter1 to PrivilegeFilter pfilter2, use code similar to the following:

```
pfilter2.setAccessRights(pfilter1.getAccessRights( ));
```

setGrantedRoleId

Syntax void PrivilegeFilter.setGrantedRoleId(string grantedRoleId)

Sets the grantedRoleId of the column for this PrivilegeFilter.

Parameter **grantedRoleId**
String. A role ID.

Example To set the granted role ID of the PrivilegeFilter pfilter to All, use code similar to the following:

```
pfilter.setGrantedRoleId("All");
```

setGrantedRoleName

Syntax void PrivilegeFilter.setGrantedRoleName(string grantedRoleName)

Sets the grantedRoleName value for this PrivilegeFilter.

Parameter **grantedRoleName**
String. A role name.

Example To set the granted role name of the PrivilegeFilter pfilter to Everyone, use code similar to the following:

```
pfilter.setGrantedRoleName("Everyone");
```

setGrantedUserId

Syntax void PrivilegeFilter.setGrantedUserId(string grantedUserId)

Sets the grantedUserId value for this PrivilegeFilter.

Parameter **grantedUserId**
String. A user ID.

Example To set the granted user ID of the PrivilegeFilter pfilter to administrator, use code similar to the following:

```
pfilter.setGrantedRoleId("Administrator");
```

setGrantedUserName

Syntax void PrivilegeFilter.setGrantedUserName(string grantedUserName)

Sets the grantedUserName value for this PrivilegeFilter.

Parameter **grantedUserName**
String. A user name.

Example To set the granted user name of the PrivilegeFilter pfilter to administrator, use code similar to the following:

```
pfilter.setGrantedRoleId("Administrator");
```

Class actuate.RequestOptions

Description The request options that loginServlet requires to authenticate requests. RequestOptions is used by other classes to provide authentication information. It also adds any customized options to the request URL.

Constructor

Syntax actuate.RequestOptions(actuate.RequestOptions requestOptions)

Constructs a new RequestOptions object.

Parameter **requestOptions**
actuate.RequestOptions object. Optional. Provides request option settings to copy into this RequestOptions object.

Function summary

Table 14-41 lists actuate.RequestOptions functions.

Table 14-41 actuate.RequestOptions functions

Function	Description
getLocale( )	Returns the current locale
setCustomParameters( )	Appends custom parameters to the request URL
setLocale( )	Sets the locale

getLocale

Syntax string RequestOptions.getLocale( )

Returns the current locale or null if no locale is set.

Returns String. The locale value; null for default.

Example This example pops up an alert box if the locale value is set to the default:

```
var locale = reqOpts.getLocale( );
if (locale == null){
   alert("Locale value is default");
}
```

setCustomParameters

Syntax void RequestOptions.setCustomParameters(object parameters)

Returns a custom parameter in the request URL.

Parameter **parameters**
Object. An associative array of name:value pairs for URL parameters.

Example To add "&myParam=myValue" in a request URL derived from RequestOptions object, use code similar to the following:

```
MyRequestOptions.setCustomParameters({myParam: "myValue"});
```

setLocale

Syntax void RequestOptions.setLocale(string Locale)

Sets the locale.

Parameter **Locale**
String. Optional. The locale value. Null indicates the default locale.

Example This example resets the locale for the reqOpts RequestOptions object to the default value provided by the actuate web service to which the JSAPI connects:

```
reqOpts.setLocale( );
```

This example resets the locale for the reqOpts RequestOptions object to the Spain using the Spanish locale code listed in <context root>\WEB-INF\localemap.xml:

```
reqOpts.setLocale("es_ES");
```

Class actuate.Viewer

Description The actuate.Viewer class retrieves and displays Actuate BIRT report contents in an HTML container. The actuate.Viewer class displays the report by page. The goto functions of this class change the current position and page displayed in the viewer.

Constructor

Syntax actuate.Viewer(object viewContainer)

actuate.Viewer(string viewContainerId)

Constructs a new viewer object. The container is an HTML object defined on the HTML page.

Parameters

viewContainer
Object. A document object that references the element that holds the viewer.

viewContainerId
String. The value of the id parameter for the element that holds the viewer.

Example To assign the viewer to display in a tag on the page, use the following constructor call:

```
var myViewer = new actuate.Viewer("containerName");
```

Function summary

Table 14-42 lists actuate.Viewer functions.

Table 14-42 actuate.Viewer functions

Function	Description
disableIV( )	Disables Interactive Viewer features
downloadReport( )	Exports a report using the specified format
downloadResultSet( )	Exports data to an external file
enableIV( )	Enables Interactive Viewer features
getChart( )	Retrieves a chart by bookmark
getClientHeight( )	Gets the viewer's height
getClientWidth( )	Gets the viewer's width
getContentByBookmark( )	Gets the report content by bookmark
getContentByPageRange( )	Gets the report content by page range

Table 14-42 actuate.Viewer functions (continued)

Function	Description
getContentMargin( )	Gets the margin dimensions of the content in pixels
getCurrentPageContent( )	Returns the report content displayed in the viewer
getCurrentPageNum( )	Returns the current page number
getDataItem( )	Retrieves a data item by bookmark
getFlashObject( )	Retrieves a Flash object by bookmark
getGadget( )	Retrieves a gadget by bookmark
getHeight( )	Returns the viewer height setting
getHelpBase( )	Gets the help URL
getLabel( )	Returns the ID of this object
getReportletBookmark( )	Returns the Actuate web application URL that this Viewer accesses
getReportName( )	Retrieves a label by bookmark
getTable( )	Returns the bookmark of a Reportlet displayed in the viewer
getText( )	Returns the report file displayed in the viewer
getTotalPageCount( )	Returns the viewer's request options
getUIConfig( )	Retrieves a table by bookmark
getUIOptions( )	Retrieves a text element by bookmark
getViewer( )	Returns the total number of pages
getWidth( )	Gets the UIConfig object assigned to the viewer
gotoBookmark( )	Returns the UIOptions object
gotoPage( )	Returns a viewer object containing the given bookmarked element
isInteractive( )	Returns the viewer width setting
saveReportDesign( )	Goes to the position in the report specified by the bookmark
saveReportDocument( )	Goes to the specified page
setContentMarg( )	Returns whether interactive viewing features are enabled

(continues)

Table 14-42 actuate.Viewer functions (continued)

Function	Description
setFocus( )	Saves a report design to the repository
setHeight( )	Saves a report document to the repository
setHelpBase( )	Sets the viewer content margin
setParameters( )	Sets the focus element on the viewer
setParameterValues( )	Sets the viewer height
setReportletBookmark( )	Sets the base help URL
setReportName( )	Sets the parameters to run a report using a list of literal string pairs
setService( )	Sets the parameters to run a report using a generated object
setSize( )	Sets bookmark name for a Reportlet
setSupportSVG( )	Sets the report file to render within this Viewer
setService( )	Sets the target service URL
setSize( )	Sets the size of the viewer
setSupportSVG( )	Sets the Scalable Vector Graphic support flag to enable Scalable Vector Graphics content
setUIOptions( )	Sets UIOptions using a UIOptions object
setWidth( )	Sets the width of the viewer
showDownloadReportDialog( )	Enables the export report dialog window
showDownloadResultSetDialog( )	Enables the download data dialog window
showFacebookCommentPanel( )	Shows the Facebook comments panel.
showParameterPanel( )	Shows the parameter panel
showPrintDialog( )	Enables the print dialog window
showTocPanel( )	Shows the table of contents panel
submit( )	Submits all the asynchronous operations that the user has requested on this Viewer and renders the viewer component on the page

disableIV

Syntax void Viewer.disableIV(function callback)

Disables the Interactive Viewer features of this viewer object. This is an asynchronous setting committed by submit( ).

Parameter **callback**
Function. The callback function to call after the Interactive Viewer is disabled.

Example To disable the Interactive Viewer option for myViewer, use code similar to the following:

```
myViewer.disableIV(function alertUser( ){alert("IV disabled");});
```

downloadReport

Syntax void Viewer.downloadReport(string format, string pages, actuate.viewer.RenderOptions renderoption)

Exports the report with a specified format. The downloadReport function does not return any object. The report is exported to the client side. Then the browser opens a download window for the user to specify a location for the report.

Parameters **format**
String. The format in which to export the report. Valid values and their corresponding formats are:

- afp: IBM Advanced Function Printing
- doc: Word
- docx: Word 2007
- html: HTML-encoded web page
- ppt: PowerPoint
- pptx: PowerPoint 2007
- pdf: Adobe PDF
- ps: PostScript
- xls: Excel
- xlsx: Excel 2007

pages
String. The pages to retrieve. Indicate page ranges by using the first page number of the range and the last page number separated by a dash. To use more than one value, separate individual page numbers or page ranges with commas.

renderoption
actuate.viewer.RenderOptions object. Optional. Sets the rendering options for the download, which currently only applies to multisheet xls format reports.

Example To download the first five pages of the report in the viewer, use the following code:

```
viewer.downloadReport("pdf", "1-5", null);
```

downloadResultSet

Syntax void Viewer.downloadResultSet(actuate.data.Request request, function callback)

Gets all the data from the report as specified by the request. This function makes an AJAX call to the server for the data that is not in the current page. Write a callback function to process the result set. The callback must take an actuate.data.ResultSet object as an argument.

Parameters

request
actuate.data.Request object. The request to generate the result set.

callback
Function. The callback function to call after retrieving the results. The callback function must take an actuate.data.ResultSet object as an argument.

Example This example creates an actuate.data.ResultSet object from the report in myViewer as specified by myRequest and passes it to a callback function:

```
myViewer.downloadResultSet(myRequest, callback);
```

enableIV

Syntax void Viewer.enableIV(function callback)

Enables interactive viewing features for this Viewer, which enables the selection and modification of report content. This function must be used in the callback of viewer.submit( ) as shown in the following example:

```
function runInteractive(){
myviewer.setReportName("/Public/BIRT and BIRT Studio Examples
   /Sales by Customer.rptdesign");
myviewer.submit(function() {myviewer.enableIV(callback);});
}
```

Parameter

callback
Function. The callback function to call after enabling the Interactive Viewer features.

Example This function must be used in the callback of viewer.submit( ) as shown in the following example:

```
function runInteractive(){
myviewer.setReportName("/Public/BIRT and BIRT Studio Examples
   /Sales by Customer.rptdesign");
myviewer.submit(function() {myviewer.enableIV(callback);});
}
```

getChart

Syntax actuate.report.Chart Viewer.getChart(string bookmark)

Returns an instance of the chart referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.Chart object.

Example This example returns the chart with the bookmark ChartBookmark:

```
function getMyChartByBookmark(myReport) {
   var bviewer = myReport.getViewer("Chart");
   var bpagecontents = bviewer.getCurrentPageContent( );
   return bpagecontents.getChart("ChartBookmark");
}
```

getClientHeight

Syntax integer Viewer.getClientHeight( )

Gets the browser window's height.

Returns Integer. Height in pixels.

Example To reset the viewer height to 20 pixels less than the browser window if it is larger than the browser window, use code similar to the following:

```
if(myViewer.getClientHeight( ) < myViewer.getHeight( )){
   myViewer.setHeight(myViewer.getClientHeight( ) - 20);
}
```

getClientWidth

Syntax integer Viewer.getClientWidth( )

Gets the browser window's width.

Returns Integer. Width in pixels.

Example To reset the viewer width to 20 pixels less than the browser window if it is larger than the browser window, use code similar to the following:

```
if(myViewer.getClientWidth( ) < myViewer.getWidth( )){
   myViewer.setWidth(myViewer.getClientWidth( ) - 20);
}
```

getContentByBookmark

Syntax void Viewer.getContentByBookmark(string bookmark, string format, function callback)

Gets the report content by bookmark and passes the content as data to a callback.

Parameters

bookmark
String. The bookmark of a report element to retrieve.

format
String. The output format, which is either html or xhtml.

callback
Function. Callback to be called once the operation is finished. The callback must take actuate.data.ReportContent object as an argument.

Example To retrieve the content with the bookmark FirstChart as html, use code similar to the following:

```
myViewer.getContentByBookmark("FirstChart", "html", processChart);
```

getContentByPageRange

Syntax void Viewer.getContentByPageRange(string PageRange, string format, function callback)

Gets the report content by Page Range and passes the content as data to a callback.

Parameters

PageRange
String. Page range to retrieve the report content, separated by a dash.

format
String. The output format, which is either html or xhtml.

callback
Function. Callback to be called once the operation is finished. The callback must take actuate.data.ReportContent object as an argument.

Example To retrieve the content from pages 3 through 5 as html, use code similar to the following:

```
myViewer.getContentByPageRange("3-5", "html", processPages);
```

getContentMargin

Syntax integer | object Viewer.getContentMargin( )

Gets the viewer content margin.

Returns Integer or Object. An integer indicates the same margin on all sides, in pixels. The object contains the pixel values for the top, bottom, left, and right margins of the viewer in an array. For example, a 25-pixel top content margin and no margin in the other directions would be the object array {top:25, left:0, right:0, bottom:0}.

Example To set the margin of the viewer newViewer to match the margin of myViewer, use code similar to the following:

```
newViewer.setContentMargin(myViewer.getContentMargin( ));
```

getCurrentPageContent

Syntax actuate.viewer.Content Viewer.getCurrentPageContent( )

Returns the report content displayed in the viewer. This function is the entry point for retrieving the report elements from this viewer object.

Returns actuate.viewer.PageContent object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the table "mytable" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( ).
   getTableByBookmark("mytable");
```

getCurrentPageNum

Syntax integer Viewer.getCurrentPageNum( )

Returns the page number for the page currently being displayed.

Returns Integer. The current page number.

Example This function is useful to move to another page relative to the current page. To go to the next page in a document, use the following code:

```
viewer.gotoPage(viewer.getCurrentPageNum( ) + 1);
```

getDataItem

Syntax actuate.report.DataItem Viewer.getDataItem(string bookmark)

Returns an instance of report data referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.DataItem object.

Example To get the report data with the bookmark FirstDataItem and store it in the variable myDataItem, use code similar to the following:

```
var myDataItem = myViewer.getDataItem("FirstDataItem");
```

getFlashObject

Syntax actuate.report.FlashObject Viewer.getFlashObject(string bookmark)

Returns an instance of the Flash object referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.FlashObject object.

Example To get the Flash object with the bookmark FirstFlashObject and store it in the variable myFlashObject, use code similar to the following:

```
var myFlashObject = myViewer.getFlashObject("FirstFlashObject");
```

getGadget

Syntax actuate.report.Gadget Viewer.getGadget(string bookmark)

Returns an instance of the gadget referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.Gadget object.

Example To get the gadget with the bookmark FirstGadget and store it in the variable myGadget, use code similar to the following:

```
var myGadget = myViewer.getGadget("FirstGadget");
```

getHeight

Syntax string Viewer.getHeight( )

Returns the height value of the viewer.

Returns String.

Example This example decreases the viewer's height by 10:

```
var height = myViewer.getHeight( );
myViewer.setHeight(height - 10);
```

getHelpBase

Syntax string Viewer.getHelpBase( )

Returns the URL of the help base. The help base is the base URL for the product help documentation.

Returns String. The base URL of the help documentation.

Example This example displays the help base URL in an alert box:

```
alert("The help base URL is " + myViewer.getHelpBase( ))
```

getLabel

Syntax actuate.report.Label Viewer.getLabel(string bookmark)

Returns an instance of the label referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.Label object.

Example To get the label with the bookmark FirstLabel and store it in the variable myLabel, use code similar to the following:

```
var myLabel = myViewer.getLabel("FirstLabel");
```

getReportletBookmark

Syntax string Viewer.getReportletBookmark( )

Returns the bookmark of the current report page or element.

Returns String. Bookmark.

Example This example displays the bookmark of the current report page in an alert box:

```
alert ("Report bookmark is " + myViewer.getReportletBookmark( ));
```

getReportName

Syntax string Viewer.getReportName( )

Returns the name of the report file, either a report design file or report document file, that is currently displayed in this Viewer.

Returns String.

Example This example displays the currently displayed report file name in an alert box:

```
alert ("Currently displaying " + myViewer.getReportName( ));
```

getTable

Syntax actuate.report.Table Viewer.getTable(string bookmark)

Returns an instance of the table referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.Table object.

Example To get the table with the bookmark FirstTable and store it in the variable myTable, use code similar to the following:

```
var myTable = myViewer.getTable("FirstTable");
```

getText

Syntax actuate.report.Text Viewer.getText(string bookmark)

Returns an instance of the Text object referenced by a bookmark.

Parameter **bookmark**
String. The bookmark name.

Returns actuate.report.Text object.

Example To get the Text object with the bookmark Title and store it in the variable myText, use code similar to the following:

```
var myText = myViewer.getText("Title");
```

getTotalPageCount

Syntax integer Viewer.getTotalPageCount( )

Returns the total number of pages in the report being viewed.

Returns Integer.

Example This function is useful to move to the last page of a document. To go to the last page in a document, use the following code:

```
viewer.gotoPage(viewer.getTotalPageCount( ));
```

getUIConfig

Syntax actuate.viewer.UIConfig Viewer.getUIConfig( )

Returns the current UI configuration.

Returns actuate.viewer.UIConfig object. This function returns null when no UIConfig object is set.

Example To retrieve and store the content pane from the viewer, use the following code:

```
var contentpane = viewer.getUIConfig( ).getContentPane( );
```

getUIOptions

Syntax actuate.viewer.UIOptions Viewer.getUIOptions( )

Returns the UIOptions set in this viewer object.

Returns actuate.viewer.UIOptions object. This function returns null when no UIOptions object is set.

Example To retrieve and store the uiOptions for the viewer, use the following code:

```
var options = myViewer.getUIOptions( );
```

getViewer

Syntax actuate.Viewer Viewer.getViewer(string bookmark)

actuate.Viewer Viewer.getViewer(object elementID)

Returns a viewer object containing the report element that is associated with a bookmark or contained in an HTML element.

Parameters

bookmark
String. The bookmark of the report element to view.

elementID
Object. An HTML element that contains the viewer.

Returns actuate.Viewer object or null if the viewer is not found.

Example This example uses getViewer( ) to retrieve a report element and return the bookmark of the chart in that report:

```
function chartBookmark(myReport){
   var bviewer = myReport.getViewer("Chart");
   var bpagecontents = bviewer.getCurrentPageContent( );
   return bpagecontents.getChartByBookmark("ChartBookmark");
}
```

getWidth

Syntax string Viewer.getWidth( )

Returns the width value of the viewer.

Returns String.

Example This example decreases the viewer's width by 10:

```
var width = myViewer.getWidth( );
myViewer.setWidth(width - 10);
```

gotoBookmark

Syntax void Viewer.gotoBookmark(string bookmark)

Goes to the page position by the specified bookmark. The viewer displays to the first page when the bookmark is not found.

Parameter **bookmark**
String. The bookmark of a report element.

Example To move the viewer to the page position specified by the value of the 'bookmark' parameter, use this code:

```
viewer.gotoBookmark(document.getElementById('bookmark').value);
```

gotoPage

Syntax void Viewer.gotoPage(integer pageNumber)

Goes to the specified page. The viewer throws an exception when the page is not found.

Parameter **pageNumber**
Integer. A page number in the report.

Example To go to the first page of a report, use the following code:

```
viewer.gotoPage(1);
```

isInteractive

Syntax boolean Viewer.isInteractive( )

Returns the interactive viewing status of the viewer. Enables or disables the interactive viewing features with actuate.Viewer.enableIV( ).

Returns Boolean. True when interactive viewing features are enabled.

Example This example displays an alert box with the interactive status of the viewer:

```
alert("Interactivity of this viewer is set to " +
   myViewer.isInteractive( ));
```

saveReportDesign

Syntax void Viewer.saveReportDesign(string filename, function callback)

Saves the current viewer content as a report design. The viewer must enable interactive viewing with enableIV( ) prior to saving a report design.

Parameters **filename**
String. Sets the name of the saved file. The current file name is used if null. The file name must be a path relative to the viewer's repository.

callback
Function. Optional. The function to execute after the asynchronous call processing is done. The callback takes the current actuate.Viewer object as an input parameter.

Example To save the content of the viewer as the report design called NewDesign, use the following code:

```
myViewer.saveReportDesign("NewDesign");
```

saveReportDocument

Syntax void Viewer.saveReportDocument(string filename, function callback)

Saves the current viewer content as a report document. The viewer must enable interactive viewing with enableIV( ) prior to saving a report design.

Parameters

filename
String. Sets the name of the saved file. The current file name is used if null. The file name must be a path relative to the viewer's repository.

callback
Function. Optional. The function to execute after the asynchronous call processing is done. The callback takes the current actuate.Viewer object as an input parameter.

Example To save the content of the viewer as the report document called NewDocument, use the following code:

```
myViewer.saveReportDocument("NewDocument");
```

setContentMarg

Syntax void Viewer.setContentMargin(string[ ] margin)

void Viewer.setContentMargin(int margin)

Sets the viewer content margin.

Parameter

margin
Array of strings or integer. Each member of the array is the margin for the top, left, right, and bottom internal margins for the viewer. An integer sets all margins to that value.

Example To set the internal margin of the viewer to a 10-pixel buffer, use the following code:

```
myViewer.setContentMargin(10);
```

setFocus

Syntax void setFocus(boolean focus)

Sets the focus for the viewer.

Parameter

focus
Boolean. The viewer's context menu is in focus when this parameter is set to true.

Example This example blurs the context menu for the viewer:

```
viewer.setFocus(false);
```

setHeight

Syntax void Viewer.setHeight(integer height)

Sets the viewer height.

Parameter **height**
Integer. The height in pixels.

Example To set the height of the viewer to 600 pixels, use the following code:

```
viewer.setHeight(600);
```

setHelpBase

Syntax void Viewer.setHelpBase(string helpBase)

Sets the URL of the help base. The help base is the base URL for the product help documentation.

Parameter **helpBase**
String. The URL where the help documentation is located.

Example This example sets the help base URL to http://developer.actuate.com resources/documentation/ihub/ihub31:

```
myViewer.setHelpBase("http://developer.actuate.com/resources/
   documentation/ihub/ihub31");
myViewer.submit( );
```

setParameters

Syntax void Viewer.setParameters(string[ ] params)

Sets parameters for executing report using literal string pairs.

Parameter **params**
Array of strings. Each string in the array is constructed of name:"value" pairs. Use a literal list, such as {param1:"value1", param2:"value2", … }.

Example To set the value of a parameter, city, to the value, New York, use the following object literal:

```
viewer.setParameters({ city:"New York"});
```

setParameterValues

Syntax void Viewer.setParameterValues(actuate.parameter.ParameterValue[ ] parameters)

Sets parameter values for executing a report using ParameterValue objects.

Parameter **parameters**
Array of actuate.parameter.ParameterValue objects. An array of this kind is returned by actuate.Parameter.downloadParameterValues( ) and is the recommended function for creating the parameters input.

Example To set the parameter values for a report to the parameters in the pvs array, use this code:

```
viewer.setParameterValues(pvs);
```

setReportletBookmark

Syntax void Viewer.setReportletBookmark(string bookmark)

Sets the bookmark for the Reportlet rendered.

Parameter **bookmark**
String. The bookmark ID used to render the Reportlet. Viewer requires a bookmark to render a Reportlet. Viewer does not support automatically generated generic bookmarks from a BIRT report.

Example To open the Top 5 Customers Reportlet of the Customer Dashboard, set the Reportlet bookmark by name and then call viewer.submit, as shown in the following example:

```
viewer.setReportName("/Public/BIRT and BIRT Studio Examples
   /Customer Dashboard.rptdocument");
viewer.setReportletBookmark("Top 5 Customers");
viewer.submit( );
```

setReportName

Syntax void Viewer.setReportName(string reportFile)

Sets the report file, either a report design or report document, to render in this Viewer.

Parameter **reportFile**
String. The report file path for a report design file or report document file. To set the version for the report, add a semicolon and the version number. For example, "/Public/BIRT and BIRT Studio Examples/Customer Dashboard.rptdesign;1" retrieves version 1 of Customer Dashboard.rptdesign.

Example To open the Top 5 Sales Performers report, set the report by name and then call submit( ), as shown in the following example:

```
viewer.setReportName("/Public/BIRT and BIRT Studio Examples/Top 5
   Sales Performers.rptdesign");
viewer.submit( );
```

setService

Syntax void Viewer.setService(string iPortalURL, actuate.RequestOptions requestOptions)

Sets the target service URL to which this Viewer links. When the service URL is not set, this Viewer links to the default service URL, which is set on the actuate object.

Parameters

iPortalURL
String. The target Actuate web application URL, either a Java Component or iHub Visualization Platform client.

requestOptions
actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, volume, or repository type. The URL can also include custom parameters.

Example This example sets the URL for the BIRT Java Component web application service:

```
myViewer.setService("http://127.0.0.1:8080/ajc",
   myRequestOptions);
```

setSize

Syntax void Viewer.setSize(integer width, integer height)

Resizes the viewer's width and height.

Parameters

width
Integer. The new width is specified in pixels.

height
Integer. The new height is specified in pixels.

Example To set the viewer's size to 300 pixels by 300 pixels, use code similar to the following:

```
myViewer.setSize(300, 300);
```

setSupportSVG

Syntax void Viewer.setSupportSVG(boolean usvgFlag)

Controls Scalable Vector Graphics support for the viewer.

Parameter **svgFlag**
Boolean. True enables Scalable Vector Graphic support.

Example To disable Scalable Vector Graphic support for the myViewer viewer, use code similar to the following:

```
myViewer.setSupportSVG(false);
```

setUIOptions

Syntax void Viewer.setUIOptions(actuate.viewer.UIOptions options)

Sets the UI options for the viewer using an actuate.viewer.UIOptions object.

Parameter **options**
actuate.viewer.UIOptions object. Enables or disables various controls and features.

Example To hide the toolbar for the viewer, use the following code:

```
uioptions.enableToolBar(false);
viewer.setUIOptions(uioptions);
viewer.submit( );
```

setWidth

Syntax void Viewer.setWidth(string width)

Sets the viewer width.

Parameter **width**
String.

Example To set the width of the viewer to 800 pixels, use the following code:

```
viewer.setWidth(800);
```

showDownloadReportDialog

Syntax void Viewer.showDownloadReportDialog( )

Displays the export report dialog window.

Example Use this code to display the report dialog window:

```
myViewer.showDownloadReportDialog( );
```

showDownloadResultSetDialog

Syntax void Viewer.showDownloadResultSetDialog( )

Displays the export data dialog window.

Example Use this code to display the result set download dialog window:

```
viewer.showDownloadResultSetDialog( );
```

showFacebookCommentPanel

Syntax void Viewer.showFacebookCommentPanel( )

Displays the Facebook comments panel.

Example Use this code to display the Facebook comments panel:

```
viewer.showFacebookCommentPanel( );
```

showParameterPanel

Syntax void Viewer.showParameterPanel( )

Displays the parameter panel.

Example Use this code to display the parameter panel:

```
viewer.showParameterPanel( );
```

showPrintDialog

Syntax void Viewer.showPrintDialog( )

Displays the print dialog window.

Example Use this code to display the print dialog window:

```
viewer.showPrintDialog( );
```

showTocPanel

Syntax void Viewer.showTocPanel( )

Displays the table of contents panel.

Example Use this code to display the table of contents panel:

```
viewer.showTocPanel( );
```

submit

Syntax void Viewer.submit(function callback, boolean rerun)

Updates and reloads the viewer after submitting requests for the viewer. The submit( ) function triggers an AJAX request for all asynchronous operations. When the server finishes the processing, it returns a response and the results are

rendered on the page in the viewer container. Calling submit( ) when a previous submit( ) is pending throws an exception.

Parameters

callback
Function. Optional. The function to execute after the asynchronous call processing is done.

rerun
Boolean. Optional. Indicates whether to re-run the report design when refreshing. Default to true.

Example

To open the Top 5 Sales Performers report, set the report by name and then call submit( ), as shown in the following example:

```
viewer.setReportName("/Public/BIRT and BIRT Studio Examples/Top 5
   Sales Performers.rptdesign");
viewer.submit( );
```

Class actuate.viewer.BrowserPanel

Description A container for a browser content panel in a viewer. This class defines the default scroll bars for a content panel.

Constructor

Syntax actuate.Viewer.BrowserPanel( )

Constructs a new BrowserPanel object for the parent viewer. The browser panel has vertical and horizontal scroll bars for navigation.

Class actuate.viewer.EventConstants

Description Defines the event constants supported by this API. Table 14-43 lists the viewer event constants.

Table 14-43 Actuate JavaScript API viewer event constants

Event	Description
ON_CONTENT_CHANGED	Calls a registered event handler when the report content is reloaded. The event handler must take the viewer instance that fired the event as an input argument.
ON_CONTENT_SELECTED	Calls a registered event handler when the relevant part of the report content is selected. Supported selected contents are: ■ Column ■ Table ■ Data ■ Label ■ Text When the content is selected, the corresponding object is passed into user's event handler function. For example, if the table area is selected in a viewer, actuate.Viewer.Table is passed into the event handler. The event handler must take the viewer instance that fired the event and an instance of actuate.viewer.SelectedContent as input arguments.
ON_DIALOG_OK	This event fires when the user clicks the OK button in a dialog. The event handler must take the viewer object that fired the event and a dialog.AdvancedFilterDialog object as input parameters.
ON_EXCEPTION	An exception event is broadcast when an error occurs. The event handler must take the viewer instance that fired the event and an instance of actuate.viewer.Exception as input arguments.
ON_SESSION_TIMEOUT	Calls a registered event handler when the session expires. The event handler must take the viewer instance that fired the event as an input argument.

Class actuate.viewer.PageContent

Description A container for the content of a report document file. actuate.Viewer.PageContent contains a comprehensive list of report elements, such as tables, charts, labels, and data items.

Constructor

The PageContent object is constructed by actuate.viewer.getCurrentPageContent( ).

Function summary

Table 14-44 lists actuate.viewer.PageContent functions.

Table 14-44 actuate.viewer.PageContent functions

Function	Description
getChartByBookmark( )	Returns a chart element specified by the given bookmark
getDataItemByBookmark( )	Returns a data element specified by the given bookmark
getFlashObjectByBookmark( )	Returns a Flash object specified by the given bookmark
getGadgetByBookmark( )	Returns a Flash gadget specified by the given bookmark
getLabelByBookmark( )	Returns a label element specified by the given bookmark
getTableByBookmark( )	Returns a table element specified by the given bookmark
getTextByBookmark( )	Returns a text element specified by the given bookmark
getViewerId( )	Returns the viewer ID

getChartByBookmark

Syntax actuate.report.Chart PageContent.getChartByBookmark(string bookmark)

Returns the chart element specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a chart element. When the bookmark value is not given, this function returns the first chart element found in the report content.

Returns actuate.report.Chart object.

Example This example retrieves the Chart object and changes the chart title:

```
this.onclick = function(event){
   var bviewer = this.getViewer( );
   var bpagecontents = bviewer.getCurrentPageContent( );
   var bchart = bpagecontents.getChartByBookmark("ChartBookmark");
   bchart.setChartTitle("Orders By Country (Classic Cars)");
   bchart.submit( );
}
```

getDataItemByBookmark

Syntax actuate.report.DataItem PageContent.getDataItemByBookmark(string bookmark)

Returns the data element specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a data element. When the bookmark value is not given, the first data element found in the report content is returned.

Returns actuate.report.DataItem object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the data element "myDataItem" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( ).
   getDataItemByBookmark("myDataItem");
```

getFlashObjectByBookmark

Syntax actuate.report.FlashObject PageContent.getFlashObjectByBookmark(string bookmark)

Returns the Flash object specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a Flash object. When the bookmark value is not given, the first data element found in the report content is returned.

Returns actuate.report.FlashObject object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the Flash object "myFlashObj" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( ).
   getFlashObjectByBookmark("myFlashObj");
```

getGadgetByBookmark

Syntax actuate.report.Gadget PageContent.getGadgetByBookmark(string bookmark)

Returns the gadget element specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a gadget element. When the bookmark value is not given, the first data element found in the report content is returned.

Returns actuate.report.Gadget object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the gadget "myGadget" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( ).
   getGadgetByBookmark("myGadget");
```

getLabelByBookmark

Syntax actuate.report.Label PageContent.getLabelByBookmark(string bookmark)

Returns the label element specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a label element. When the bookmark value is not given, the first label element found in the report content is returned.

Returns actuate.report.Label object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the label "LabelOne" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( ).
   getLabelByBookmark("LabelOne");
```

getTableByBookmark

Syntax actuate.report.Table PageContent.getTableByBookmark(string bookmark)

Returns the table element specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a table element. When the bookmark value is not given, the first table element found in the report content is returned.

Returns actuate.report.Table object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the table mytable on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( )
   .getTableByBookmark("mytable");
```

getTextByBookmark

Syntax actuate.report.TextItem PageContent.getTextByBookmark(string bookmark)

Returns the text element specified by the given bookmark.

Parameter **bookmark**
String. A bookmark to identify a text element. If the bookmark value is not given, the first text element found in the report content is returned.

Returns actuate.report.TextItem object.

Example Use this function to access the bookmarks for specific elements in the page content. For example, to access the text item "myTextItem" on the page loaded in the myViewer viewer object, use the following code:

```
var element = myViewer.getCurrentPageContent( ).
   getTextByBookmark("myTextItem");
```

getViewerId

Syntax string PageContent.getViewerId( )

Returns the viewer ID.

Returns String. The viewer ID.

Example This example displays the viewer ID in an alert box:

```
alert("The Viewer ID is " + myViewer.getViewerId( ));
```

Class actuate.viewer.ParameterValue

Description The ParameterValue class is a JavaScript version of the com.actuate.schemas.ParameterValue class.

Constructor

Syntax actuate.parameter.ParameterValue( )

Constructs a new ParameterValue object.

Function summary

Table 14-45 lists the actuate.viewer.ParameterValue functions.

Table 14-45 actuate.viewer.ParameterValue functions

Function	Description
getName( )	Returns the name value
getValue( )	Returns the value value
getValueIsNull( )	Returns the valueIsNull value
setColumnName( )	Sets the name value
setValue( )	Sets the value value
setValueIsNull( )	Sets the valueIsNull value

getName

Syntax string ParameterValue.getName( )

Returns the name value.

Returns String. The name value.

Example To store the name of a viewer.ParameterValue object in a variable called vPVname, use code similar to the following:

```
var vPVname = myParamValue.getName( );
```

getValue

Syntax object ParameterValue.getValue( )

Returns the value value.

Returns Object. The value value, a string or array of strings.

Example To store a ParameterValue's value in vPVvalue, use the following code:

```
var vPVvalue = myParamValue.getValue( );
```

getValueIsNull

Syntax boolean ParameterValue.getValueIsNull( )

Returns the valueIsNull value.

Returns Boolean. The valueIsNull value.

Example This example displays an alert with the valueIsNull of the ParameterValue object:

```
alert("Value is null: " + myParamValue.getValueIsNull( ));
```

setColumnName

Syntax void ParameterValue.setColumnName(string columnName)

Sets the columnName value.

Parameter **columnName**
String. The column name.

Example To set the column name to "Motorcycles", use code similar to the following:

```
myParamValue.setColumnName("Motorcycles");
```

setValue

Syntax void ParameterValue.setValue(object value)

Sets the value. A value can be a string or an array of strings.

Parameter **value**
Object. The value for this ParameterValue object, a string or an array of strings.

Example To set the value for a ParameterValue to myValues, use the following code:

```
var myValues = myParamValue.setValue(myValues);
```

setValueIsNull

Syntax void ParameterValue.setValueIsNull(boolean valueIsNull)

Sets the valueIsNull value.

Parameter **valueIsNull**
Boolean. The valueIsNull value.

Example To set a ParameterValue's setValueIsNull to true, use the following code:

```
myParamValue.setValueIsNull(true);
```

Class actuate.viewer.RenderOptions

Description The RenderOptions class specifies render options for the actuate.Viewer.downloadReport( ) function. Currently, the only supported option is multisheet.

Constructor

Syntax actuate.Viewer.RenderOptions( )

Constructs a new RenderOptions object for the parent viewer.

Function summary

Table 14-46 lists actuate.viewer.RenderOptions functions.

Table 14-46 actuate.viewer.RenderOptions functions

Function	Description
getOptions( )	Returns whether mouse scrolling is enabled
setOption( )	Returns whether mouse panning is enabled

getOptions

Syntax Object[ ] RenderOptions.getOptions( )

Returns the render options map.

Returns Array, arranged in string and object pairs corresponding to option names and option values.

Example This example displays an alert box with the options status of render options:

```
alert("Rendering Options: " + options.getOptions( ));
```

setOption

Syntax void RenderOptions.setOption(string option, boolean value)

Specifies a render option and its setting.

Parameters

option
String. The permitted value is actuate.viewer.RenderOptions.IS_MULTISHEET, which is used for xls format download only.

value
Boolean. Enabled value for IS_MULTISHEET. True indicates that the xls format file has multiple worksheets.

Example To disable multisheet for the options object, use code similar to the following:

```
options.setOption(actuate.viewer.RenderOptions.IS_MULTISHEET,
   false);
```

Class actuate.viewer.ScrollPanel

Description A container for a scrolling content panel in a viewer, which includes the scroll panel control, as shown in Figure 14-1.

Figure 14-1 Scroll panel control

A ScrollPanel object enhances the viewer with scroll controls, such as mouse wheel scrolling.

Constructor

Syntax actuate.Viewer.ScrollPanel( )

Constructs a new ScrollPanel object for the parent viewer enabled scroll controls.

Function summary

Table 14-47 lists actuate.viewer.ScrollPanel functions.

Table 14-47 actuate.viewer.ScrollPanel functions

Function	Description
getMouseScrollingEnabled( )	Returns whether mouse scrolling is enabled
getPanInOutEnabled( )	Returns whether mouse panning is enabled
getScrollControlEnabled( )	Returns whether scrolling is enabled
setMouseScrollingEnabled( )	Enables mouse scrolling
setPanInOutEnabled( )	Enables panning
setScrollControlEnabled( )	Enables scrolling

getMouseScrollingEnabled

Syntax boolean ScrollPanel.getMouseScrollingEnabled( )

Returns true when mouse scrolling is enabled.

Returns Boolean.

Example This example displays an alert with the mouse scrolling status of a scroll panel:

```
alert("Mouse scrolling enabled: " +
   sPanel.getMouseScrollingEnabled( ));
```

getPanInOutEnabled

Syntax boolean ScrollPanel.getPanInOutEnabled( )

Returns true when panning in and out is enabled.

Returns Boolean.

Example This example displays an alert with the panning in and out status of a scroll panel:

```
alert("Panning enabled: " + scrollPanel.getPanInOutEnabled( ));
```

getScrollControlEnabled

Syntax boolean ScrollPanel.getScrollControlEnabled( )

Returns true when scrolling is enabled.

Returns Boolean.

Example This example displays an alert box with the scrolling status of a scroll panel:

```
alert("Scrolling enabled: " + sPanel.getScrollControlEnabled( ));
```

setMouseScrollingEnabled

Syntax void ScrollPanel.setMouseScrollingEnabled(boolean enabled)

Enables mouse scrolling for this scroll panel.

Parameter **enabled**
Boolean.

Example To disable mouse scrolling for sPanel, use code similar to the following:

```
sPanel.setMouseScrollingEnabled(false);
```

setPanInOutEnabled

Syntax void ScrollPanel.setPanInOutEnabled(boolean enabled)

Enables panning in and out for this scroll panel.

Parameter **enabled**
Boolean.

Example To disable panning for the sPanel object, use code similar to the following:

```
sPanel.setPanInOutEnabled(false);
```

setScrollControlEnabled

Syntax void ScrollPanel.setScrollControlEnabled(boolean enabled)

Enables scrolling for this scroll panel.

Parameter **enabled**
Boolean.

Example To disable scrolling for sPanel, use code similar to the following:

```
sPanel.setScrollControlEnabled(false);
```

Class actuate.viewer.SelectedContent

Description A container for content selected in the viewer. SelectedContent provides an object to pass to a handler when the user-defined ON_CONTENT_SELECTED event occurs. This object contains an instance of the element selected in the viewer.

Constructor

The SelectedContent object is constructed when an ON_CONTENT_SELECTED event occurs.

Function summary

Table 14-48 lists actuate.viewer.SelectedContent functions.

Table 14-48 actuate.viewer.SelectedContent functions

Function	Description
getColumnIndex( )	Returns the currently selected table column index number
getSelectedElement( )	Returns a copy of the currently selected element

getColumnIndex

Syntax integer SelectedContent.getColumnIndex( )

Returns the numerical index for the currently selected column. Returns null when the user selects a non-table element.

Returns Integer.

Example To retrieve the index of a column selected, use the following code:

```
var index = selected.getColumnIndex( );
```

getSelectedElement

Syntax object SelectedContent.getSelectedElement( )

Returns an instance of the currently selected element. The instance can be one of the following objects:

- actuate.report.Chart
- actuate.report.DataItem
- actuate.report.Label

- actuate.report.Table
- actuate.report.TextItem

To determine the object type, use the Object.getType( ) function. The type strings for the above objects are "Chart", "Data", "Label", "Table", or "Text", respectively.

Returns Object. An instance of the currently selected element.

Example To retrieve and store a label bookmark if a selected element is a label, use the following code:

```
var selected = selected.getColumnIndex( );
if (selected.getType( ) == "Label"){
   var bmark = Object.getBookmark( );
}
```

Class actuate.viewer.UIConfig

Description The UIConfig class specifies feature availability for the viewer.

Constructor

Syntax void actuate.viewer.UIConfig( )

Generates a new UIConfig object to manage the content panel for the viewer. By default, the content panel is an actuate.viewer.ScrollPanel object with ScrollControl, PanInOut, and MouseScrolling enabled.

Function summary

Table 14-49 lists actuate.viewer.UIConfig functions.

Table 14-49 actuate.viewer.UIConfig functions

Function	Description
getContentPanel( )	Returns the content panel configuration
getShowToc( )	Gets the showToc flag
setContentPanel( )	Sets the content panel configuration
setShowToc( )	Sets the showToc flag

getContentPanel

Syntax object UIConfig.getContentPanel( )

Returns the content panel object.

Returns Object. Valid objects are actuate.viewer.BrowserPanel, actuate.viewer.ScrollPanel, and null. A null value indicates a content panel configured with the browser scroll bar enabled.

Example To retrieve and store the content panel from the viewer, use the following code:

```
var contentpanel = viewer.getUIConfig( ).getContentPanel( );
```

getShowToc

Syntax boolean UIConfig.getShowToc( )

Returns the showToc flag.

Returns Boolean.

Example To determine if the showToc flag is set to true, use the following code:

```
if (!viewer.getUIConfig( ).getShowToc( )){ ...}
```

setContentPanel

Syntax void UIConfig.setContentPanel(objectcontentPanel)

Sets the content panel for the viewer.

Parameter **contentPanel**
Object. Valid objects are actuate.viewer.BrowserPanel, actuate.viewer.ScrollPanel, and null. A null value sets a content panel configured with the browser scroll bar enabled.

Example To set the content panel to BrowserPanel if it is null, use the following code:

```
var contentpanel = viewer.getUIConfig( ).getContentPanel( );
if (contentpanel == null){
   var newconfig = viewer.getUIConfig( );
   newconfig.setContentPanel(new actuate.viewer.BrowserPanel( ));
   viewer.setUIConfig(newconfig);
}
```

setShowToc

Syntax void UIConfig.setShowToc(boolean showToc)

Sets the showToc flag.

Parameter **showToc**
Boolean.

Example To hide the Toc in the UI, use the following code:

```
var newconfig = viewer.getUIConfig( );
newconfig.setShowToc(false);
viewer.setUIConfig(newconfig);
```

Class actuate.viewer.UIOptions

Description The UIOptions class specifies feature availability for the viewer object.

Constructor

Syntax void actuate.viewer.UIOptions( )

Generates a new UIOptions object to manage the features of the viewer.

Function summary

Table 14-50 lists actuate.viewer.UIOptions functions.

Table 14-50 actuate.viewer.UIOptions functions

Function	Description
enableAdvancedSort( )	Enables the advanced sort feature
enableAggregation( )	Enables the aggregation feature
enableCalculatedColumn( )	Enables the calculated column feature
enableChartProperty( )	Enables the chart properties feature
enableChartSubType( )	Enables the chart subtype selection
enableCollapseExpand( )	Enables the collapse/expand feature
enableColumnEdit( )	Enables the column editing feature
enableColumnResize( )	Enables the column resizing feature
enableContentMargin( )	Enables the content margin feature
enableDataAnalyzer( )	Enables the Interactive Crosstab feature
enableDataExtraction( )	Enables the data extraction feature
enableEditReport( )	Enables the report editing feature
enableExportReport( )	Enables the export report feature
enableFilter( )	Enables the filter feature
enableFacebookComments( )	Enables the Facebook comments feature.
enableFlashGadgetType( )	Enables the Flash gadget type change feature
enableFormat( )	Enables the format editing feature
enableGroupEdit( )	Enables the group editing feature
enableHideShowItems( )	Enables the hide/show item feature
enableHighlight( )	Enables the highlight feature

(continues)

Table 14-50 actuate.viewer.UIOptions functions (continued)

Function	Description
enableHoverHighlight( )	Enables the hover highlight feature
enableLaunchViewer( )	Enables the launch viewer feature
enableLinkToThisPage( )	Enables the "link to this page" feature
enableMainMenu( )	Enables the main menu feature
enableMoveColumn( )	Enables column moving
enablePageBreak( )	Enables the page break editing feature
enablePageNavigation( )	Enables the page navigation feature
enableParameterPage( )	Enables the parameter page feature
enablePrint( )	Enables the print feature
enableReorderColumns( )	Enables the column reordering
enableRowResize( )	Enables row resizing
enableSaveDesign( )	Enables the report design save feature
enableSaveDocument( )	Enables the report document save feature
enableShowToolTip( )	Enables the show tooltip feature
enableSort( )	Enables the sort feature
enableSuppressDuplicate( )	Enables the duplication suppression feature
enableSwitchView( )	Enables the switch view feature
enableTextEdit( )	Enables the text editing feature
enableTOC( )	Enables the table of contents feature
enableToolBar( )	Enables the toolbar feature
enableToolbarContextMenu( )	Enables the toolbar context menu feature
enableToolbarHelp( )	Enables the toolbar help feature
enableTopBottomNFilter( )	Enables the top N and bottom N filter feature
enableUndoRedo( )	Enables the undo and redo feature
getFeatureMap( )	Returns a list of enabled and disabled features

enableAdvancedSort

Syntax void UIOptions.enableAdvancedSort(boolean enabled)

Enables or disables the advanced sort feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the advanced sort feature, use code similar to the following:

```
viewerOpts.enableAdvancedSort(false);
```

enableAggregation

Syntax void UIOptions.enableAggregation(boolean enabled)

Enables or disables the aggregation feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the aggregation feature, use code similar to the following:

```
viewerOpts.enableAggregation(false);
```

enableCalculatedColumn

Syntax void UIOptions.enableCalculatedColumn(boolean enabled)

Enables or disables the calculated column feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the calculated column feature, use code similar to the following:

```
viewerOpts.enableCalculatedColumn(false);
```

enableChartProperty

Syntax void UIOptions.enableChartProperty(boolean enabled)

Enables or disables the chart properties feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the chart properties feature, use code similar to the following:

```
viewerOpts.enableChartProperty(false);
```

enableChartSubType

Syntax void UIOptions.enableChartSubType(boolean enabled)

Enables or disables the chart subtype selection feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the chart subtype selection feature, use code similar to the following:

```
viewerOpts.enableChartSubType(false);
```

enableCollapseExpand

Syntax void UIOptions.enableCollapseExpand(boolean enabled)

Enables or disables the collapse/expand feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the collapse/expand feature, use code similar to the following:

```
viewerOpts.enableCollapseExpand(false);
```

enableColumnEdit

Syntax void UIOptions.enableColumnEdit(boolean enabled)

Enables or disables the column editing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the column editing feature, use code similar to the following:

```
viewerOpts.enableColumnEdit(false);
```

enableColumnResize

Syntax void UIOptions.enableColumnResize(boolean enabled)

Enables or disables the column resizing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the column resizing feature, use code similar to the following:

```
viewerOpts.enableColumnResize(false);
```

enableContentMargin

Syntax void UIOptions.enableContentMargin(boolean enabled)

Enables or disables the content margin feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the content margin feature, use code similar to the following:

```
viewerOpts.enableContentMargin(false);
```

enableDataAnalyzer

Syntax void UIOptions.enableDataAnalyzer(boolean enabled)

Enables or disables the Interactive Crosstab feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the Interactive Crosstab feature, use code similar to the following:

```
viewerOpts.enableDataAnalyzer(false);
```

enableDataExtraction

Syntax void UIOptions.enableDataExtraction(boolean enabled)

Enables or disables the data extraction feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the data extraction feature, use code similar to the following:

```
viewerOpts.enableDataExtraction(false);
```

enableEditReport

Syntax void UIOptions.enableEditReport(boolean enabled)

Enables or disables the report editing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the report editing feature, use code similar to the following:

```
viewerOpts.enableEditReport(false);
```

enableExportReport

Syntax void UIOptions.enableExportReport(boolean enabled)

Enables or disables the export report feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the export report feature, use code similar to the following:

```
viewerOpts.enableExportReport(false);
```

enableFilter

Syntax void UIOptions.enableFilter(boolean enabled)

Enables or disables the filter feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the filter feature, use code similar to the following:

```
viewerOpts.enableFilter(false);
```

enableFacebookComments

Syntax void UIOptions.enableFacebookComments(boolean enabled)

Enables or disables the Facebook comments feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the Facebook comments feature, use code similar to the following:

```
viewerOpts.enableFacebookComments(false);
```

enableFlashGadgetType

Syntax void UIOptions.enableFlashGadgetType(boolean enabled)

Enables or disables the Flash gadget type change control.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the Flash gadget type change control, use code similar to the following:

```
viewerOpts.enableFlashGadgetType(false);
```

enableFormat

Syntax void UIOptions.enableFormat(boolean enabled)

Enables or disables the format editing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the format editing feature, use code similar to the following:

```
viewerOpts.enableFormat(false);
```

enableGroupEdit

Syntax void UIOptions.enableGroupEdit(boolean enabled)

Enables or disables the group editing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the group editing feature, use code similar to the following:

```
viewerOpts.enableGroupEdit(false);
```

enableHideShowItems

Syntax void UIOptions.enableHideShowItems(boolean enabled)

Enables or disables the hide/show item feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the hide/show feature, use code similar to the following:

```
viewerOpts.enableHideShowItems(false);
```

enableHighlight

Syntax void UIOptions.enableHighlight(boolean enabled)

Enables or disables the highlight feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the highlight feature, use code similar to the following:

```
viewerOpts.enableHighlight(false);
```

enableHoverHighlight

Syntax void UIOptions.enableHoverHighlight(boolean enabled)

Enables or disables the hover highlight feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the hover highlight feature, use code similar to the following:

```
viewerOpts.enableHoverHighlight(false);
```

enableLaunchViewer

Syntax void UIOptions.enableLaunchViewer(boolean enabled)

Enables or disables the launch viewer feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the launch viewer feature, use code similar to the following:

```
viewerOpts.enableLaunchViewer(false);
```

enableLinkToThisPage

Syntax void UIOptions.enableLinkToThisPage(boolean enabled)

Enables or disables the "link to this page" feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the "link to this page" feature, use code similar to the following:

```
viewerOpts.enableLinkToThisPage(false);
```

enableMainMenu

Syntax void UIOptions.enableMainMenu(boolean enabled)

Enables or disables the main menu feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the main menu feature, use code similar to the following:

```
viewerOpts.enableMainMenu(false);
```

enableMoveColumn

Syntax void UIOptions.enableMoveColumn(boolean enabled)

Enables or disables the option to move columns.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the option to move columns, use code similar to the following:

```
viewerOpts.enableMoveColumn(false);
```

enablePageBreak

Syntax void UIOptions.enablePageBreak(boolean enabled)

Enables or disables the page break editing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the page break editing feature, use code similar to the following:

```
viewerOpts.enablePageBreak(false);
```

enablePageNavigation

Syntax void UIOptions.enablePageNavigation(boolean enabled)

Enables or disables the page navigation feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the page navigation feature, use code similar to the following:

```
viewerOpts.enablePageNavigation(false);
```

enableParameterPage

Syntax void UIOptions.enableParameterPage(boolean enabled)

Enables or disables the parameter page feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the parameter page feature, use code similar to the following:

```
viewerOpts.enableParameterPage(false);
```

enablePrint

Syntax void UIOptions.enablePrint(boolean enabled)

Enables or disables the print feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the print feature, use code similar to the following:

```
viewerOpts.enablePrint(false);
```

enableReorderColumns

Syntax void UIOptions.enableReorderColumns(boolean enabled)

Enables or disables the column reordering feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the column reordering feature, use code similar to the following:

```
viewerOpts.enableReorderColumns(false);
```

enableRowResize

Syntax void UIOptions.enableRowResize(boolean enabled)

Enables or disables row resizing.

Parameter **enabled**
Boolean. True enables this option.

Example To disable row resizing, use code similar to the following:

```
viewerOpts.enableRowResize(false);
```

enableSaveDesign

Syntax void UIOptions.enableSaveDesign(boolean enabled)

Enables or disables the report design save feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the report design save feature, use code similar to the following:

```
viewerOpts.enableSaveDesign(false);
```

enableSaveDocument

Syntax void UIOptions.enableSaveDocument(boolean enabled)

Enables or disables the report document save feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the report document save feature, use code similar to the following:

```
viewerOpts.enableSaveDocument(false);
```

enableShowToolTip

Syntax void UIOptions.enableShowToolTip(boolean enabled)

Enables or disables the showing of tooltips.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the showing of tooltips, use code similar to the following:

```
viewerOpts.enableShowToolTip(false);
```

enableSort

Syntax void UIOptions.enableSort(boolean enabled)

Enables or disables the sort feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the sort feature, use code similar to the following:

```
viewerOpts.enableSort(false);
```

enableSuppressDuplicate

Syntax void UIOptions.enableSuppressDuplicate(boolean enabled)

Enables or disables the duplication suppression feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the duplication suppression feature, use code similar to the following:

```
viewerOpts.enableSuppressDuplicate(false);
```

enableSwitchView

Syntax void UIOptions.enableSwitchView(boolean enabled)

Enables or disables the switch view feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the switch view feature, use code similar to the following:

```
viewerOpts.enableSwitchView(false);
```

enableTextEdit

Syntax void UIOptions.enableTextEdit(boolean enabled)

Enables or disables the text editing feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the text editing feature, use code similar to the following:

```
viewerOpts.enableTextEdit(false);
```

enableTOC

Syntax void UIOptions.enableTOC(boolean enabled)

Enables or disables the table of contents feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the table of contents feature, use code similar to the following:

```
viewerOpts.enableTOC(false);
```

enableToolBar

Syntax void UIOptions.enableToolBar(boolean enabled)

Enables or disables the toolbar feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the toolbar feature, use code similar to the following:

```
viewerOpts.enableToolBar(false);
```

Example This code initializes a new viewer display, using enableToolBar(false) to disable the toolbar:

```
function initDisplay( ){
   var uioptions = new actuate.viewer.UIOptions( );
   viewer = new actuate.Viewer("viewerpane");

   var viewerwidth = 800;
   var viewerheight = 600;
```

```
   viewer.setWidth(viewerwidth);
   viewer.setHeight(viewerheight);

   uioptions.enableToolBar(false);

   viewer.setUIOptions(uioptions);

   document.getElementById("display").disabled = false;
}
```

enableToolbarContextMenu

Syntax void UIOptions.enableToolbarContextMenu(boolean enabled)

Enables or disables the context menu feature.

Parameter **enabled**
Boolean. True enables this option.

Example This code initializes a new viewer display, using enableToolbarHelp(true) to enable the toolbar help feature:

```
function initDisplay( ){
   var uioptions = new actuate.viewer.UIOptions( );
   viewer = new actuate.Viewer("viewerpane");

   var viewerwidth = 800;
   var viewerheight = 600;

   viewer.setWidth(viewerwidth);
   viewer.setHeight(viewerheight);

   uioptions.enableToolBar(true);
   uioptions.enableToolbarHelp(true);

   viewer.setUIOptions(uioptions);

   document.getElementById("display").disabled = false;
}
```

enableToolbarHelp

Syntax void UIOptions.enableToolbarHelp(boolean enabled)

Enables or disables the toolbar help feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the toolbar help feature, use code similar to the following:

```
viewerOpts.enableToolbarHelp(false);
```

enableTopBottomNFilter

Syntax void UIOptions.enableTopBottomNFilter(boolean enabled)

Enables or disables the top N and bottom N filter feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the top N and bottom N filter feature, use code similar to the following:

```
viewerOpts.enableTopBottomNFilter(false);
```

enableUndoRedo

Syntax void UIOptions.enableUndoRedo(boolean enabled)

Enables or disables the undo and redo feature.

Parameter **enabled**
Boolean. True enables this option.

Example To disable the undo and redo feature, use code similar to the following:

```
viewerOpts.enableUndoRedo(false);
```

getFeatureMap

Syntax object UIOptions.getFeatureMap( )

Returns the features and their Boolean values as an associative array. This function makes the name of each feature an object property and sets the value of that property to the associated enabled Boolean value.

Returns Object.

Class actuate.viewer.ViewerException

Description A container for an exception. ViewerException provides an object to pass to a handler when the user-defined ON_EXCEPTION event occurs. It contains a reference to the element that generated the exception.

Constructor

The ViewerException object is constructed when an ON_EXCEPTION event occurs. The exceptions are divided into three types, which determine the contents of the Exception object. These types are:

- ERR_CLIENT: Exception type for a client-side error
- ERR_SERVER: Exception type for a server error
- ERR_USAGE: Exception type for a JSAPI usage error

Function summary

Table 14-51 lists actuate.viewer.ViewerException functions.

Table 14-51 actuate.viewer.ViewerException functions

Function	Description
getElement( )	Returns the element for which the exception occurred
getErrorMessage( )	Returns the exception message

getElement

Syntax object ViewerException.getElement( )

Returns an instance of the element that caused the exception, if applicable. The instance can be an object of one of following types:

- actuate.report.Chart
- actuate.report.DataItem
- actuate.report.Label
- actuate.report.Table
- actuate.report.TextItem

To determine the object type, use the Object.getType( ) function. The type strings for the above objects are "Chart", "Data", "Label", "Table", or "Text", respectively.

Returns Object. An instance of the element that generated the exception.

Example This example displays the type of element that generated the exception in an alert box:

```
alert("Exception in " + vException.getElement.getType( ));
```

getErrorMessage

Syntax string ViewerException.getErrorMessage( )

Returns the error message for the exception.

Returns String. A server error message.

Example This example displays the server error code in an alert box:

```
alert("Server error message: " + vException.getErrorMessage( ));
```

Chapter

14

BIRT Interactive Crosstabs API classes

This chapter contains:

- About the BIRT Interactive Crosstabs JavaScript API
- Interactive Crosstabs API reference
- Interactive Crosstabs JavaScript classes quick reference

About the BIRT Interactive Crosstabs JavaScript API

The Interactive Crosstabs portion of the Actuate JavaScript API is a set of JavaScript classes that modify, analyze, and display data within cross tab elements. These classes are available to users of iHub Visualization Platform client and BIRT Java Components. The Actuate JavaScript API functions that are described in this chapter invoke and control the Interactive Crosstabs viewer and elements that are associated with the viewer. The Interactive Crosstabs JavaScript can be placed within a web page or any other location where the Actuate JavaScript API interfaces with a cross tab.

The actuate.xtabAnalyzer class represents the Interactive Crosstabs viewer that contains cross tab information. Load the analyzer with actuate.load( ).

```
actuate.load("xtabAnalyzer");
```

Load support for dialog boxes from the Actuate JavaScript API with actuate.load( ), as shown in the following code:

```
actuate.load("dialog");
```

Load the XTabAnalyzer and dialog components to prepare the actuate.XTabAnalyzer component for use within a web page. Call actuate.XTabAnalyzer functions to create and prepare an analytics cross tab. Call the XTabAnalyzer's submit( ) function to display an existing cross tab in a specified HTML element on a web page.

Use the following JavaScript code to create an instance of a Interactive Crosstabs viewer:

```
var ctViewer = new actuate.XTabAnalyzer("cTab");
```

In this example, cTab is the name value for the element that holds the cross tab content. The web page body must contain a element with an ID value of cTab, as shown in the following code:

```
<DIV ID="cTab"></DIV>
```

When no element with the correct ID value exists in the web page body, the Interactive Crosstabs viewer launches in a pop-up window.

To load a cross tab or a data cube, use setReportName( ).

```
ctViewer.setReportName("/Public/BIRT and BIRT Studio Examples
   /Crosstab Sample Revenue.rptdocument");
```

The example code loads a report document that consists of a single data cube and cross tab. The report document can be loaded into the Interactive Crosstabs viewer directly.

To access a cross tab element that is part of a larger report, use the cross tab element's bookmark after setting the report name. A bookmark is set in a report

designer or by an external function. Retrieve a cross tab element with actuate.xtabanalyzer.PageContent.getCrosstabByBookmark( ). For example, the following code retrieves a cross tab with the bookmark SampleRevenue:

```
var content = ctViewer.getCurrentPageContent( );
var crosstab = content.getCrosstabByBookmark("SampleRevenue");
```

The code in this example retrieves the current page content and the cross tab element within that page, returning an actuate.xtabanalyzer.Crosstab object. This cross tab object supports the modification of the cross tab with the functions in the Xtabanalyzer subclasses.

To set the bookmark for a cross tab element, create a bookmark for the element within BIRT Designer Professional or call setXTabBookmark( ), as shown in the following code:

```
ctViewer.setXTabBookmark("SampleRevenue");
```

This example code assigns the bookmark SampleRevenue to the cross tab.

The XTabAnalyzer.submit( ) function triggers an AJAX request to display the report with all the asynchronous operations that previous viewer functions have prepared. Call submit( ) as shown in the following code:

```
ctViewer.submit( );
```

Upon executing submit( ), the Actuate web application returns the report with the cross tab in the assigned element.

Interactive Crosstabs API reference

This section provides an alphabetic listing of the Interactive Crosstabs API classes.

The examples in this section consist of JavaScript functions usable by a typical web page. These examples use a sample report document called reportfile.rptdocument. The sample report document contains a cross tab that has been bookmarked within BIRT Designer Professional with the value of Sample Revenue. Use any equivalent file of that design. Place the Interactive Crosstabs viewer in the acviewer container. The acviewer container is a tag in the HTML page with the following code:

```
<DIV ID="acviewer" STYLE="border-width: 1px; border-style:
   solid;display:none;"></DIV>
```

The JavaScript setup for the examples includes the initialization of the Data Analytics module and the setup of variables for use by the examples, as shown in the following code:

```
<HTML>
...
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!-- Load the xtabAnalyzer viewer component-->
actuate.load("xtabAnalyzer");
actuate.load("dialog");
actuate.initialize("../../",null,null,null,run)

var content;
var crosstab;
var viewer;
var container;
function run( ){
   container = document.getElementById("acviewer");
   viewer = new actuate.XTabAnalyzer(container);
   viewer.setReportName("reportfile.rptdocument");
   viewer.setXTabBookmark("Sample Revenue");
   viewer.submit( );
   content = viewer.getCurrentPageContent( );
   crosstab = content.getCrosstabByBookmark( );
}

<!-- JavaScript application functions -->

</SCRIPT>

<!-- Other HTML code -->
...
</HTML>
```

The viewer variable points to the XTabAnalyzer object. The content variable points to the data within the web page. The crosstab variable points to the cross tab. These variables are used throughout the examples as needed.

Place example functions in the area marked "JavaScript application functions". The section marked "Other HTML code" contains and other tags necessary for the web page.

Call the examples as any other JavaScript function. For example, the following HTML code creates a button with the label "Job 1" on it. When a user clicks that button, the page runs the JavaScript function Job1.

```
<INPUT TYPE="button" CLASS="btn" VALUE="Job 1" ONCLICK="Job1( );">
```

Interactive Crosstabs JavaScript classes quick reference

Table 15-1 lists the Interactive Crosstabs JavaScript classes.

Table 15-1 Actuate Interactive Crosstabs JavaScript classes

JavaScript class	Description
actuate.XTabAnalyzer	A Interactive Crosstabs viewer component that can be embedded in an HTML page
actuate.xtabanalyzer.Crosstab	A cross tab element
actuate.xtabanalyzer.Dimension	A data dimension
actuate.xtabanalyzer.Driller	A helper class for drilling down through cross tab data
actuate.xtabanalyzer.EventConstants	Global constants for Interactive Crosstabs events class
actuate.xtabanalyzer.Exception	Exception object sent to calling function
actuate.xtabanalyzer.Filter	Filter conditions to filter data
actuate.xtabanalyzer.GrandTotal	A cross tab grand total
actuate.xtabanalyzer.Level	A cross tab level
actuate.xtabanalyzer.LevelAttribute	An attribute for a level
actuate.xtabanalyzer.Measure	A data measure
actuate.xtabanalyzer.MemberValue	Data as a member value
actuate.xtabanalyzer.Options	Options for the cross tab
actuate.xtabanalyzer.PageContent	The content shown in the Interactive Crosstabs viewer
actuate.xtabanalyzer.ParameterValue	A cross tab parameter value
actuate.xtabanalyzer.Sorter	Conditions for sorting data
actuate.xtabanalyzer.SubTotal	A cross tab subtotal
actuate.xtabanalyzer.Total	A cross tab total
actuate.xtabanalyzer.UIOptions	Enables UI elements of the Interactive Crosstabs

Class actuate.XTabAnalyzer

Description The XTabAnalyzer class represents a Interactive Crosstabs viewer, used to view and operate a crosstab.

Constructor

Syntax actuate.XTabAnalyzer( )

Constructs a new Interactive Crosstabs object.

actuate.XTabAnalyzer(object xtabContainer, actuate.xtabanalyzer.UIOptions uiOptions)

actuate.XTabAnalyzer(string xtabContainerId, actuate.xtabanalyzer.UIOptions uiOptions)

Constructs a new Interactive Crosstabs object in the specified container.

Parameters

xtabContainer
Object. A document object referencing the HTML element that contains the xTabAnalyzer viewer.

xtabContainerId
String. The value of the ID parameter for an HTML element to hold the xTabAnalyzer viewer. For example, with 'containerName' as the xtabContainer parameter, a <DIV ID='containerName' /> tag on the page displays the viewer at the location of the element.

uiOptions
actuate.xtabanalyzer.UIOptions object. Optional. UIOptions references display options for the viewer.

Function summary

Table 15-2 lists actuate.XTabAnalyzer functions.

Table 15-2 actuate.XTabAnalyzer functions

Function	Description
commit( )	Commits all changes to the report design
forceSoftRestart( )	Forces the viewer to restart
getCurrentPageContent( )	Returns the Current Page Content object
getCurrentPageNum( )	Returns the current page number
getGadgetId	Returns the gadget ID of the shown cross tab
getHeight( )	Returns the viewer height

Table 15-2 actuate.XTabAnalyzer functions (continued)

Function	Description
getLeft( )	Returns the viewer left margin
getParameterValues( )	Returns the parameter values.
getPosition( )	Returns the CSS position attribute value
getTop( )	Returns the viewer top margin
getTop( )	Returns the total page count
getTotalPageCount( )	Returns the actuate.xtabanalyzer.UIOptions object assigned to this viewer
getUIOptions( )	Gets a viewer within a container
getViewer( )	Returns the viewer width
getWidth( )	Returns the bookmark of the cross tab displayed in the viewer
getXTabBookmark( )	Returns the instance ID of the cross tab displayed in the viewer
isActive( )	Checks if current viewer pop-up is active
isDashboard( )	Checks if the current viewer pop-up is a dashboard
isInteractive( )	Checks if the current viewer is interactive
registerEventHandler( )	Registers an event handler
removeEventHandler( )	Removes an event handler
reset( )	Resets the viewer object
resizeTo( )	Resizes the viewer
rollback( )	Rolls back all changes in the viewer and refreshes its content
setGadgetId	Sets the gadget id of the cross tab
setHeight( )	Sets the viewer height
setIVMode( )	Sets whether the viewer is in IV mode
setLeft( )	Sets the viewer left margin
setOnClosed( )	Sets callback when the pop-up window is closed
setPageNum( )	Sets the page number
setPosition( )	Sets the CSS position attribute
setParameterValues( )	Sets the parameter values

(continues)

Table 15-2 actuate.XTabAnalyzer functions (continued)

Function	Description
setReportletDocumentMode( )	Sets a Reportlet to document mode
setReportName( )	Sets the report to load into the interactive crosstab.
setService( )	Sets the Actuate web application service and request options
setSupportSVG( )	Sets whether or not the client browser supports SVG
setTop( )	Sets the top margin
setUIOptions( )	Sets the user interface options for the viewer
setWidth( )	Sets the viewer width
setXTabBookmark( )	Sets a bookmark for the cross tab
setXTabIid( )	Sets the instance ID of the cross tab
submit( )	Submits asynchronous operations and renders the requested components

commit

Syntax void XTabAnalyzer.commit(function callback)

Commits all design changes to a generated document as a single operation. If ivMode is not set to true, call setIVMode( ) to set the value of ivMode to true before calling commit( ).

Parameter **callback**
Function. The callback function called after commit finishes.

Example This example opens a design with a cross tab and pivots the cross tab:

```
function pivot( ){
// make a change to the cross tab.
   crosstab.pivot( );
   crosstab.submit( );
   viewer.commit( );
}
```

forceSoftRestart

Syntax void XTabAnalyzer.forceSoftRestart( )

Forces the viewer to restart.

Example This example restarts the viewer:

```
this.onclick = function(event){
   forceSoftRestart( );
}
```

getCurrentPageContent

Syntax actuate.xtabanalyzer.PageContent XTabAnalyzer.getCurrentPageContent( )

Returns the Current Page Content object.

Returns actuate.xtabanalyzer.PageContent object. Content from the current page.

Example This example retrieves the cross tab from the current page:

```
function getCrosstab(analyzerViewer){
   var content = analyzerViewer.getCurrentPageContent( );
   return content.getCrosstabByBookmark( );
}
```

getCurrentPageNum

Syntax integer XTabAnalyzer.getCurrentPageNum( )

Returns the current page number.

Returns Integer. The current page number.

Example This example retrieves the page number:

```
function retrievePageNum( ){
   return analyzerViewer.getCurrentPageNum( );
}
```

getGadgetId

Syntax string XTabAnalyzer.getGadgetId( )

Returns the gadget ID of the shown cross tab. This function is used for dashboard integration.

Returns String. A gadget ID.

Example This example retrieves the gadget ID:

```
function retrieveGadgetID( ){
   return analyzerViewer.getGadgetId( );
}
```

getHeight

Syntax integer XTabAnalyzer.getHeight( )

Returns the height of the viewer.

Returns Integer. The height in pixels.

Example This example retrieves the current height of the viewer and doubles the height if the current height is lower than 630 pixels:

```
function doubleHeight( ){
   var height = viewer.getHeight( );
   if (height < 630){
      viewer.setHeight(height * 2);
      viewer.submit( );
   }
}
```

getLeft

Syntax integer XTabAnalyzer.getLeft( )

Returns the left margin of the viewer.

Returns Integer. The left margin in pixels.

Example This example retrieves the position of the viewer's left margin and moves the margin 20 pixels to the right if the left margin is fewer than 45 pixels from the left edge of the screen:

```
function moveLeftMargin( ){
   var left = viewer.getLeft( );
   if (left < 45){
      viewer.setLeft(left + 20);
      viewer.submit( );
   }
}
```

getParameterValues

Syntax actuate.xtabanalyzer.ParameterValue[ ] XTabAnalyzer.getParameterValues( )

Returns the parameter values.

Returns actuate.xtabanalyzer.ParameterValue[ ] or actuate.parameter.ParameterValue[ ]. An array of parameter values.

getPosition

Syntax string XTabAnalyzer.getPosition( )

Returns the CSS position attribute for the viewer.

Returns String. The CSS position attribute.

Example This example changes the CSS positioning type from relative to absolute:

```
function changePosition( ){
   if (viewer.getPosition( ) == 'relative'){
      viewer.setPosition('absolute');
      viewer.submit( );
   }
}
```

getTop

Syntax integer XTabAnalyzer.getTop( )

Returns the top margin of the viewer.

Returns Integer. The top margin in pixels.

Example This example retrieves the value for the viewer's top margin and moves the margin 20 pixels down the screen if the margin was fewer than 45 pixels from the top of the screen:

```
function moveTopMargin( ){
   var top = viewer.getTop( );
   if (top < 45){
      viewer.setTop(top + 20);
      viewer.submit( );
   }
}
```

getTotalPageCount

Syntax integer XTabAnalyzer.getTotalPageCount( )

Returns the total page count.

Returns Integer. The total number of pages.

Example This example displays an alert with the total page count from viewer:

```
alert("Total pages: " + viewer.getTotalPageCount( ));
```

getUIOptions

Syntax actuate.xtabanalyzer.UIOptions getUIOptions( )

Returns the user interface options object for the cross tab analyzer. The UIOptions object specifies what features are used within the viewer.

Returns actuate.xtabanalyzer.UIOptions object. Interactive Crosstabs user interface options.

Example This example retrieves the user interface options and sets one of the UIOptions values:

```
function resetUIOptions( ){
   var options = viewer.getUIOptions( );
   options.enableToolbar(false);
   viewer.setUIOptions(options);
}
```

getViewer

Syntax static XTabAnalyzer.getViewer(HTMLElement container)

Returns a viewer by container. To retrieve the viewer for the current object, do not specify a container. This function is useful to retrieve the instance ID for a specific viewer when there are multiple viewers on a page.

Parameter **container**
HTMLElement. The container instance ID from which to retrieve the viewer.

Returns XTabAnalyzer object. The Interactive Crosstabs viewer.

Example This example retrieves the viewer:

```
function retrieveViewer( ){
   return viewer.getViewer( );
}
```

getWidth

Syntax string XTabAnalyzer.getWidth( )

Returns the width value of the viewer.

Returns String. The width in pixels.

Example This example retrieves the width of the viewer, then alters it based on the size:

```
function doubleWidth( ){
   var width = viewer.getWidth( );
   if (width < 630){
      viewer.setWidth(width * 2);
      viewer.submit( );
   }
}
```

getXTabBookmark

Syntax string XTabAnalyzer.getXTabBookmark( )

Returns the bookmark name for the cross tab set to render in the viewer.

Returns String. The bookmark for a cross tab.

Example This example retrieves the bookmark that the cross tab is associated with, changes the bookmark, and resets the bookmark. This functionality supports the use of multiple cross tab elements within a single design.

```
function changeBookmark( ){
   var oldBookMark = viewer.getXTabBookmark( );
   viewer.setXTabBookmark("crosstab2");
   viewer.submit( );
}
```

getXTablid

Syntax string XTabAnalyzer.getXTablid( )

Returns the current instance ID of the interactive crosstab. This function is useful in integration with Interactive Viewer and supports the ability of Interactive Viewer to obtain and use the interactive crosstab instance ID.

Returns String. A interactive crosstab instance ID.

Example This example retrieves the interactive crosstab instance ID:

```
function retrieveXTablid( myviewer ){
   return myviewer.getXTablid( );
}
```

isActive

Syntax boolean XTabAnalyzer.isActive( )

Returns true when a pop-up containing an interactive crosstab is active and false in all other cases.

Returns Boolean. True indicates an active interactive crosstab pop-up window.

Example This example checks if a viewer exists by checking two conditions: the viewer variable exists, or isActive( ) returns true. When both conditions fail, the example code creates a new viewer object within a container:

```
function checkViewer( ){
   if(!viewer || !viewer.isActive( )){
      viewer = new actuate.XTabAnalyzer(container);
   }
}
```

isDashboard

Syntax boolean XTabAnalyzer.isDashboard( )

Returns true when dashboard mode is active and false in all other cases.

Returns Boolean. True indicates dashboard mode.

isInteractive

Syntax boolean XTabAnalyzer.isInteractive( )

Returns whether this Interactive Crosstabs Viewer is in Interactive mode.

Returns Boolean. True indicates dashboard mode.

Example This example displays whether myDataAnalyzer is interactive:

```
alert("Interactive mode: " + myDataAnalyzer.isInteractive( ));
```

registerEventHandler

Syntax void XTabAnalyzer.registerEventHandler(string viewerEvent, function handler)

Registers an event handler for the specified event. This function throws actuate.xtabanalyzer.Exception when invalid arguments are passed.

Parameters

viewerEvent
String. Specifies the event that triggers the handler call. For a list of supported events, see actuate.xtabanalyzer.EventConstants.

handler
Function. Called when the event occurs.

Example This example changes an event handler from one function to another:

```
function changeEventHandler( event ){
   viewer.removeEventHandler(actuate.xtabanalyzer.EventConstants.
                             ON_CONTENT_CHANGED,
                             oldChangedHandler);
   viewer.registerEventHandler(actuate.xtabanalyzer.
                               EventConstants.ON_CONTENT_CHANGED,
                               newChangedHandler);
}
```

removeEventHandler

Syntax void XTabAnalyzer.removeEventHandler(string viewerEvent, function handler)

Removes an event handler from the specified event. This function throws actuate.xtabanalyzer.Exception when invalid arguments are passed.

Parameters

viewerEvent
String. Specifies the event from which to remove the event handler. For a list of supported events see actuate.xtabanalyzer.EventConstants.

handler
Function. The function to deregister from the event.

Example This example changes an event handler from one function to another:

```
function changeEventHandler( event ){
   viewer.removeEventHandler(actuate.xtabanalyzer.EventConstants.
                             ON_CONTENT_CHANGED,
                             oldChangedHandler);
   viewer.registerEventHandler(actuate.xtabanalyzer.
                                EventConstants.ON_CONTENT_CHANGED,
                                newChangedHandler);
}
```

reset

Syntax void XTabAnalyzer.reset( )

Resets the viewer to its initial state.

Example This example resets the viewer. All changes to the viewer made prior to this call are lost:

```
function resetViewer( ){
   viewer.reset( );
}
```

resizeTo

Syntax void XTabAnalyzer.resizeTo(integer width, integer height)

Resizes the viewer to the specified height and width.

Parameters

width
Integer. The width in pixels.

height
Integer. The height in pixels.

Example This example resizes the viewer when the new width is fewer than 1000 pixels and the new height is fewer than 650 pixels:

```
function resizeViewer(width,height){
   if ((width < 1000) && (height < 650)){
      viewer.resizeTo(width,height);
   }
}
```

rollback

Syntax void XTabAnalyzer.rollback(function callback)

Rolls back all changes in the viewer since the last commit( ) call and refreshes the viewer's content. The value of ivMode must be true for rollback( ) to function.

Parameter **callback**
Function. The callback function called after rollback finishes.

Example This example rolls back all changes to the viewer made since the last commit or submit function call:

```
function rollbackViewer( ){
   viewer.rollback( );
}
```

setGadgetId

Syntax void XTabAnalyzer.setGadgetId(string gadgetId)

Sets the cross tab gadget ID. This function is used for dashboard integration.

Parameter **gadgetId**
String. The gadget ID used to render the cross tab.

Example This example sets the gadget ID:

```
function setGadgetID(id){
   viewer.setGadgetId(id);
}
```

setHeight

Syntax void XTabAnalyzer.setHeight(integer height)

Changes the height of the viewer.

Parameter **height**
Integer. The height in pixels.

Example This example retrieves the viewer's current height. When the current height is fewer than 630 pixels, the example code doubles the viewer's height.

```
function doubleHeight( ){
   var height = viewer.getHeight( );
   if (height < 630){
      height = height * 2;
      viewer.setHeight(height);
      viewer.submit( );
   }
}
```

setIVMode

Syntax void XTabAnalyzer.setIVMode(boolean ivMode)

Sets IVMode for the viewer. Integrating a Data Analytics viewer with the Interactive Viewer affects the undo/redo feature. When set to true, all changes to the Data Analytics viewer must be committed as one transaction. The Interactive Viewer can undo or redo the entire batch.

Parameter **ivMode**
Boolean. Set to true if using IV mode.

Example This example sets IVMode for the viewer:

```
function setViewerMode(mode){
   viewer.setIVMode(mode);
}
```

setLeft

Syntax void XTabAnalyzer.setLeft(integer left)

Sets the position of the viewer's left margin.

Parameter **left**
Integer. The left margin for the viewer in pixels.

Example This example retrieves the left margin of the viewer and moves the margin 20 pixels to the right when the margin is less than 45 pixels from the edge of the screen:

```
function moveLeftMargin( ){
   var left = viewer.getLeft( );
   if (left < 45){
      viewer.setLeft(left + 20);
      viewer.submit( );
   }
}
```

setOnClosed

Syntax void XTabAnalyzer.setOnClosed(function callback)

Sets a callback function to call when a viewer pop-up closes.

Parameter **callback**
Function. The function to call when the pop-up closes.

Example This example checks to see if a pop-up window is active and sets a callback function to trigger when the pop-up closes:

```
function setPopupCloser( ){
   if(viewer.isActive( )){
      viewer.setOnClosed(closerCallbackFunctionName);
   }
}
```

setPageNum

Syntax void XTabAnalyzer.sePageNum(function pageNum)

Sets the page number.

Parameter **pageNum**
Integer. The page number.

Example This example sets the sets the page number to the first page:

```
function setPageNumberToFirst( ){
   if(viewer.isActive( )){
      viewer.setPageNum(1);
   }
}
```

setPosition

Syntax void XTabAnalyzer.setPosition(string position)

Sets the CSS position attribute.

Parameter **position**
String. The value for the CSS position attribute.

Example This example changes the type of CSS positioning in use:

```
function changePosition( ){
   var pos = viewer.getPosition( );
   if (pos == 'relative'){
      viewer.setPosition('absolute');
      viewer.submit( );
   }
}
```

setParameterValues

Syntax void XTabAnalyzer.setParameterValues(actuate.xtabanalyzer.ParameterValue[ ] parameterValues)

Sets the CSS position attribute.

Parameter **parameterValues**
actuate.xtabanalyzer.ParameterValue[ ] or actuate.parameter.ParameterValue[ ].
An array of parameter values.

setReportletDocumentMode

Syntax void XTabAnalyzer.setReportletDocumentMode(boolean reportletMode)

Sets whether the viewer displays documents as Reportlets.

Parameter **reportletMode**
Boolean. True indicates Reportlet display mode.

setReportName

Syntax void XTabAnalyzer.setReportName(string reportName)

Sets the report file name for the viewer. The file must be a report document file or report design file.

Parameter **reportName**
String. The name of the report file.

Example This example sets the report name to reportfile.rptdocument and reloads the Interactive Crosstabs viewer with its content:

```
function run( ){
   container = document.getElementById("acviewer");
   viewer = new actuate.XTabAnalyzer(container);
   viewer.setReportName("reportfile.rptdocument");
   viewer.submit( );
}
```

setService

Syntax void XTabAnalyzer.setService(string iPortalURL, actuate.RequestOptions requestOptions)

Sets the Actuate web application URL. This function can request options for that URL.

Parameters **iPortalURL**
String. The URL of the Actuate web application.

requestOptions
actuate.RequestOptions object. Request options for the web application. This parameter is optional.

Example This example sets the service and request options:

```
function setServerOptions(URL,options){
   viewer.setService(URL,options);
}
```

setSupportSVG

Syntax void XTabAnalyzer.setSupportSVG(boolean svgFlag)

Sets a flag indicating whether or not the browser supports SVG.

Parameter **svgFlag**
Boolean. Flag indicating SVG support in the browser. This parameter's value is true when the browser supports SVG and false in all other cases.

Example This example sets the browser's level of SVG support:

```
function setSVG(flag){
   viewer.setSupportSVG(flag);
}
```

setTop

Syntax void XTabAnalyzer.setTop(integer top)

Sets the top margin for the viewer.

Parameter **top**
Integer. The top margin for the viewer in pixels.

Example This example retrieves the current top margin for the viewer and moves the margin 20 pixels down the screen when the current position of the margin is fewer than 45 pixels from the top of the screen:

```
function moveTopMargin( ){
   var top = viewer.getTop( );
   if (top < 45){
      top = top + 20;
      viewer.setTop(top);
      viewer.submit( );
   }
}
```

setUIOptions

Syntax void XTabAnalyzer.setUIOptions(actuate.xtabanalyzer.uioptions options)

Sets the user interface options enabled for the viewer.

Parameter **options**
Actuate.xtabanalyzer.uioptions object. The options object for the viewer.

Example This example retrieves the user interface options and sets one of the UIOptions values:

```
function resetUIOptions( ){
   var options = viewer.getUIOptions( );
   options.enableToolbar(false);
   viewer.setUIOptions(options);
}
```

setWidth

Syntax void XTabAnalyzer.setWidth(integer width)

Sets the width for the viewer.

Parameter **width**
Integer. The width for the viewer in pixels.

Example This example retrieves the width of the viewer. When the viewer is fewer than 630 pixels wide, the example code doubles the viewer's width:

```
function doubleWidth( ){
   var width = viewer.getWidth( );
   if (width < 630){
      viewer.setWidth(width * 2);
      viewer.submit( );
   }
}
```

setXTabBookmark

Syntax void XTabAnalyzer.setXTabBookmark(string bookmark)

Sets the bookmark for a cross tab to render in the viewer.

Parameter **bookmark**
String. The bookmark for a cross tab.

Example This example retrieves the bookmark for the cross tab the viewer is associated with, changes the bookmark, and reloads the bookmark. This functionality enables the use of multiple cross tab elements within a single design.

```
function changeBookmark( ){
   var oldBookMark = viewer.getXTabBookmark( );
   viewer.setXTabBookmark("crosstab2");
   viewer.submit( );
}
```

setXTablid

Syntax void XTabAnalyzer.setXTablid(string iid)

Sets the instance ID for viewer rendering. This function is useful in integration with Interactive Viewer, and supports the ability of Interactive Viewer to obtain and use the cross tab instance ID.

Parameter **iid**
String. The instance ID.

Example This example sets the cross tab instance ID:

```
function setxtabInstance(id){
   viewer.setXTablid(id);
}
```

submit

Syntax void XTabAnalyzer.submit(function callback, boolean rerun)

Submits requests to the server for the Interactive Crosstabs viewer. This method triggers an AJAX request to submit all pending operations for this object. The server returns a response after processing the pending operations. The results render on the page in the Interactive Crosstabs container. The submit( ) function throws an exception when another submit( ) operation is pending. A CONTENT_CHANGED event fires when the Interactive Crosstabs content changes.

Parameters **callback**
Function. Optional. A function called when submit completes. This function receives the current XTabAnalyzer object as an input parameter.

rerun
Boolean. Optional. Indicates whether re-run the report design when it refreshes. Default to true.

Example This example retrieves the left margin of the viewer and expands the margin. The change does not take effect until submit( ) executes. The submit( ) function calls the function in the submitCallback parameter when submit( ) finishes executing. The callback function contains any processing that must occur after submit( ) finishes. Do not place code after the submit( ) call in the same function because submit( ) is asynchronous.

```
function moveLeftMargin( ){
   var left = viewer.getLeft( );
   if (left < 45){
      viewer.setLeft(left + 20);
      viewer.submit(submitCallback);
   }
}
```

Class actuate.xtabanalyzer.Crosstab

Description The actuate.xtabanalyzer.Crosstab class represents a cross tab report element.

Constructor

Syntax actuate.xtabanalyzer.Crosstab( )

Constructs a new Crosstab object.

Function summary

Table 15-3 lists actuate.xtabanalyzer.Crosstab functions.

Table 15-3 actuate.xtabanalyzer.Crosstab functions

Function	Description
addDimension( )	Adds a dimension to the cross tab
addMeasure( )	Adds a measure to the cross tab
applyOptions( )	Sets options for the cross tab
changeMeasureDirection( )	Switches measure direction
clearFilters( )	Clears cross tab filters
drill( )	Drills up or down measure levels, replacing drill and filter conditions
drillDown( )	Drills down a measure level, updating drill conditions
drillUp( )	Drills up a measure level, updating drill conditions
editMeasure( )	Edits a measure
getBookmark( )	Retrieves the cross tab element bookmark
getColumn( )	Retrieves table data by column index
getData( )	Returns the data from a cross tab
getHtmlDom( )	Retrieves the HTML DOM object
getPageContent( )	Retrieves the content of the page the cross tab belongs to
getRow( )	Retrieves table data by row index
getType( )	Retrieves the report element type
hideDetail( )	Hides the detail of a specified level

(continues)

Table 15-3 actuate.xtabanalyzer.Crosstab functions (continued)

Function	Description
pivot( )	Pivots the cross tab
removeDimension( )	Removes a dimension from the cross tab
reorderDimension( )	Removes a measure from the cross tab
removeMeasure( )	Reorders a dimension
reorderMeasure( )	Reorders a measure
setFilters( )	Sets the cross tab's filters
setSorters( )	Sets the cross tab's sorters
setTotals( )	Sets the cross tab's totals
showDetail( )	Shows details to the lower level
submit( )	Applies changes made to the cross tab

addDimension

Syntax void Crosstab.addDimension(actuate.xtabanalyzer.Dimension dimension)

Adds a dimension to the cross tab object.

Parameter **dimension**
actuate.xtabanalyzer.Dimension object. The dimension to add.

Example This example adds a date-based, multi-level dimension to a cross tab:

```
function addDimension( ){
// Create a dimension for dates in the first column
   var dimension = new actuate.xtabanalyzer.Dimension( );
   dimension.setIndex(0);
   dimension.setAxisType(actuate.xtabanalyzer.Dimension.
                         COLUMN_AXIS_TYPE);
   dimension.setDimensionName("dates");

// Create levels using levels from the data cube.
   var level = new actuate.xtabanalyzer.Level( );
   level.setLevelName("year");
   dimension.addLevel(level);
   var level = new actuate.xtabanalyzer.Level( );
   level.setLevelName("quarter");
   dimension.addLevel(level);

// Add the dimension to the cross tab.
   crosstab.addDimension(dimension);
   crosstab.submit( );
}
```

addMeasure

Syntax void Crosstab.addMeasure(actuate.xtabanalyzer.Measure measure, integer options)

Adds a measure to the cross tab object.

Parameters

measure
actuate.xtabanalyzer.Measure object. The measure to add.

options
Integer. The options for the add measure operation. These options distinguish the function call's origin, which can be from another dialog or directly from the Actuate JavaScript API.

Example This example adds a measure to a cross tab:

```
function addMeasure( ){
//Create a measure for revenue organized by date and product line.
   var measure = new actuate.xtabanalyzer.Measure( );
   measure.setIndex(1);
   measure.setMeasureName("Quarter Rate");
   measure.setExpression("[revenue]/[revenue_SalesDate
                        /year_Product/PRODUCTLINE]");

// Apply the measure to the cross tab
   crosstab.addMeasure(measure);
   crosstab.submit( );
}
```

In this example, the expression set with setExpression( ) is in EasyScript, which is described in *Using Actuate BIRT Designer Professional*.

applyOptions

Syntax void Crosstab.applyOptions(string | actuate.xtabanalyzer.Options measureDirection, string rowMirrorStartingLevel, string columnMirrorStartingLevel, string emptyCellValue)

Sets measure direction, empty settings, row mirror starting level, column mirror starting level, and empty cell value.

Parameters

measureDirection
String or actuate.xtabanalyzer.Options object. When measureDirection is a string, measureDirection is set to horizontal or vertical and the other parameters set options individually. When an actuate.xtabanalyzer.Options object is specified, all the options are set using settings from this object and applyOptions ignores all subsequent parameters.

rowMirrorStartingLevel
String. Sets the mirror starting level empty setting for a row.

columnMirrorStartingLevel
String. Sets the mirror starting level empty setting for a column.

emptyCellValue
String. Sets the value of an empty cell.

changeMeasureDirection

Syntax void Crosstab.changeMeasureDirection( )

Switches the measure direction between horizontal and vertical.

Example This example changes the measure direction:

```
function changeMeasureDirection( ){
   if( crosstab ){
      crosstab.changeMeasureDirection( );
      crosstab.submit( );
   }
}
```

clearFilters

Syntax void Crosstab.clearFilters(actuate.xtabanalyzer.Level level, String filterType)

Clears the filters from a level.

Parameters

level
actuate.xtabanalyzer.Level object. Optional. The level from which to clear the filters. To clear all filters, do not specify a level.

filterType
String. Optional. The filter type. To clear all filter types, do not specify a filter type.

Example This example clears the filters from the level filterLevel:

```
function clearLevelFilters( ){
   if( crosstab ){
      crosstab.clearFilters("filterLevel");
      crosstab.submit( );
   }
}
```

drill

Syntax void Crosstab.drill(actuate.xtabanalyzer.Driller driller)

Drills up or down a dimension level. Removes all drill/filter conditions defined on specified dimension first, then adds new drill/filter conditions.

Parameter **driller**
actuate.xtabanalyzer.Driller object. The driller object specifies drill conditions on a dimension.

Example This example drills to a level within a dimension. Any existing drill conditions are replaced.

```
function drillToDimension(memberVal){
   var driller = new actuate.xtabanalyzer.Driller( );
   driller.setAxisType(actuate.xtabanalyzer.Dimension.
                       ROW_AXIS_TYPE);
   driller.addMember(memberVal);
   myCrosstab.drill(driller);
   myCrosstab.submit( );
}
```

drillDown

Syntax void Crosstab.drillDown(actuate.xtabanalyzer.Driller driller)

Drills down a dimension level. This method updates the drill conditions specified in the Driller object and leaves all other conditions in place.

Parameter **driller**
actuate.xtabanalyzer.Driller object. A drill condition object.

Example This example drills down a level within a dimension. Any existing drill conditions are unchanged.

```
function drillToDimension(memberVal){
   var driller = new actuate.xtabanalyzer.Driller( );
   driller.setAxisType(actuate.xtabanalyzer.Dimension.
                       ROW_AXIS_TYPE);
   driller.addMember(memberVal);
   myCrosstab.drillDown(driller);
   myCrosstab.submit( );
}
```

drillUp

Syntax void Crosstab.drillUp(actuate.xtabanalyzer.Driller driller)

Drills up a dimension level. This method updates the drill conditions specified in the Driller object and leaves all other conditions in place.

Parameter **driller**
A drill condition object.

Example This example drills up a level within a dimension. Any existing drill conditions are unchanged.

```
function drillToDimension( ){
   var driller = new actuate.xtabanalyzer.Driller( );
   driller.setAxisType(actuate.xtabanalyzer.Dimension.
                       ROW_AXIS_TYPE);
// Add the member list to the Driller. Add the Driller to the
// crosstab.
   driller.addMember(memberVal);
   myCrosstab.drillUp(driller);
   myCrosstab.submit( );
}
```

editMeasure

Syntax void Crosstab.editMeasure(actuate.xtabanalyzer.Meaure Measure, integer opts)

Edits a measure in the Computed Measure view.

Parameters

Measure
actuate.xtabanalyzer.Measure object. A measure to change.

opts
Integer. Optional. Options for the editMeasure function. These options distinguish the function call's origin, which can be from another dialog or directly from the Actuate JavaScript API.

Example This example edits a measure:

```
function editComputedMeasure( ){
   if( crosstab ){
      var measure = new actuate.xtabanalyzer.Measure( );
      measure.setMeasureName("measureName");
      measure.setExpression("measureExpression");
      crosstab.editMeasure(measure);
      crosstab.submit( );
   }
}
```

getBookmark

Syntax string Crosstab.getBookmark( )

Returns the bookmark that is associated with the cross tab element.

Returns String. The cross tab bookmark.

Example The following code retrieves the bookmark that is associated with the cross tab object:

```
function getCrosstabBookmark( ){
   var crosstabBookmark = crosstab.getBookmark( );
   if( !crosstabBookmark ){
      alert( "No cross tab bookmark found!" )
      return null;
   }
   return crosstabBookmark;
}
```

getColumn

Syntax string[ ] Crosstab.getColumn(integer columnIndex)

Returns the table data by column index.

Parameter **columnIndex**
Integer. The column index, starting with 1.

Returns String[ ]. The column data as an array of strings. This function returns null when the value of columnIndex is out of range. This function only returns data from the current visible page.

Example The following code retrieves data from a data column:

```
function getColumnData(index,value){
   var columnData = crosstab.getColumn(index);
   if( !columnData ){
      alert( "Invalid column index!" )
      return null;
   }
   return columnData[value];
}
```

getData

Syntax String[ ] Crosstab.getData(boolean forceReparse)

Returns the data in a cross tab.

Parameter **forceReparse**
Boolean. Forces a cache refresh when true.

Returns String[ ]. The data from the cross tab as an array of strings.

getHtmlDom

Syntax HTMLElement Crosstab.getHtmlDom( )

Returns the HTML element DOM object.

Returns HTMLElement. The DOM element containing the cross tab.

Example The following code retrieves the DOM object and uses the DOM object to retrieve an element within the document:

```
function getContainer(containerName){
   var HTMLDom = crosstab.getHtmlDom( );
   var container = HTMLDom.getElementById(containerName);
   return container;
}
```

getPageContent

Syntax actuate.xtabanalyzer.PageContent Crosstab.getPageContent( )

Returns the page content from the current page to which this cross tab belongs. This function returns the same information as XTabAnalyzer.getCurrentPageContent( ).

Returns actuate.xtabanalyzer.PageContent. The report content.

Example This example retrieves the page content:

```
function retrievePageContent( ){
   return crosstab.getPageContent( );
}
```

getRow

Syntax string[ ] Crosstab.getRow(integer rowIndex)

Returns table data based on row index.

Parameter **rowIndex**
Integer. The row index, starting with 1.

Returns String[ ]. The row data as an array of string values. This function returns null when the value of rowIndex is out of range. This function only returns data from the current visible page.

Example The following code retrieves data from a data row:

```
function getRowData(index,value){
   var rowData = crosstab.getRow(index);
   if( !rowData ){
      alert( "Invalid row index!" )
      return null;
   }

   return rowData[value];
}
```

getType

Syntax string Crosstab.getType( )

Returns the report element type.

Returns String containing the value "Crosstab".

hideDetail

Syntax void Crosstab.hideDetail(string levelName)

Hides details of the specified level.

Parameter **levelName**
String. The full name of a dimension level to hide.

Example This example hides lower level details in a level:

```
function hideDetail( ){
   if(crosstab){
      var levelName = "rollLevelName";
      crosstab.hideDetail(levelName);
      crosstab.submit( );
   }
}
```

pivot

Syntax void Crosstab.pivot( )

Pivots the cross tab.

Example This example pivots a cross tab:

```
function pivot(crosstab){
   crosstab.pivot( );
   crosstab.submit( );
}
```

removeDimension

Syntax void Crosstab.removeDimension(object dimension, integer axisType, integer[ ] levels)

Removes a dimension from the cross tab.

Parameters **dimension**
actuate.xtabanalyzer.dimension object, a dimension index, or a dimension name. The dimension to remove.

axisType
Integer. The dimension axis type. Axis type can be one of the following values:

- actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE
- actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE

levels
The levels assigned in the dimension, as an array of actuate.xtabanalyzer.Level objects, a level index array, or a level name array.

Example This example removes a dimension with several layers. The level names are in a text control named levelNames and are separated by semicolons.

```
function removeDimension( ){
   if(crosstab){
      crosstab.removeDimension("dimensionName",null,"levelName";);
      crosstab.submit( );
   }
}
```

reorderDimension

Syntax void Crosstab.reorderDimension(actuate.xtabanalyzer.Dimension dimension, integer axisType, integer newIndex, integer newAxisType)

Reorders a dimension within a cross tab. This function can change a dimension's index or axis type.

Parameters

dimension
actuate.xtabanalyzer.dimension object, or a dimension index or a dimension name. The dimension to reorder.

axisType
Integer. The dimension axis type. Axis type can be one of the following values:

- actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE
- actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE

newIndex
The new index for the dimension.

newAxisType
The new axis type.

Example This example changes the index and axis type of a dimension:

```
function changeDimensionOrder( ){
   var dimensionIndex = 5;
   var newDimensionIndex = 2;

   var axisType = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
   var newAxisType = actuate.xtabanalyzer.Dimension.
                     COLUMN_AXIS_TYPE;
   crosstab.reorderDimension(dimensionIndex, axisType,
                             newDimensionIndex, newAxisType);
   crosstab.submit( );
}
```

removeMeasure

Syntax void Crosstab.removeMeasure(actuate.xtabanalyzer.Measure measure)

void Crosstab.removeMeasure(integer measure)

void Crosstab.removeMeasure(string measure)

Removes a measure from the cross tab.

Parameter **measure**
actuate.xtabanalyzer.measure object, index, or name. The measure to remove.

Example This example removes a measure from a cross tab:

```
function removeMeasure( ){
   crosstab.removeMeasure("measureName");
   crosstab.submit( );
}
```

reorderMeasure

Syntax void Crosstab.reorderMeasure(actuate.xtabanalyzerMeasure measure, integer newIndex)

void Crosstab.reorderMeasure(integer measure,integer newIndex)

void Crosstab.reorderMeasure(string measure,integer newIndex)

Reorders a measure within a cross tab.

Parameters **measure**
actuate.xtabanalyzer.Measure object, or a measure index or a measure name. The measure to reorder.

newIndex
The new index for the measure.

Example This example reorders a measure:

```
function changeMeasureOrder( ){
   var index = 6;
   var newIndex = 3;
   crosstab.reorderMeasure(index,newIndex);
   crosstab.submit( );
}
```

setFilters

Syntax void Crosstab.setFilters(actuate.xtabanalyzer.Filter[ ] filters)

Sets an array of filters for the cross tab.

Parameter **filters**
Array of actuate.xtabanalyzer.Filter objects. The filter conditions.

Example This example creates a Filter object and then places it into the cross tab:

```
function filterLevel( ){
   var levelName = "levelName";
   var operator = "BETWEEN";
   var filterValue = "20000;50000";
   var filter = new actuate.xtabanalyzer.Filter(levelName,
                 operator);
   filter.setValues(filterValue.split(";"));
   crosstab.setFilters(filter);
   crosstab.submit( );
}
```

setSorters

Syntax void Crosstab.setSorters(actuate.xtabanalyzer.Sorter[ ] sorters)

Sets an array of sorters for the cross tab.

Parameter **sorters**
Array of actuate.xtabanalyzer.Sorter objects. The sort settings.

Example This example creates a sorter and adds it to the cross tab:

```
function sortLevel( ){
   var levelName = "levelName";
   var sortAscending = true;
   var sorter = new actuate.xtabanalyzer.Sorter(levelName);
   sorter.setAscending(sortAscending);
   crosstab.setSorters(sorter);
   crosstab.submit( );
}
```

setTotals

Syntax void Crosstab.setTotals(actuate.xtabanalyzer.GrandTotal[ ] grandTotals, actuate.xtabanalyzer.SubTotal[ ] subTotals)

Sets totals for the cross tab.

Parameters

grandTotals
Array of actuate.xtabanalyzer.GrandTotal objects. Grand totals. To set a subtotal, set this parameter to null.

subTotals
Array of actuate.xtabanalyzer.SubTotal objects. Subtotals.

Example This example adds a grand total to a cross tab:

```
function addGrandTotal( ){
   var grandTotal = new actuate.xtabanalyzer.GrandTotal( );
   grandTotal.setAxisType(
      actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);

   var total = new actuate.xtabanalyzer.Total( );
   total.setMeasureIndex(1);
   total.setAggregationFunction("SUM");
   total.setEnabled(true);
   grandTotal.addTotal(total);

   crosstab.setTotals(grandTotal);
   crosstab.submit( );
}
```

showDetail

Syntax void Crosstab.showDetail(string axisType)

Shows a level of detail within a cross tab.

Parameter

axisType
String. The dimension axis type. Axis type can be one of the following values:

- actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE
- actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE

Example This example uses showDetail to expose extra detail on a level:

```
function showDetail( ){
   var axisType = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
   crosstab.showDetail(axisType);
   crosstab.submit( );
}
```

submit

Syntax void Crosstab.submit(function callback)

Applies the changes made to this element. This is an asynchronous operation.

Parameter **callback**
Function. Optional. The function called when submit( ) completes. This function receives the current XTabAnalyzer object as an input parameter.

Example This example uses submit( ) to confirm changes to the cross tab:

```
function showDetail(crosstab){
   var axisType = actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE;
   crosstab.showDetail(axisType);
   crosstab.submit( );
}
```

Class actuate.xtabanalyzer.Dimension

Description The Dimension class specifies a cross tab Dimension object.

Constructor

Syntax actuate.xtabanalyzer.Dimension( )

The Dimension class is used to specify a Dimension object.

Function summary

Table 15-4 lists actuate.xtabanalyzer.Dimension functions.

Table 15-4 actuate.xtabanalyzer.Dimension functions

Function	Description
addLevel( )	Adds the level to the dimension
getAxisType( )	Returns the axis type
getDimensionName( )	Returns the dimension name
getIndex( )	Returns the index of the dimension
getLevels( )	Returns cross tab levels
getNewAxisType( )	Returns the new axis type
getNewIndex( )	Returns the new index
setAxisType( )	Sets the axis type
setDimensionName( )	Sets the dimension name
setIndex( )	Sets the index
setLevels( )	Sets the levels
setNewAxisType( )	Sets the new axis type
setNewIndex( )	Sets the new index axis type

addLevel

Syntax void Dimension.addLevel(actuate.xtabanalyzer.Level level)

Adds a level to the dimension.

Parameter **level**
actuate.xtabanalyzer.Level object. A level to add to the dimension.

Example This example adds a level to a dimension:

```
function addLvl(dimension,levelName){
   var level = new actuate.xtabanalyzer.Level( );
   level.setLevelName(levelName);
   dimension.addLevel(level);
}
```

getAxisType

Syntax integer Dimension.getAxisType( )

Returns the axis type for the dimension.

Returns Integer. The axis type can be one of the following values:

- actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE
- actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE

Example This example retrieves and sets the axis type:

```
function swapAxis(dimension){
   if (dimension.getAxisType( ) ==
         actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE){
      dimension.setNewAxisType(
      actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
   } else {
      dimension.setNewAxisType(
      actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
   }
}
```

getDimensionName

Syntax string Dimension.getDimensionName( )

Returns the name of this dimension.

Returns String. The dimension name.

Example This example retrieves the dimension name:

```
function getDimName(dimension){
   if(dimension){
      return dimension.getDimensionName( );
   }
   return null;
}
```

getIndex

Syntax integer Dimension.getIndex( )

Returns the dimension index.

Returns Integer. The dimension index.

Example This example retrieves and increments the index:

```
function incrementIndex(dimension){
   var newIndex = dimension.getIndex( ) + 1;
   dimension.setNewIndex(newIndex);
}
```

getLevels

Syntax actuate.xtabanalyzer.Level[ ] Dimension.getLevels( )

Returns the dimension levels.

Returns actuate.xtabanalyzer.Level[ ]. Array of dimension levels.

Example This example retrieves the dimension levels:

```
function getDimLevels(dimension){
   if(dimension){
      return dimension.getLevels( );
   }
   return null;
}
```

getNewAxisType

Syntax integer Dimension.getNewAxisType( )

Returns the new axis type.

Returns Integer containing the new axis type.

Example This example retrieves the new axis type:

```
function getNewDimAxis(dimension){
   if(dimension){
      return dimension.getNewAxisType( );
   }
   return null;
}
```

getNewIndex

Syntax integer Dimension.getNewIndex( )

Returns the new index.

Returns Integer. The new index.

Example This example retrieves the new index:

```
function getNewDimIndex(dimension){
   if(dimension){
      return dimension.getNewIndex( );
   }
   return null;
}
```

setAxisType

Syntax void Dimension.setAxisType(integer axisType)

Sets the axis type when creating a new dimension. Use setNewAxisType( ) to change a dimension that already exists.

Parameter **axisType**
The axis type for the dimension. The axis type has the following legal values:

- actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE
- actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE

Example This example sets the axis type for a new dimension:

```
function setRowAxis(dimension){
     dimension.setAxisType(
     actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
}
```

setDimensionName

Syntax void Dimension.setDimensionName(string dimensionName)

Sets the name for a dimension during its creation.

Parameter **dimensionName**
String. The name of the dimension.

Example This example sets the dimension name to a value taken from a page element:

```
function setDimensionName(dimension){
   var dimensionName = document.getElementById("dimensionName").
                       value;
   dimension.setDimensionName(dimensionName);
}
```

setIndex

Syntax void Dimension.setIndex(integer index)

Sets the index for the dimension.

Parameter **index**
The index of the dimension.

Example This example sets the dimension index to a value taken from a page element:

```
function setDimensionIndex(dimension){
   var dimensionIndex = document.getElementById("dimensionIndex").
                        value;
   dimension.setIndex(dimensionIndex);
}
```

setLevels

Syntax void Dimension.setLevels(xtabanalyzer.Level[ ] levels)

Sets levels for the dimension.

Parameter **levels**
Array of xtabanalyzer.Level objects representing the levels for the dimension.

Example This example sets the dimension levels:

```
function setDimensionLevels(dimension,levels){
   if (dimension && levels){
      dimension.setLevels(levels);
   }
}
```

setNewAxisType

Syntax void Dimension.setNewAxisType(integer newAxisType)

Sets the new axis type.

Parameter **newAxisType**
Integer. The new axis type.

Example This example retrieves and changes the axis type:

```
function swapAxis(dimension){
   if (dimension.getAxisType( ) ==
         actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE){
      dimension.setNewAxisType(
         actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
   } else {
      dimension.setNewAxisType(
         actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
   }
}
```

setNewIndex

Syntax void Dimension.setNewtIndex(integer newIndex)

Sets the new index.

Parameter **newIndex**
Integer. The new index.

Example This example retrieves and increments the index:

```
function incrementIndex(dimension){
   var newIndex = dimension.getIndex( ) + 1;
   dimension.setNewIndex(newIndex);
}
```

Class actuate.xtabanalyzer.Driller

Description The Driller class enables an application to drill down or up levels on a member within a dimension.

Constructor

Syntax actuate.xtabanalyzer.Driller( )

Creates a Driller object.

Function summary

Table 15-5 lists actuate.xtabanalyzer.Driller functions.

Table 15-5 actuate.xtabanalyzer.Driller functions

Function	Description
addMember( )	Adds a member to the drill condition
getDimension( )	Retrieves the driller dimension
getMembers( )	Retrieves the members used by the drill
setDimension( )	Sets the driller dimension
setMembers( )	Adds an array of members to the drill condition

addMember

Syntax void Driller.addMember(actuate.xtabanalyzer.MemberValue member)

Adds a dimension member to the drill condition. Functional candidates are Dimension members with levels.

Parameter **member**
actuate.xtabanalyzer.MemberValue object. A member value to add.

Example This example adds a member to a Driller object:

```
function drillDownDimension( ){
   var driller = new actuate.xtabanalyzer.Driller( );
   driller.setDimension(actuate.xtabanalyzer.Dimension.
                        ROW_AXIS_TYPE);
   var memberValue = new actuate.xtabanalyzer.
                     MemberValue("drillLevelName");
   memberValue.setValue("drillLevelValue");
   driller.addMember(memberValue);
```

```
   crosstab.drill( driller );
   crosstab.submit( );
}
```

getDimension

Syntax string Driller.getDimension( )

Returns the dimension name for the drill condition.

Returns String. A dimension name.

Example This example retrieves the dimension of the driller:

```
function getDrillerAxis(driller){
   if (driller){
      return driller.getDimension( );
   }
   return null;
}
```

getMembers

Syntax actuate.xtabanalyzer.MemberValue[ ] Driller.getMembers( )

returns the list of members assigned to the driller.

Returns Array of actuate.xtabanalyzer.MemberValue. A dimension member.

Example This example retrieves the members that a driller uses:

```
function getDrillerMembers(driller){
   if (driller){
      return driller.getMembers( );
   }
   return null;
}
```

setDimension

Syntax void Driller.setDimension(string dimension)

Sets the dimension for the driller by name.

Parameter **dimension**
String. A dimension name.

Example This example sets the dimension name for the driller:

```
function setRowAxis(driller){
   if (driller){
      dimension.setDimension("Row");
   }
}
```

setMembers

Syntax void Driller.setMembers(actuate.xtabanalyzer.MemberValue[ ] member)

Sets an array of members to the drill condition.

Parameter **member**
Array of actuate.xtabanalyzer.MemberValue objects. An array of members.

Example This example sets the axis type for the driller:

```
function setDrillerMembers(driller,members){
   if (driller && members){
      driller.setMembers(members);
   }
}
```

Class actuate.xtabanalyzer.EventConstants

Description Defines constants for xtabanalyzer events. Table 15-6 lists the cross tab analyzer event constants.

Table 15-6 actuate.xtabanalyzer.Dimension constants

Constant	Description
ON_CONTENT_CHANGED	Content changed event. Triggers when the displayed content has changed, for example when changing cross tab report content. The event handler takes an actuate.XTabAnalyzer object that represents the viewer for which the event occurred, as the only parameter.
ON_CONTENT_SELECTED	Content selected event. Triggers when a user clicks on report elements. The event handler takes the following parameters: ■ actuate.XTabAnalyzer: object viewer for which event occurred ■ actuate.xtabanalyzer.SelectedContent: the SelectedContent object
ON_EXCEPTION	Exception event. Triggers when an exception occurs during an asynchronous operation. The event handler takes the following arguments: ■ actuate.XTabAnalyzer: viewer for which the event occurred ■ actuate.Exception: Exception object
ON_SESSION_TIMEOUT	Session time-out event. When a session time-out event occurs and the user tries to perform any operation on a viewer, a prompt dialog appears asking the user whether or not to log in again. When the user chooses to log in again, the ON_SESSION_TIMEOUT event triggers. When no handler is registered for this event, a default built-in login dialog will be displayed. The event handler takes one parameter: an actuate.XTabAnalyzer object, representing the viewer where the event occurred.

Class actuate.xtabanalyzer.Exception

Description A container for an XTabAnalyzer exception that supports specific exceptions. The Exception class provides an object to pass to a callback function or event handler when an exception occurs. The Exception class contains references to the exception's origin, description, and messages.

Constructor

The Exception object is constructed when unspecified exceptions occur. The exceptions are divided into three types, which determine the contents of the Exception object. These types are:

- ERR_CLIENT: Exception type for a client-side error
- ERR_SERVER: Exception type for a server error
- ERR_USAGE: Exception type for a JSAPI usage error

Function summary

Table 15-7 lists actuate.xtabanalyzer.Exception functions.

Table 15-7 actuate.xtabanalyzer.Exception functions

Function	Description
getDescription( )	Returns details of the exception
getElement( )	Returns the report element for which the exception occurred, if available
getErrCode( )	Returns the error code for ERR_SERVER
getMessage( )	Returns a short message about the error
getType( )	Returns the type of error exception
isExceptionType( )	Returns Boolean indicating whether exception is of certain type

getDescription

Syntax string Exception.getDescription( )

Returns exception details as provided by the Server, Client, and User objects.

Returns String. A detailed description of the error. Information is provided according to the type of exception generated, as shown below:

- ERR_SERVER: The SOAP string

- ERR_CLIENT: For the Firefox browser, a list comprised of fileName+number+stack
- ERR_USAGE: Any value set when the object was created

Example This example consists of a function that registerEventHandler( ) set as a callback. The callback function takes an instance of the Exception class. Each of the functions for the Exception class can be called with the results formatted to create a message or for some other use.

```
function errorHandler(viewerInstance, exception){
   alert(exception.getDescription( ));
}
```

getElement

Syntax string Exception.getElement( )

Returns the report element for which the exception occurred, if available.

Returns String. The report element for which the exception occurred.

Example This example uses getElement( ):

```
function errorHandler(viewerInstance, exception){
   alert("Error in " + exception.getElement( ));
}
```

getErrCode

Syntax string Exception.getErrCode( )

Returns the error code for ERR_SERVER.

Returns String. The error code for ERR_SERVER.

Example This example uses getErrCode( ):

```
function errorHandler(viewerInstance, exception){
   alert(exception.getErrCode( ));
}
```

getMessage

Syntax string Exception.getMessage( )

Returns a short message about the error.

Returns String. A short message about the exception.

Example This example uses getMessage( ):

```
function errorHandler(viewerInstance, exception){
   alert(exception.getMessage( ));
}
```

getType

Syntax string Exception.getType( )

Returns the type of exception error.

Returns String. The errType exception type.

Example This example uses getType( ):

```
function errorHandler(viewerInstance, exception){
   alert(exception.getType( ));
}
```

isExceptionType

Syntax boolean Exception.isExceptionType(object exceptionType)

Checks an exception's type for a match against a specified type.

Parameter **exceptionType**
An exception type as string, or exception class. For example, "actuate.viewer.ViewerException" or actuate.viewer.ViewerException.

Returns True if the exception is of the stated type, false otherwise.

Example This example checks to see if the exception is a client error type:

```
function errorHandler(viewerInstance, exception){
   if (exception.isExceptionType(ERR_CLIENT){
      alert("CLIENT ERROR");
   }
}
```

Class actuate.xtabanalyzer.Filter

Description The Filter class creates a filter condition on a cross tab dimension level. The condition is expressed as value1 operator value2. The values can either be a single value, or an array of values, depending on the operator. For example, IN can be expressed as value1 IN value2 value3 ... valueN.

Constructor

Syntax

actuate.data.Filter(string levelName, string levelAttributeName, string operator, string value, string filterType)

actuate.data.Filter(string levelName, string levelAttributeName, string operator, string value1, string value2, string filterType)

actuate.data.Filter(string levelName, string levelAttributeName, string operator, string[ ] values, string filterType)

Constructs a cross tab Filter object.

Parameters

levelName
String. The dimension level full name.

levelAttributeName
String. The dimension level attribute name.

operator
String. The operator can be any operator. Table 15-8 lists the valid filter operators and the number of arguments to pass to the constructor or setValues( ).

Table 15-8 Filter operators

Operator	Description	Number of arguments
BETWEEN	Between an inclusive range	2
BOTTOM_N	Matches the bottom n values	1
BOTTOM_PERCENT	Matches the bottom percent of the values	1
EQ	Equal	1
FALSE	Matches false Boolean values	0
GREATER_THAN	Greater than	1
GREATER_THAN_OR_EQUAL	Greater than or equal	1

Table 15-8 Filter operators

Operator	Description	Number of arguments
IN	Matches any value in a set of values	1+
LESS_THAN	Less than	1
LESS_THAN_OR_EQUAL	Less than or equal	1
LIKE	Search for a pattern	1
MATCH	Equal	1
NOT_BETWEEN	Not between an inclusive range	2
NOT_EQ	Not equal	1
NOT_IN	Does not match any value in a set of values	1+
NOT_LIKE	Searches for values that do not match a pattern	1
NOT_MATCH	Not equal	1
NOT_NULL	Is not null	0
NULL	Is null	0
TOP_N	Matches the top n values	1
TOP_PERCENT	Matches the top percent of the values	1
TRUE	Matches true Boolean values	0

value
String. The value to compare to the column value.

value1
String. The first value to compare to the column value for the BETWEEN or NOT_BETWEEN operators.

value2
String. The second value to compare to the column value for the BETWEEN or NOT_BETWEEN operators.

values
Array of strings. The values to compare to the column value for the IN and NOT_IN operators.

filterType
String. The filter type.

Function summary

Table 15-9 lists actuate.xtabanalyzer.Filter functions.

Table 15-9 actuate.xtabanalyzer.Filter functions

Function	Description
getFilterType( )	Returns the filter type
getLevelAttributeName( )	Returns the dimension level attribute name
getLevelName( )	Returns the name of the filtered level
getOperator( )	Returns the filter operator
getValues( )	Returns the set of values the filter is using
setFilterType( )	Sets the filter type
setLevelAttributeName( )	Sets the dimension level attribute name
setLevelName( )	Sets the dimension level name
setOperator( )	Sets the filter operator
setValues( )	Sets the values for the filter

getFilterType

Syntax string Filter.getFilterType( )

Returns the filter type.

Returns String. The filter type.

Example This example retrieves the filter type for a filter:

```
function getType(filter){
   if(filter){
      return filter.getFilterType( );
   }
   else{
      return null;
   }
}
```

getLevelAttributeName

Syntax string Filter.getLevelAttribute Name( )

Returns the name of the dimension level attribute to which this filter applies.

Returns String. The level attribute name.

Example This example retrieves the filter level attribute name for a filter:

```
function getLevelAttribute(filter){
   if(filter){
      return filter.getLevelAttributeName( );
   }
   else{
      return null;
   }
}
```

getLevelName

Syntax string Filter.getLevelName( )

Returns the name of the dimension level to which this filter applies.

Returns String. A level name.

Example This example retrieves the filter level name for a filter:

```
function getLevel(filter){
   if(filter){
      return filter.getLevelName( );
   }
   else{
      return null;
   }
}
```

getOperator

Syntax string Filter.getOperator( )

Returns the filter operator.

Returns String. The filter operator.

Example This example retrieves the filter operator:

```
function getFilterOp(filter){
   if(filter){
      return filter.getOperator( );
   }else{
      return null;
   }
}
```

getValues

Syntax string[ ] Filter.getValues( )

Returns an array containing the values used in the filter.

Returns Array of strings. The values for the filter.

Example This example retrieves the filter level name for a filter:

```
function getFilterOp(filter){
   if(filter){
      return filter.getValues( );
   }else{
      return null;
   }
}
```

setFilterType

Syntax void Filter.setFilterType(string filterType)

Sets the filter type to filter.

Parameter **filterType**
String. The type of filter.

Example This example sets the filter type to equality:

```
function filterLevel( ){
   var filterType = "equality";
   var filter = new actuate.xtabanalyzer.Filter("levelName",
      "attributeName","EQ","2000","blank");
   filter.setFilterType(filterType);
   crosstab.setFilters( filter );
   crosstab.submit( );
}
```

setLevelAttributeName

Syntax void Filter.setLevelAttributeName(string levelAttributeName)

Sets the dimension level attribute to filter on by name.

Parameter **levelAttributeName**
String. The name of the level attribute to filter.

Example This example sets the level attribute name to attributeName:

```
function filterLevel( ){
   var attributeName = "attributeName";
   var filter = new actuate.xtabanalyzer.Filter("levelName",
                    "blank","EQ","2000","equality");
   filter.setLevelAttributeName(attributeName);
   crosstab.setFilters( filter );
   crosstab.submit( );
}
```

setLevelName

Syntax void Filter.setLevelName(string level)

Sets the level to filter by name.

Parameter **level**
String. The name of the level to filter.

Example This example sets the filter level name to levelName:

```
function filterLevel( ){
   var levelName = "levelName";
   var filter = new actuate.xtabanalyzer.Filter("blank",
                    "attributeName",EQ","2000","equality");
   filter.setLevelName(levelName);
   crosstab.setFilters( filter );
   crosstab.submit( );
}
```

setOperator

Syntax void Filter.setOperator(string operator)

Sets the filter operator.

Parameter **operator**
String. The filter operator.

Example This example sets the filter operator to EQ:

```
function filterLevel( ){
   var operator = "EQ";
   var filter = new actuate.xtabanalyzer.Filter("levelName",
                    "attributeName","NOT","2000","equality");
   filter.setOperator(operator);
   crosstab.setFilters( filter );
   crosstab.submit( );
}
```

setValues

Syntax void Filter.setValues(string[ ] value1, string[ ] value2)

Sets the values for the filter.

Parameters

value1
String or array of strings. The first value of the filter.

value2
String or array of strings. Optional. The second value of the filter.

Example This example sets the filter values to 2000 and 2004:

```
function filterLevel( ){
   if(crosstab){
      var filterValue = "2000;2004";
      var filter = new actuate.xtabanalyzer.Filter
                   ("levelName","attributeName","BETWEEN");
      filter.setValues(filterValue.split(";") );
      crosstab.setFilters( filter );
      crosstab.submit( );
   }
}
```

Class actuate.xtabanalyzer.GrandTotal

Description The GrandTotal class specifies a cross tab GrandTotal object.

Constructor

Syntax actuate.xtabanalyzer.GrandTotal( )

Constructs a new GrandTotal object.

Function summary

Table 15-10 lists actuate.xtabanalyzer.GrandTotal functions.

Table 15-10 actuate.xtabanalyzer.GrandTotal functions

Function	Description
addTotal( )	Adds a total
getAxisType( )	Returns the axis type
getTotals( )	Returns the totals array
getType( )	Returns the grand total type
setAxisType( )	Sets the axis type
setTotals( )	Sets the totals array

addTotal

Syntax void GrandTotal.addTotal(object total)

Adds a total to the cross tab.

Parameter **total**
actuate.xtabanalyzer.total. The total to add to the cross tab.

Example This example adds totals to a grand total:

```
function addTotal(grandTotal){
// The indexStr can be set from a web page or other source as
// necessary.
   var indexStr = "0;1;2;3;4";
   var indexs = indexsStr.split(";");
   var count = indexs.length;
   var measureIndexs = [ ];
   for(var i = 0;i < count;i++){
      measureIndexs.push(parseInt(indexs[i]));
   }
```

```
   for( var i = 0; i < measureIndexs.length; i++){
      var total = new actuate.xtabanalyzer.Total( );
      total.setMeasureIndex(measureIndexs[i]);
      total.setAggregationFunction("SUM");
      total.setEnabled(true);
      grandTotal.addTotal(total);
   }
}
```

getAxisType

Syntax integer GrandTotal.getAxisType( )

Returns the axis type for the total.

Returns Integer. The following values are legal axis types:

- actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE
- actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE

Example This example retrieves and sets the axis type:

```
function swapAxis(grandtotal){
   if (grandtotal.getAxisType( ) ==
         actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE){
      grandtotal.setNewAxisType(
      actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
   } else {
      grandtotal.setNewAxisType(
      actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
   }
}
```

getTotals

Syntax object[ ] GrandTotal.getTotals( )

Returns an array containing the totals.

Returns Array of Total objects. The totals.

Example This example retrieves totals from a GrandTotal object:

```
var totalsArray = [ ];
function getTotals(grandTotal,totalsArray){
   totalsArray = grandTotal.getTotals( );
}
```

getType

Syntax string GrandTotal.getType( )

Returns the type for the total.

Returns String. The total type.

setAxisType

Syntax void GrandTotal.setAxisType(integer axisType)

Sets the axis type for the total.

Parameter **axisType**
Integer. Axis type for the total.

Example This example retrieves and sets the axis type:

```
function swapAxis(grandtotal){
   if (grandtotal.getAxisType( ) ==
         actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE){
      grandtotal.setNewAxisType(
      actuate.xtabanalyzer.Dimension.COLUMN_AXIS_TYPE);
   } else {
      grandtotal.setNewAxisType(
      actuate.xtabanalyzer.Dimension.ROW_AXIS_TYPE);
   }
}
```

setTotals

Syntax void GrandTotal.setTotals(actuate.xtabanalyzer.Total[ ] totals)

Sets totals as an array.

Parameter **totals**
Array of actuate.xtabanalyzer.Total objects to add to the grand total.

Example This example copies the totals from grandtotal1 into grandtotal2:

```
grandtotal2.setTotals(grandtotal1.getTotals( ));
```

Class actuate.xtabanalyzer.Level

Description Defines a cross tab dimension level, its controls, and content.

Constructor

Syntax actuate.xtabanalyzer.Level( )

Creates a cross tab Level object.

Function summary

Table 15-11 lists actuate.xtabanalyzer.Level functions.

Table 15-11 actuate.xtabanalyzer.Level functions

Function	Description
addAttribute( )	Adds the level attribute
getAttributes( )	Returns the level attributes
getIndex( )	Returns the index of the level
getLevelName( )	Returns the level name
setIndex( )	Sets the index level
setLevelName( )	Sets the level name

addAttribute

Syntax void Level.addAttribute(actuate.xtabanalyzer.LevelAttribute attr)

Adds the level attribute.

Parameter **index**
actuate.xtabanalyzer.LevelAttribute object. A level attribute.

Example This example sets a name for newly created level attribute and assigns the attribute to a level:

```
var attribute = new actuate.xtabanalyzer.LevelAttribute( );
attribute.setName("pounds");
level.addLevelAttribute( attribute );
```

getAttributes

Syntax actuate.xtabanalyzer.LevelAttribute[ ] Level.getAttributes( )

Returns the level attributes.

Returns Array of actuate.xtabanalyzer.LevelAttribute objects. The level attributes.

Example This example retrieves the level index and stores it in a variable called lattributes:

```
var lattributes = new actuate,xtabanalyzer.LevelAttribute[ ];
lattributes = level.getAttributes( );
```

getIndex

Syntax integer Level.getIndex( )

Returns the level index.

Returns Integer. The level index.

Example This example retrieves the level index:

```
function levelIndex(level){
   if (level){
      return level.getIndex( );
   }
   return null;
}
```

getLevelName

Syntax string Level.getLevelName( )

Returns the level name.

Returns String. The level name.

Example This example retrieves the level name:

```
function levelName(level){
   if (level){
      return level.getLevelName( );
   }
   return null;
}
```

setIndex

Syntax void Level.setIndex(integer index)

Sets the level index.

Parameter **index**
Integer. The level index.

Example This example sets the level index:

```
function assignIndex(level,index){
   if (level){
      return level.setIndex(index);
   }
}
```

setLevelName

Syntax void Level.setLevelName(string levelName)

Sets the level name.

Parameter **levelName**
String. The level name.

Example This example sets level names for newly created levels:

```
var levelNames ="year;month;day";
...
function addLevels(dimension,levelNames);{
   var levelNamesArray = levelNames.split(";");
   for( var i = 0; i < levelNames.length; i++ ){
      var level = new actuate.xtabanalyzer.Level( );
      level.setLevelName(levelNames[i]);
      dimension.addLevel( level );
   }
}
```

Class actuate.xtabanalyzer.LevelAttribute

Description Defines an attribute for a level.

Constructor

Syntax actuate.xtabanalyzer.LevelAttribute( )

Creates a cross tab level attribute object.

Function summary

Table 15-12 lists actuate.xtabanalyzer.LevelAttribute functions.

Table 15-12 actuate.xtabanalyzer.Level functions

Function	Description
getName( )	Returns the level attribute name
setName( )	Sets the level attribute name

getName

Syntax string LevelAttribute.getName( )

Returns the level attribute name.

Returns String. A level attribute name.

Example This example retrieves the level attribute name and stores it in a variable attname:

```
var attname = levelattribute.getName( );
```

setName

Syntax void LevelAttribute.setName(string attributeName)

Sets the level attribute name.

Parameter **attributeName**
String. The level attribute name.

Example This example sets a name for newly created level attribute and assigns the attribute to a level:

```
var attribute = new actuate.xtabanalyzer.LevelAttribute( );
attribute.setName("pounds");
level.addLevelAttribute( attribute );
```

Class actuate.xtabanalyzer.Measure

Description Defines a cross tab measure.

Constructor

Syntax actuate.xtabanalyzer.Measure( )

Creates a cross tab measure object.

Function summary

Table 15-13 lists actuate.xtabanalyzer.Measure functions.

Table 15-13 actuate.xtabanalyzer.Measure functions

Function	Description
getAggregationFunction( )	Returns the aggregation function name
getDataType( )	Returns the computed column data type
getExpression( )	Returns the computed measure expression
getIndex( )	Returns the measure index
getMeasureName( )	Returns the measure name
getNewIndex( )	Returns the new index
setAggregationFunction( )	Sets the aggregation function name
setDataType( )	Sets the computed column data type
setExpression( )	Sets the computed measure expression
setIndex( )	Sets the measure index
setMeasureName( )	Sets the measure name
setNewIndex( )	Sets the new index

getAggregationFunction

Syntax string Measure.getAggregationFunction( )

Returns the aggregation function name.

Returns String. An aggregation function name.

Example This example changes the aggregation function:

```
function swapMeasureAggregation(measure){
   if (measure.getAggregation( ) == "EQ"){
      measure.setAggregation("NE");
   }else{
      measure.setAggregation("EQ");
   }
}
```

getDataType

Syntax string Measure.getDataType( )

Returns the computed column data type.

Returns String. The data type.

Example This example retrieves the computed column data type:

```
function getColumnDataType(measure){
   if (measure){
      return measure.getDataType( );
   }
   return null;
}
```

getExpression

Syntax string Measure.getExpression( )

Returns the computed measure expression.

Returns String. An expression.

Example This example retrieves the computed measure expression:

```
function getMeasureExpression(measure){
   if (measure){
      return measure.getExpression( );
   }
   return null;
}
```

getIndex

Syntax integer Measure.getIndex( )

Returns the measure index.

Returns Integer. The measure index.

Example This example retrieves the measure index:

```
function getMeasureIndex(measure){
   if (measure){
      return measure.getIndex( );
   }
   return null;
}
```

getMeasureName

Syntax string Measure.getMeasureName( )

Returns the measure name.

Returns String. The name of the measure.

Example This example retrieves the measure name:

```
function getMeasureName(measure){
   if (measure){
      return measure.getMeasureName( );
   }
   return null;
}
```

getNewIndex

Syntax integer Measure.getNewIndex( )

Retrieves the new index. The new index is set by setNewIndex and represents the index value the measure has after submit( ) finishes executing.

Returns Integer. The new index.

Example This example retrieves the new measure index:

```
function getNewMeasureIndex(measure){
   if (measure){
      return measure.getNewIndex( );
   }
   return null;
}
```

setAggregationFunction

Syntax void Measure.setAggregationFunction(string aggregationFunction)

Sets the aggregation function name.

Parameter **aggregationFunction**
String. The aggregation function name.

Example This example changes the aggregation function:

```
function swapMeasureAggregation(measure){
   if (measure.getAggregation( ) == "EQ"){
      measure.setAggregation("NE");
   }else{
      measure.setAggregation("EQ");
   }
}
```

setDataType

Syntax void Measure.setDataType(string dataType)

Sets the computed column data type name.

Parameter **dataType**
String. The data type.

setExpression

Syntax void Measure.setExpression(string expression)

Sets the computed measure expression.

Parameter **expression**
String. The computed measure expression.

Example This example uses setExpression:

```
function addMeasure(viewer){
   var crosstab = getCrosstab(viewer);
   if(crosstab){
      var measureName  = "measureName";
      var measureExpression  =
         "[revenue]/[revenue_SalesDate/year_Product/PRODUCTLINE]";

      var measure = new actuate.xtabanalyzer.Measure( );
      measure.setIndex(1);
      measure.setMeasureName(measureName);
      measure.setExpression(measureExpression);

      crosstab.addMeasure(measure);
      crosstab.submit( );
   }
}
```

setIndex

Syntax void Measure.setIndex(integer index)

Sets the index.

Parameter **index**
Integer. The index of this measure.

Example This example uses setIndex to add a new measure to a cross tab:

```
function setIndex(measure, index){
   measure.setIndex(index);
}
```

setMeasureName

Syntax void Measure.setMeasureName(string measureName)

Sets the measure name.

Parameter **measureName**
String. The measureName.

Example This example sets the measure name which is taken from a page element:

```
function renameMeasure(measure){
   var measureName = document.getElementById("measureName").value;
   measure.setMeasureName(measureName);
}
```

setNewIndex

Syntax void Measure.setNewIndex(integer newIndex)

Sets a new measure index.

Parameter **newIndex**
Integer. The new measure index.

Example This example changes the index for the measure:

```
function changeIndex(measure,index){
   if (measure){
      measure.setNewIndex(index);
   }
}
```

Class actuate.xtabanalyzer.MemberValue

Description Defines a member value used for sort, filter, or drill functionality.

Constructor

Syntax actuate.xtabanalyzer.MemberValue(levelName, value, (MemberValue))

Creates a MemberValue object for a given level and value. The object can contain multiple member values.

Parameters

levelName
String. Dimension level name of member.

value
String. Value for the member to contain.

MemberValue
Optional actuate.xtabanalyzer.MemberValue object. MemberValue object to add during construction.

Function summary

Table 15-14 lists actuate.xtabanalyzer.MemberValue functions.

Table 15-14 actuate.xtabanalyzer.MemberValue functions

Function	Description
addMember( )	Adds a member value object
getLevelName( )	Retrieves the level name
getMembers( )	Retrieves an array of members
getValue( )	Returns the level value
setLevelName( )	Sets the level name
setValue( )	Sets the member value

addMember

Syntax void MemberValue.addMember(actuate.xtabanalyzer.MemberValue member)

Adds a member value.

Parameter

member
actuate.xtabanalyzer.MemberValue object. A member value.

Example MemberValue is an embedded class that can be a single value or an array of values. This example has a single member that contains four members:

```
function addMembers(memberData){
   var mv1 = new MemberValue('dim/state','CA');
   var mv2 = new MemberValue('dim/state','CN');
   var mv3 = new MemberValue(memberData);
   var mv = new MemberValue('dim/country','USA');
   mv.addMember(mv1);
   mv.addMember(mv2);
   mv.addMember(mv3);
   return mv;
}
```

getLevelName

Syntax string MemberValue.getLevelName( )

Returns the level name of the member.

Returns String. The level name.

Example This example retrieves the level name for the member value:

```
function getLevelName(level){
   if (level){
      return level.getLevelName( );
   }
   return null;
}
```

getMembers

Syntax actuate.xtabanalyzer.MemberValue[ ] MemberValue.getMembers( )

Returns all the member value objects contained in this member value object.

Returns Array of actuate.xtabanalyzer.MemberValue. An array of MemberValue objects.

Example This example returns the number of members in a member object:

```
function getMemberCount(members){
   if (members){
      var membersArray[] = members.getMembers( );
      return membersArray.length;
   }
   return null;
}
```

getValue

Syntax string MemberValue.getValue( )

Returns the level value.

Returns String. The level value.

Example This example returns the value for the level:

```
function getMemberValue(members){
   if (members){
      return members.getValue( );
   }
   return null;
}
```

setLevelName

Syntax void MemberValue.setLevelName(string level)

Sets the level name.

Parameter **level**
String. The name of the level.

Example This example sets the level name:

```
function setMemberLevelName(member,lvlName){
   if (member){
      member.setLevelName(lvlName);
   }
}
```

setValue

Syntax void MemberValue.setValue(string level)

Sets the level value.

Parameter **level**
String. The value for the level.

Example This example sets the level value:

```
function setMemberLevelValue(member,lvlValue){
   if (member){
      member.setValue(lvlValue);
   }
}
```

Class actuate.xtabanalyzer.Options

Description The Options class specifies options for the cross tab.

Constructor

Syntax actuate.xtabanalyzer.Options(string measureDirection, string rowMirrorStartingLevel, string columnMirrorStartingLevel, string emptyCellValue, boolean enablePageBreak, integer rowPageBreakInterval, integer columnPageBreakInterval)

Creates an options object that contains options for how the cross tab displays data.

Parameters

measureDirection
String. The measure direction. Legal values for measure direction are:

- DIRECTION_HORIZONTAL
- DIRECTION_VERTICAL

rowMirrorStartingLevel
String. Row mirror starting level name.

columnMirrorStartingLevel
String. Column mirror starting level name.

emptyCellValue
String. Value to display for an empty cell.

enablePageBreak
Boolean. Enables page breaks when true.

rowPageBreakInterval
Integer. Row page break interval.

columnPageBreakInterval
Integer. Column page break interval.

grandTotalsDisplayOption
String. Grand totals display option. Legal values for total display options are:

- DIRECTION_HORIZONTAL
- DIRECTION_VERTICAL

subtotalsDisplayOption
String. Subtotals display option. Legal values for total display options are:

- DIRECTION_HORIZONTAL
- DIRECTION_VERTICAL

Function summary

Table 15-15 lists actuate.xtabanalyzer.Options functions.

Table 15-15 actuate.xtabanalyzer.Options functions

Function	Description
getColumnMirrorStartingLevel( )	Returns the column mirror starting level full name
getColumnPageBreakInterval( )	Returns the column page break interval
getEmptyCellValue( )	Returns the empty cell value
getEnablePageBreak( )	Returns the page break enabled or disabled status
getMeasureDirection( )	Returns the measure direction
getRowMirrorStartingLevel( )	Returns the row mirror starting level full name
getRowPageBreakInterval( )	Returns the row page break interval
setColumnMirrorStartingLevel( )	Sets the column mirror starting level full name
setColumnPageBreakInterval( )	Sets the column page break interval
setEmptyCellValue( )	Sets the empty cell value
setEnablePageBreak( )	Sets the flag to enable page breaks
setMeasureDirection( )	Sets the measure direction
setRowMirrorStartingLevel( )	Sets the row mirror starting level full name
setRowPageBreakInterval( )	Sets the row page break interval

getColumnMirrorStartingLevel

Syntax string Options.getColumnMirrorStartingLevel( )

Returns the column mirror starting level name.

Returns String. Column mirror starting level name.

Example This example retrieves the column mirror starting level:

```
function getColumnMirrorStart(options){
   if (options){
      return options.getColumnMirrorStartinglevel( );
   }
   return null;
}
```

getColumnPageBreakInterval

Syntax integer Options.getColumnPageBreakInterval( )

Returns the column page break interval.

Returns Integer. The column page break interval.

Example This example retrieves the column page break interval:

```
function getColumnPBInterval(options){
   if (options){
      return options.getColumnPageBreakInterval( );
   }
   return null;
}
```

getEmptyCellValue

Syntax string Options.getEmptyCellValue( )

Returns the empty cell value.

Returns String. Value to display for an empty cell.

Example This example retrieves the empty cell:

```
function getEmptyCell(options){
   if (options){
      return options.getEmptyCellValue( );
   }
   return null;
}
```

getEnablePageBreak

Syntax boolean Options.getEnablePageBreak( )

Returns the page break status.

Returns Boolean. Page breaks are enabled when the value is true.

Example This example retrieves the column page break interval when page breaks are enabled:

```
function getColumnPBEnabled(options){
   if (options.getEnablePageBreak( ))
      return options.getColumnPageBreakInterval( );
   else
      alert ("Page Breaks Not Enabled.");
   return null;
}
```

getMeasureDirection

Syntax string Options.getMeasureDirection( )

Returns the measure direction.

Returns String. The measure direction. Legal values for measure direction are:

- DIRECTION_HORIZONTAL
- DIRECTION_VERTICAL

Example This example retrieves the measure direction:

```
function getMeasureDirection(options){
   if (options){
      return options.getMeasureDirection( );
   }
   return null;
}
```

getRowMirrorStartingLevel

Syntax string Options.getRowMirrorStartingLevel( )

Returns the row mirror starting level name.

Returns String. Row mirror starting level name.

Example This example retrieves the row mirror starting level:

```
function getRowMirrorStart(options){
   if (options){
      return options.getRowMirrorStartinglevel( );
   }
   return null;
}
```

getRowPageBreakInterval

Syntax integer Options.getRowPageBreakInterval( )

Returns the row page break interval.

Returns Integer. The row page break interval.

Example This example retrieves the row page break interval:

```
function getRowPBInterval(options){
   if (options){
      return options.getRowPageBreakInterval( );
   }
   return null;
}
```

setColumnMirrorStartingLevel

Syntax void Options.setColumnMirrorStartingLevel(string levelName)

Sets the column mirror starting level name.

Parameter **levelName**
String. The column mirror starting level name.

Example This example sets the column mirror starting level:

```
function setColumnMirrorLevel(options,level)(
   if (options){
      options.setColumnMirrorStartingLevel(level);
   }
}
```

setColumnPageBreakInterval

Syntax void Options.setColumnPageBreakInterval(integer columnPageBreakInterval)

Sets the column page break interval.

Parameter **columnPageBreakInterval**
Integer. The column page break interval.

Example This example sets the column page break interval:

```
function setColumnPBInterval(options,interval)(
   if (options){
      options.setColumnPageBreakInterval(interval);
   }
}
```

setEmptyCellValue

Syntax void Options.setEmptyCellValue(string emptyCellValue)

Sets the empty cell value.

Parameter **emptyCellValue**
String. The empty cell value.

Example This example sets the empty cell value:

```
function setEmptyCell(options, cellValue)(
   if (options){
      options.setEmptyCellValue(cellValue);
   }
}
```

setEnablePageBreak

Syntax void Options.setEnablePageBreak(boolean enablePageBreak)

Enables or disables page breaks.

Parameter **enablePageBreak**
Boolean. Enables page breaks when true.

Example This example enables page breaks and sets the row page break interval:

```
function enablesetRowPBInterval(options,interval)(
   if (options){
      options.setEnablePageBreak(true);
      options.setRowPageBreakInterval(interval);
   }
}
```

setMeasureDirection

Syntax void Options.setMeasureDirection(string measureDirection)

Sets the measure direction.

Parameter **measureDirection**
String. The measure direction. The measure direction. Legal values for measure direction are:

- DIRECTION_HORIZONTAL
- DIRECTION_VERTICAL

Example This example sets the measure direction:

```
function setMeasureDirection(options,direction){
   if (options){
      options.setMeasureDirection(direction);
   }
}
```

setRowMirrorStartingLevel

Syntax void Options.setRowMirrorStartingLevel(string levelName)

Sets the row mirror starting level.

Parameter **levelName**
String. Row mirror starting level name.

Example This example sets the row mirror starting level:

```
function setRowMirrorLevel(options,level){
   if (options){
      options.setRowMirrorStartingLevel(level);
   }
}
```

setRowPageBreakInterval

Syntax void Options.setRowPageBreakInterval(integer rowPageBreakInterval)

Sets the row page break interval.

Parameter **rowPageBreakInterval**
Integer. The row page break interval.

Example This example sets the row page break interval:

```
function setRowPBInterval(options,interval)(
   if (options){
      options.setRowPageBreakInterval(interval);
   }
}
```

Class actuate.xtabanalyzer.PageContent

Description A container for the content of a cross tab page. It contains a comprehensive list of report elements, such as tables, charts, labels, and data items.

Constructor

Syntax actuate.xtabanalyzer.PageContent( )

Creates a PageContent object that represents the report content that is generated by a report design file or document file.

Function summary

Table 15-16 lists actuate.xtabanalyzer.PageContent functions.

Table 15-16 actuate.xtabanalyzer.PageContent functions

Function	Description
getCrosstabByBookmark( )	Returns a report cross tab object
getViewerId( )	Returns the cross tab viewer ID

getCrosstabByBookmark

Syntax actuate.xtabanalyzer.crosstab PageContent.getCrosstabByBookmark(string bookmark)

Returns a cross tab object associated with a bookmark.

Parameter **bookmark**
The bookmark name of the item requested.

Returns actuate.xtabanalyzer.crosstab object.

Example This example retrieves the viewer ID, then retrieves the cross tab:

```
function getCrosstab( ){
   var viewer = PageContent.getViewerId( );
   var content = viewer.getCurrentPageContent( );
   var crosstab = content.getCrosstabByBookmark( );
   return crosstab;
}
```

getViewerId

Syntax string PageContent.getViewerId( )

Returns the XTabAnalyzer ID. The XTabAnalyzer is the cross tab viewer element.

Returns String. The XTabAnalyzer ID.

Example This example retrieves the viewer ID, then retrieves the cross tab:

```
function getCrosstab( ){
   var viewer = PageContent.getViewerId( );
   var content = viewer.getCurrentPageContent( );
   var crosstab = content.getCrosstabByBookmark( );
   return crosstab;
}
```

Class actuate.xtabanalyzer.ParameterValue

Description A container for the ParameterValue in the xtabanalyzer.

Constructor

Syntax actuate.xtabanalyzer.ParameterValue(string name, string value, boolean valueIsNull)

The ParameterValue class is used to specify a cross tab ParameterValue object.

Parameters

name
String. The parameter name.

value
String. The parameter value.

valueIsNull
Boolean. Whether the value is null.

Function summary

Table 15-17 lists actuate.xtabanalyzer.ParameterValue functions.

Table 15-17 actuate.xtabanalyzer.ParameterValue functions

Function	Description
getViewerId( )	Returns the parameter name
getName( )	Returns the parameter value
getValue( )	Returns whether the parameter has a null value
getValueIsNull( )	Sets the parameter name
setName( )	Sets the parameter value
setValue( )	Sets whether the parameter has a null value

getName

Syntax string ParameterValue.getName( )

Returns the name for the parameter.

Returns String. The parameter name.

Example This example retrieves the parameter name:

```
function getParameterName(parametervalue){
   if (parametervalue){
      return parametervalue.getName( );
   }
   return null;
}
```

getValue

Syntax String[ ] Dimension.getValue( )

Returns the name for the ParameterValue.

Returns String or array of strings. The parameter value or values.

Example This example retrieves the parameter value:

```
function getParameterValue(parametervalue){
   if (parametervalue){
      return parametervalue.getValue( );
   }
   return null;
}
```

getValueIsNull

Syntax boolean ParameterValue.getValueIsNull( )

Returns whether the parameter value is null.

Returns Boolean. True indicates the parameter value is null.

Example This example switches whether the parameter value is null:

```
if (parametervalue){
   if (parametervalue.getValueIsNull){
      parametervalue.setValueIsNull(false);
   } else {
      parametervalue.setValueIsNull(true);
   }
}
```

setName

Syntax void ParameterValue.setName(string name)

Sets the parameter name.

Parameter **name**
String. The parameter name.

Example This example sets the parameter name:

```
function setParameterName(parametervalue, name){
   parametervalue.setName(name);
}
```

setValue

Syntax void ParameterValue.setValue(string[ ] value)

Sets the parameter value.

Parameter **value**
String. The parameter value.

Example This example sets the parameter value:

```
function setParameterValue(parametervalue, value){
   parametervalue.setValue(value);
}
```

setValueIsNull

Syntax void ParameterValue.setValueIsNull(boolean valueIsNull)

Sets the valueIsNull for the ParameterValue.

Parameter **valueIsNull**
Boolean. True switches the value to null. False disables the null value setting.

Example This example switches whether the parameter value is null:

```
if (parametervalue){
   if (parametervalue.getValueIsNull){
      parametervalue.setValueIsNull(false);
   } else {
      parametervalue.setValueIsNull(true);
   }
}
```

Class actuate.xtabanalyzer.Sorter

Description Defines a sort condition used to sort on a dimension level or measure.

Constructor

Syntax actuate.xtabanalyzer.Sorter(string levelName)

Constructs a new sorter object.

Function summary

Table 15-18 lists actuate.xtabanalyzer.Sorter functions.

Table 15-18 actuate.xtabanalyzer.Sorter functions

Function	Description
getKey( )	Returns the sort key
getLevelName( )	Returns the level name
getMember( )	Returns the sort member
isAscending( )	Returns the sort direction
setAscending( )	Sets ascending or descending sort
setKey( )	Sets the sort key
setLevelName( )	Sets the level name
setMember( )	Sets the sort member

getKey

Syntax string Sorter.getKey( )

Returns the sort key. This is the name of the measure or dimension level to sort the cross tab on.

Returns String. The key to sort on.

Example This example retrieves the sort key:

```
function getSortKey(sorter){
   if (sorter){
      return sorter.getKey( );
   }
   return null;
}
```

getLevelName

Syntax string Sorter.getLevelName( )

Returns dimension level to sort on.

Returns String. The name of a dimension level.

Example This example retrieves the level name associated with the sorter:

```
function getSortLevel(sorter){
   if (sorter){
      return sorter.getLevelName( );
   }
   return null;
}
```

getMember

Syntax actuate.xtabanalyzer.MemberValue Sorter.getMember( )

Returns the member value to sort on.

Returns actuate.xtabanalyzer.MemberValue object. A member value.

Example This example retrieves the sort member:

```
function getSortMember(sorter){
   if (sorter){
      return sorter.getMember( );
   }
   return null;
}
```

isAscending

Syntax boolean Sorter.isAscending( )

Returns the sort order.

Returns Boolean. True when the sorter is ascending and false in all other cases.

Example This example retrieves the level name that is associated with the sorter:

```
function ascending(sorter){
   if (sorter){
      return sorter.isAscending( );
   }
   return null;
}
```

setAscending

Syntax void Sorter.setAscending(boolean ascending)

Sets the sort order to ascending or descending.

Parameter **ascending**
Boolean. Set to true for ascending, set to false for descending.

Example This example swaps the sort direction:

```
sorter.setAscending(!(sorter.isAscending));
```

setKey

Syntax void Sorter.setSortKey(string sortKey)

Sets the key to sort on.

Parameter **sortKey**
String. The sort key.

Example This example sets the sort key:

```
function setSortKey(sorter,key){
   sorter.setKey(key);
}
```

setLevelName

Syntax void Sorter.setLevelName(string levelName)

Sets the dimension level name to sort on.

Parameter **levelName**
String. A dimension level name.

Example This example sets the level name to sort:

```
function setSortLevel(sorter,level){
   sorter.setLevelName(level);
}
```

setMember

Syntax void Sorter.setMember(actuate.xtabanalyzer.MemberValue member)

Sets the member value to sort on.

Parameter **member**
actuate.xtabanalyzer.MemberValue object. A member value.

Example This example sets the sort member:

```
function setSortMember(sorter,member){
   sorter.setMember(member);
}
```

Class actuate.xtabanalyzer.SubTotal

Description A SubTotal object.

Constructor

Syntax actuate.xtabanalyzer.SubTotal( )

Constructs a new SubTotal object.

Function summary

Table 15-19 lists actuate.xtabanalyzer.SubTotal functions.

Table 15-19 actuate.xtabanalyzer.SubTotal functions

Function	Description
addTotal( )	Add a total
getLevelName( )	Returns the full level name
getLocation( )	Returns the location
getTotals( )	Returns the totals array
getType( )	Returns the type string
setLevelName( )	Sets the full level name
setLocation( )	Sets the location
setTotals( )	Sets the totals array

addTotal

Syntax void SubTotal.addTotal(actuate.xtabanalyzer.Total total)

Adds a total to the subtotal.

Parameter **total**
actuate.xtabanalyzer.Total. The total object being added.

Example This example uses addTotal( ) to create a subtotal:

```
function addSubTotal( ){
   var subTotal = new actuate.xtabanalyzer.SubTotal( );
   subTotal.setLevelName("year");
   subTotal.setLocation("after");
   var indexStr = "0;1;2;3;4";
   var indexs = indexsStr.split(";");
   var measureIndexs = [ ];
```

```
   for(var i = 0;i < indexs.length;i++){
      measureIndexs.push(parseInt(indexs[i]));
   }
   for( var i = 0; i < measureIndexs.length; i++){
      var total = new actuate.xtabanalyzer.Total( );
      total.setMeasureIndex(measureIndexs[i]);
      total.setAggregationFunction("SUM");
      total.setEnabled(true);
      subTotal.addTotal(total);
   }
      crosstab.setTotals(null,subTotal);
      crosstab.submit( );
}
```

getLevelName

Syntax string SubTotal.getLevelName( )

Returns the level for the subtotal.

Returns String. The level name for the subtotal.

Example This example retrieves the level name from the subtotal:

```
function getLevelName(subTotal){
   if (subTotal){
      return subTotal.getLevelName( );
   }
   return null;
}
```

getLocation

Syntax string SubTotal.getLocation( )

Returns the location name for the subtotal.

Returns String. The location name.

Example This example retrieves the level name from the subtotal:

```
function getLocation(subTotal){
   if (subTotal){
      return subTotal.getLocation( );
   }
   return null;
}
```

getTotals

Syntax object[ ] SubTotal.getTotals( )

Returns the totals used to calculate the subtotal.

Returns actuate.xtabanalyzer.Total[ ]. An array of total objects.

Example This example retrieves the totals from a SubTotal object:

```
var totalsArray = [ ];
function getTotals(subTotal,totalsArray){
   totalsArray = subTotal.getTotals( );
}
```

getType

Syntax string SubTotal.getType( )

Returns the type for the subtotal.

Returns String. The type for the subtotal.

Example This example retrieves the type from the subtotal:

```
function getLevelName(subTotal){
   if (subTotal){
      return subTotal.getType( );
   }
   return null;
}
```

setLevelName

Syntax void SubTotal.setLevelName(string levelName)

Sets the level for the subtotal by name.

Parameter **levelName**
String. The level name.

Example This example sets the level name for a subtotal:

```
function subTotalLevel(subTotal,levelName){
   if(subTotal){
      subTotal.setLevelName(levelName);
   }
}
```

setLocation

Syntax void SubTotal.setLocation(string location)

Sets the location for the subtotal.

Parameter **location**
String. The location. Value can be either before or after.

Example This example sets the location for a subtotal:

```
function subTotalLocation(subTotal,location){
   if(subTotal){
      subTotal.setLocation(location);
   }
}
```

setTotals

Syntax void SubTotal.setTotals(actuate.xtabanalyzer.Total[ ] totals)

Sets the totals using an array.

Parameter **totals**
Array of actuate.xtabanalyzer.Total objects to add to the subtotal.

Example This example uses setTotals( ) to create a subtotal:

```
function addSubTotal( ){
   var subTotal = new actuate.xtabanalyzer.SubTotal( );
   subTotal.setLevelName("year");
   subTotal.setLocation("after");
   var indexStr = "0;1;2;3;4";
   var indexs = indexsStr.split(";");
   var count = indexs.length;
   var measureIndexs = [ ];
   for(var i = 0;i < count;i++){
      measureIndexs.push(parseInt(indexs[i]));
   }
   var totals = Array(count);
   for( var i = 0; i < measureIndexs.length; i++){
      var total = new actuate.xtabanalyzer.Total( );
      total.setMeasureIndex( measureIndexs[i] );
      total.setAggregationFunction( "SUM" );
      total.setEnabled(true);
      totals[i] = total;
   }
   subTotal.setTotals(totals);

   crosstab.setTotals( null, subTotal );
   crosstab.submit( );
}
```

Class actuate.xtabanalyzer.Total

Description A container for a total in the xtabanalyzer. Total handles numeric aggregation functions for a measure.

Constructor

Syntax actuate.xtabanalyzer.Total( )

The Total class is used to specify a cross tab total object.

Function summary

Table 15-20 lists actuate.xtabanalyzer.Total functions.

Table 15-20 actuate.xtabanalyzer.Total functions

Function	Description
getAggregationFunction( )	Returns the aggregation function name
getMeasureIndex( )	Returns the measure index
isEnabled( )	Returns whether or not the total is enabled
setAggregationFunction( )	Sets the aggregation function name
setEnabled( )	Sets the enabled flag
setMeasureIndex( )	Sets the index for the total

getAggregationFunction

Syntax string Total.getAggregationFunction( )

Returns the aggregation function for the total.

Returns String. An aggregation function.

Example This example changes the aggregation function:

```
function swapTotalAggregation(total){
   if (total.getAggregationFunction( ) == "SUM"){
      total.setAggregationFunction("COUNT");
   } else {
      total.setAggregationFunction("SUM");
   }
}
```

getMeasureIndex

Syntax integer Dimension.getMeasureIndex( )

Retrieves the measure index for the total.

Returns Integer. The measure index.

Example This example retrieves the measure index:

```
function getMeasureIndex(total){
   if (total){
      return total.getIndex( );
   }
   return null;
}
```

isEnabled

Syntax boolean Total.isEnabled( )

Returns whether the total is enabled.

Returns Boolean. True indicates this total is enabled.

Example This example enables and disables a total:

```
if (total){
   if (total.isEnabled){
      total.setEnabled(false);
   } else {
      total.setEnabled(true);
   }
}
```

setAggregationFunction

Syntax void Total.setAggregationFunction(string aggregationFunction)

Sets the aggregation function name.

Parameter **aggregationFunction**
String. The aggregation function name.

Example This example changes the aggregation function:

```
function swapTotalAggregation(total){
   if (total.getAggregationFunction( ) == "SUM"){
      total.setAggregationFunction("COUNT");
   } else {
      total.setAggregationFunction("SUM");
   }
}
```

setEnabled

Syntax void Total.setEnabled(boolean enabled)

Sets whether total is enabled or disabled.

Parameter **enabled**
Boolean. True if the total is enabled. False for disabled.

Example This example enables and disables a total:

```
if (total){
   if (total.isEnabled){
      total.setEnabled(false);
   } else {
      total.setEnabled(true);
   }
}
```

setMeasureIndex

Syntax void Total.setMeasureIndex(integer measureIndex)

Sets the measure index for the total.

Parameter **measureIndex**
Integer. The measure index for the total.

Example This example uses setMeasureIndex( ) to create a subtotal:

```
function addSubTotal( ){
   var subTotal = new actuate.xtabanalyzer.SubTotal( );
   subTotal.setLevelName("year");
   subTotal.setLocation("after");
   var indexStr = "0;1;2;3;4";
   var indexs = indexsStr.split(";");
   var count = indexs.length;
   var measureIndexs = [];
   for(var i = 0;i < count;i++){
      measureIndexs.push(parseInt(indexs[i]));
   }
   for( var i = 0; i < measureIndexs.length; i++) {
      var total = new actuate.xtabanalyzer.Total( );
      total.setMeasureIndex(measureIndexs[i]);
      total.setAggregationFunction("SUM");
      total.setEnabled(true);
      subTotal.addTotal(total);
   }
   crosstab.setTotals(null,subTotal);
   crosstab.submit( );
}
```

Class actuate.xtabanalyzer.UIOptions

Description Specifies feature availability for the Interactive Crosstabs viewer.

Constructor

Syntax void actuate.xtabanalyzer.UIOptions( )

Generates a new UIOptions object to manage the features of the xtabanalyzer.

Function summary

Table 15-21 lists actuate.xtabanalyzer.UIOptions functions.

Table 15-21 actuate.xtabanalyzer.UIOptions functions

Function	Description
enableCrosstabView( )	Enables the cross tab layout view feature
enableCubeView( )	Enables the cube view feature
enableFilterSummaryView( )	Enables the filter summary view
enableToolBar( )	Enables the toolbar feature
enableToolbarHelp( )	Enables the toolbar help feature
enableToolbarSave( )	Enables the toolbar save feature
enableToolbarSaveDesign( )	Enables the toolbar save design feature
enableToolbarSaveDocument( )	Enables the toolbar save document feature
getFeatureMap( )	Returns a list of enabled and disabled features

enableCrosstabView

Syntax void UIOptions.enableCrosstabView(boolean enabled)

Enables or disables the cross tab layout view.

Parameter **enabled**
Boolean. True enables this option.

Example This example enables or disables the cross tab view:

```
function setCrosstabView(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableCrosstabView(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableCubeView

Syntax void UIOptions.enableCubeView(boolean enabled)

Enables or disables the cube view.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables the cube view:

```
function setCubeView(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableCubeView(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableFilterSummaryView

Syntax void UIOptions.enableFilterSummaryView(boolean enabled)

Enables or disables the filter summary view.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables the filter summary view:

```
function setFilterSummary(flag) {
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableFilterSummaryView(enabled);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableToolBar

Syntax void UIOptions.enableToolBar(boolean enabled)

Enables or disables the toolbar feature.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables the toolbar:

```
function setToolbar(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableToolBar(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableToolbarHelp

Syntax void UIOptions.enableToolbarHelp(boolean enabled)

Enables or disables the toolbar help feature.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables toolbar help:

```
function setToolbarHelp(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableToolbarHelp(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableToolbarSave

Syntax void UIOptions.enableToolbarSave(boolean enabled)

Enables or disables the toolbar save feature.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables toolbar save:

```
function setToolbarSave(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableToolbarSave(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableToolbarSaveDesign

Syntax void UIOptions.enableToolbarSaveDesign(boolean enabled)

Enables or disables the toolbar save design feature.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables toolbar save design:

```
function setToolbarSave(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableToolbarSaveDesign(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

enableToolbarSaveDocument

Syntax void UIOptions.enableToolbarSaveDocument(boolean enabled)

Enables or disables the toolbar save document feature.

Parameter **enabled**
Boolean. A value of true enables this option.

Example This example enables or disables toolbar save document:

```
function setToolbarSave(flag){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   uiOptions.enableToolbarSaveDocument(flag);
   myXTabAnalyzer.setUIOptions(uiOptions);
}
```

getFeatureMap

Syntax Object UIOptions.getFeatureMap( )

Returns the features and their Boolean values as an associative array. This function makes the name of each feature an object property and sets the value of that property to the associated enabled Boolean value.

Returns Object. An associative array of string name and Boolean value pairs.

Example This example retrieves the feature map:

```
function retrieveFeatureMap( ){
   var uiOptions = new actuate.xtabanalyzer.UIOptions( );
   var features = uiOptions.getFeatureMap( );
   return features;
}
```

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z