

# ActuateOne™

One Design  
One Server  
One User Experience

**Using Actuate JavaScript API**

Information in this document is subject to change without notice. Examples provided are fictitious. No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, for any purpose, in whole or in part, without the express written permission of Actuate Corporation.

© 1995 - 2013 by Actuate Corporation. All rights reserved. Printed in the United States of America.

Contains information proprietary to:

Actuate Corporation, 951 Mariners Island Boulevard, San Mateo, CA 94404

[www.actuate.com](http://www.actuate.com)

The software described in this manual is provided by Actuate Corporation under an Actuate License agreement. The software may be used only in accordance with the terms of the agreement. Actuate software products are protected by U.S. and International patents and patents pending. For a current list of patents, please see <http://www.actuate.com/patents>.

Actuate Corporation trademarks and registered trademarks include:

Actuate, ActuateOne, the Actuate logo, Archived Data Analytics, BIRT, BIRT 360, BIRT Analytics, The BIRT Company, BIRT Data Analyzer, BIRT iHub, BIRT Performance Analytics, Collaborative Reporting Architecture, e.Analysis, e.Report, e.Reporting, e.Spreadsheet, Encyclopedia, Interactive Viewing, OnPerformance, The people behind BIRT, Performancesoft, Performancesoft Track, Performancesoft Views, Report Encyclopedia, Reportlet, X2BIRT, and XML reports.

Actuate products may contain third-party products or technologies. Third-party trademarks or registered trademarks of their respective owners, companies, or organizations include:

Mark Adler and Jean-loup Gailly ([www.zlib.net](http://www.zlib.net)): zlib. Adobe Systems Incorporated: Flash Player. Amazon Web Services, Incorporated: Amazon Web Services SDK, licensed under the Apache Public License (APL). Apache Software Foundation ([www.apache.org](http://www.apache.org)): Ant, Axis, Axis2, Batik, Batik SVG library, Commons Command Line Interface (CLI), Commons Codec, Crimson, Derby, Hive driver for Hadoop, Pluto, Portals, Shindig, Struts, Tomcat, Xalan, Xerces, Xerces2 Java Parser, and Xerces-C++ XML Parser. Castor ([www.castor.org](http://www.castor.org)), ExoLab Project ([www.exolab.org](http://www.exolab.org)), and Intalio, Inc. ([www.intalio.org](http://www.intalio.org)): Castor. Day Management AG: Content Repository for Java. Eclipse Foundation, Inc. ([www.eclipse.org](http://www.eclipse.org)): Babel, Data Tools Platform (DTP) ODA, Eclipse SDK, Graphics Editor Framework (GEF), Eclipse Modeling Framework (EMF), and Eclipse Web Tools Platform (WTP), licensed under the Eclipse Public License (EPL). Gargoyle Software Inc.: HtmlUnit, licensed under Apache License Version 2.0. GNU Project: GNU Regular Expression, licensed under the GNU Lesser General Public License (LGPLv3). HighSlide: HighCharts. Jason Hsueh and Kenton Varda ([code.google.com](http://code.google.com)): Protocol Buffer. IDAutomation.com, Inc.: IDAutomation. IDRsolutions Ltd.: JBIG2, licensed under the BSD license. InfoSoft Global (P) Ltd.: FusionCharts, FusionMaps, FusionWidgets, PowerCharts. Matt Inger ([sourceforge.net](http://sourceforge.net)): Ant-Contrib, licensed under Apache License Version 2.0. Matt Ingenthron, Eric D. Lambert, and Dustin Sallings ([code.google.com](http://code.google.com)): Spymemcached, licensed under the MIT OSI License. International Components for Unicode (ICU): ICU library. jQuery: jQuery, licensed under the MIT License. Yuri Kanivets ([code.google.com](http://code.google.com)): Android Wheel gadget, licensed under the Apache Public License (APL). LEAD Technologies, Inc.: LEADTOOLS. The Legion of the Bouncy Castle: Bouncy Castle Crypto APIs. Bruno Lowagie and Paulo Soares: iText, licensed under the Mozilla Public License (MPL). Microsoft Corporation (Microsoft Developer Network): CompoundDocument Library. Mozilla: Mozilla XML Parser, licensed under the Mozilla Public License (MPL). MySQL Americas, Inc.: MySQL Connector. Netscape Communications Corporation, Inc.: Rhino, licensed under the Netscape Public License (NPL). OOPS Consultancy: XMLTask, licensed under the Apache License, Version 2.0. Oracle Corporation: Berkeley DB, Java Advanced Imaging, JAXB, JDK, Jstl. PostgreSQL Global Development Group: pgAdmin, PostgreSQL, PostgreSQL JDBC driver. Progress Software Corporation: DataDirect Connect XE for JDBC Salesforce, DataDirect JDBC, DataDirect ODBC. Rogue Wave Software, Inc.: Rogue Wave Library SourcePro Core, tools.h++. Sam Stephenson ([prototype.conio.net](http://prototype.conio.net)): prototype.js, licensed under the MIT license. Sencha Inc.: Ext JS, Sencha Touch. ThimbleWare, Inc.: JMemcached, licensed under the Apache Public License (APL). World Wide Web Consortium (W3C)(MIT, ERCIM, Keio): Flute, JTIty, Simple API for CSS. XFree86 Project, Inc.: ([www.xfree86.org](http://www.xfree86.org)): xvfb. ZXing authors ([code.google.com](http://code.google.com)): ZXing, licensed under the Apache Public License (APL).

All other brand or product names are trademarks or registered trademarks of their respective owners, companies, or organizations.

Document No. 130131-2-791301 January 23, 2013

# Contents

<b>About Using Actuate JavaScript API. . . . .</b>	<b>v</b>
--	----------

## Chapter 1

<b>Creating a custom web page using the Actuate JavaScript API . . . . .</b>	<b>1</b>
--	----------

About the Actuate JavaScript API . . . . .	2
Accessing the Actuate JavaScript API . . . . .	2
About the DOCTYPE tag . . . . .	3
About UTF8 character encoding . . . . .	3
Establishing an HTTP session with an Actuate web application . . . . .	4
About Actuate JavaScript API security integration . . . . .	4
Establishing a secure connection to more than one web service . . . . .	5
Using a login servlet to connect to an Actuate web application . . . . .	6
Using a custom servlet to connect to an Actuate web application . . . . .	7
Unloading authentication information from the session . . . . .	7
Viewing reports . . . . .	8
Controlling viewer user interface features . . . . .	10
Accessing report content . . . . .	10
Accessing HTML5 Chart features . . . . .	11
Using a filter . . . . .	11
Using a sorter . . . . .	12
Navigating repository content using ReportExplorer . . . . .	12
Displaying ReportExplorer . . . . .	13
Opening files from ReportExplorer . . . . .	15
Using dashboards and gadgets . . . . .	17
Using and submitting report parameters . . . . .	19
Using a parameter component . . . . .	19
Accessing parameter values from the viewer . . . . .	20
Retrieving report content as data . . . . .	22
Using a data service component . . . . .	23
Using a result set component . . . . .	24

## Chapter 2

<b>Creating dynamic report content using the Actuate JavaScript API . . . .</b>	<b>25</b>
---	-----------

About Actuate JavaScript API scripting in a BIRT report design . . . . .	26
Using the Actuate JavaScript API in an HTML button . . . . .	27
Using the Actuate JavaScript API in chart interactive features . . . . .	28
Using the Actuate JavaScript API in chart themes . . . . .	32
Using the Actuate JavaScript API in the BIRT script editor . . . . .	33

Chapter 3	
<b>Working with BIRT Data Analyzer and cross tabs</b>	<b>35</b>
About cross tabs	36
About cubes	37
Handling Data Analyzer viewer events	38
Working with dimensions, measures, and levels	39
Adding a dimension with levels	39
Removing a dimension	40
Adding and removing measures	40
Changing measures and dimensions	41
Working with totals	42
Sorting and filtering cross tab data	43
Drilling down within a cross tab	43
Controlling the Data Analyzer viewer user interface	44
Chapter 4	
<b>Actuate JavaScript API classes</b>	<b>47</b>
Actuate JavaScript API overview	48
About the actuate namespace	48
Using the Actuate library	48
Actuate JavaScript API classes quick reference	49
Actuate JavaScript API reference	51
Class <b>actuate</b>	52
Class <b>actuate.AuthenticationException</b>	59
Class <b>actuate.ConnectionException</b>	61
Class <b>actuate.Dashboard</b>	62
Class <b>actuate.dashboard.DashboardDefinition</b>	72
Class <b>actuate.dashboard.EventConstants</b>	73
Class <b>actuate.dashboard.GadgetScript</b>	74
Class <b>actuate.dashboard.Tab</b>	77
Class <b>actuate.data.Filter</b>	79
Class <b>actuate.data.ReportContent</b>	84
Class <b>actuate.data.Request</b>	85
Class <b>actuate.data.ResultSet</b>	90
Class <b>actuate.data.Sorter</b>	92
Class <b>actuate.DataService</b>	95
Class <b>actuate.Exception</b>	97
Class <b>actuate.Parameter</b>	100
Class <b>actuate.parameter.Constants</b>	111
Class <b>actuate.parameter.ConvertUtility</b>	112
Class <b>actuate.parameter.EventConstants</b>	115
Class <b>actuate.parameter.NameValuePair</b>	116
Class <b>actuate.parameter.ParameterData</b>	118

Class <code>actuate.parameter.ParameterDefinition</code> .....	126
Class <code>actuate.parameter.ParameterValue</code> .....	143
Class <code>actuate.report.Chart</code> .....	151
Class <code>actuate.report.DataItem</code> .....	159
Class <code>actuate.report.FlashObject</code> .....	163
Class <code>actuate.report.Gadget</code> .....	167
Class <code>actuate.report.HTML5Chart.ClientChart</code> .....	172
Class <code>actuate.report.HTML5Chart.ClientOption</code> .....	179
Class <code>actuate.report.HTML5Chart.ClientPoint</code> .....	183
Class <code>actuate.report.HTML5Chart.ClientSeries</code> .....	186
Class <code>actuate.report.HTML5Chart.Highcharts</code> .....	191
Class <code>actuate.report.HTML5Chart.Renderer</code> .....	192
Class <code>actuate.report.Label</code> .....	198
Class <code>actuate.report.Table</code> .....	202
Class <code>actuate.report.TextItem</code> .....	210
Class <code>actuate.ReportExplorer</code> .....	214
Class <code>actuate.reportexplorer.Constants</code> .....	221
Class <code>actuate.reportexplorer.EventConstants</code> .....	222
Class <code>actuate.reportexplorer.File</code> .....	223
Class <code>actuate.reportexplorer.FileCondition</code> .....	231
Class <code>actuate.reportexplorer.FileSearch</code> .....	233
Class <code>actuate.reportexplorer.FolderItems</code> .....	242
Class <code>actuate.reportexplorer.PrivilegeFilter</code> .....	244
Class <code>actuate.RequestOptions</code> .....	248
Class <code>actuate.Viewer</code> .....	252
Class <code>actuate.viewer.BrowserPanel</code> .....	272
Class <code>actuate.viewer.EventConstants</code> .....	273
Class <code>actuate.viewer.PageContent</code> .....	274
Class <code>actuate.viewer.ParameterValue</code> .....	278
Class <code>actuate.viewer.RenderOptions</code> .....	280
Class <code>actuate.viewer.ScrollPanel</code> .....	282
Class <code>actuate.viewer.SelectedContent</code> .....	285
Class <code>actuate.viewer.UIConfig</code> .....	287
Class <code>actuate.viewer.UIOptions</code> .....	289
Class <code>actuate.viewer.ViewerException</code> .....	303

## Chapter 5

<b>BIRT Data Analyzer API classes</b> .....	<b>305</b>
About the BIRT Data Analyzer JavaScript API .....	306
Data Analyzer API reference .....	307
Data Analyzer JavaScript classes quick reference .....	309
Class <code>actuate.XTabAnalyzer</code> .....	310
Class <code>actuate.xtabanalyzer.Crosstab</code> .....	327

Class actuate.xtabanalyzer.Dimension .....	341
Class actuate.xtabanalyzer.Driller .....	347
Class actuate.xtabanalyzer.EventConstants .....	350
Class actuate.xtabanalyzer.Exception .....	351
Class actuate.xtabanalyzer.Filter .....	354
Class actuate.xtabanalyzer.GrandTotal .....	361
Class actuate.xtabanalyzer.Level .....	364
Class actuate.xtabanalyzer.LevelAttribute .....	367
Class actuate.xtabanalyzer.Measure .....	368
Class actuate.xtabanalyzer.MemberValue .....	373
Class actuate.xtabanalyzer.Options .....	376
Class actuate.xtabanalyzer.PageContent .....	383
Class actuate.xtabanalyzer.ParameterValue .....	385
Class actuate.xtabanalyzer.Sorter .....	388
Class actuate.xtabanalyzer.SubTotal .....	392
Class actuate.xtabanalyzer.Total .....	396
Class actuate.xtabanalyzer.UIOptions .....	399
<b>Index .....</b>	<b>403</b>

# A b o u t   U s i n g   A c t u a t e J a v a S c r i p t   A P I

---

*Using Actuate JavaScript API* is a guide to designing custom reporting web applications with the Actuate JavaScript API.

*Using Actuate JavaScript API* includes the following chapters:

- *About Using Actuate JavaScript API.* This chapter provides an overview of this guide.
- *Chapter 1. Creating a custom web page using the Actuate JavaScript API.* This chapter describes the Actuate JavaScript API requirements and common implementations.
- *Chapter 2. Creating dynamic report content using the Actuate JavaScript API.* This chapter describes using Actuate JavaScript API code in a BIRT report.
- *Chapter 3. Working with BIRT Data Analyzer and cross tabs.* This chapter describes the BIRT Data Analyzer and the use of cross tabs.
- *Chapter 4. Actuate JavaScript API classes.* This chapter lists all of the standard Actuate JavaScript API classes and their methods.
- *Chapter 5. BIRT Data Analyzer API classes.* This chapter lists all of the cross tab classes and their methods.





# Creating a custom web page using the Actuate JavaScript API

This chapter contains the following topics:

- 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 dashboards and gadgets
- Using and submitting report parameters
- Retrieving report content as data

---

## 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 the BIRT Viewer and Data Analyzer.

To use the Actuate JavaScript API, connect to Actuate Information Console or Deployment Kit for BIRT Reports.

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

```
<Context Root>\portal\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://localhost:8700/iportal/jsapi"></script>
    <script type="text/javascript" language="JavaScript">
      ... <!--functionality goes here-->
    </script>
  </div>
</body>
</html>

```

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

```

<script type="text/javascript" src="http://localhost:8700
  /iportal/jsapi">
</script>

```

where

- localhost:8700 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 `<div>` 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://localhost:8700/iportal", null, null,
    null, runReport, null);
```

where

- `http://localhost:8700/iportal` is a URL for the Actuate report application service. This URL must correspond to an Actuate Deployment Kit for BIRT Report application or Information Console 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 URL parameter.
- 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 `<div>` 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:

- **Deployment Kit 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. See *Actuate BIRT Java Components Developer Guide* for information about customizing security for Actuate Deployment Kit.
- **Deployment Kit using an Encyclopedia volume repository:** Encyclopedia volumes are managed by Actuate BIRT iHub. To connect to a Deployment Kit that accesses an Encyclopedia 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 *Actuate BIRT Java Components Developer Guide* for information about customizing security for Actuate Deployment Kit.
- **Information Console:** Actuate Information Console connects to an Encyclopedia volume and requires authentication. To connect to an 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. Information Console provides a login page to establish the secure session. See *Information Console Developer Guide* for information about customizing security for Actuate 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 Deployment Kit for BIRT Reports or Information Console 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/loginervlet?" 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 loginervlet.

## 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 BIRT Java Components Developer Guide* or *Information Console Developer 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 Deployment Kit for BIRT Reports or Information Console 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 `<div>` 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 `<div>` element which holds the report content. The page body must contain a `<div>` 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.rptdesign"` indicates the Customer List report design 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 `<div>` 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://localhost:8700/iportal/jsapi"></script>

  <script type="text/javascript" language="JavaScript">
function init( ){
  actuate.load("viewer");
  actuate.initialize( "http://localhost:8700/iportal", null,
    null, null, runReport);
} function runReport( ) {
  var viewer = new actuate.Viewer("viewerpane");
  viewer.setReportName("/Public/BIRT and BIRT Studio
    Examples/Top 5 Sales Performers.rptdesign");
  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");
  actuate.initialize( "http://localhost:8700/iportal", null,
    null, null, runReport);
  function runReport( ) {
    var viewer = new actuate.Viewer("viewerpane");
    viewer.setReportName("/Public/BIRT and BIRT Studio
      Examples/Top 5 Sales Performers.rptdesign");
    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 1-1.



**Figure 1-1** 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 Encyclopedia 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 `<div>` 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://localhost:8900/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 `<div>` element which holds the report explorer content. The page body must contain a `<div>` 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 `<div>` element.

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

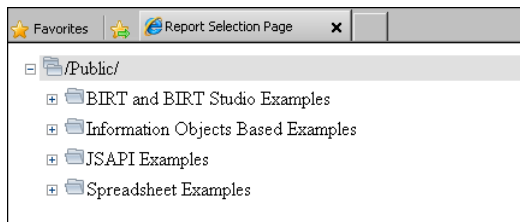
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="content-type" content="text
/html; charset=utf-8" />
  <title>Report Explorer Page</title>
</head>
<body onload="init( )">
<div id="explorerpane">
  <script type="text/javascript" language="JavaScript"
    src="http://localhost:8700/iportal/jsapi"></script>

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

function init( ) {
  actuate.load("reportexplorer");
  requestOpts = new actuate.RequestOptions( );
  actuate.initialize( "http://localhost:8900/iportal",
    requestOpts, 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 1-2.



**Figure 1-2** 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://localhost:8700/iportal/jsapi"></script>

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

var file = "unknown";

function init( ) {
    actuate.load("reportexplorer");
    actuate.load("viewer");
    actuate.load("dialog");
    requestOpts = new actuate.RequestOptions( );
    actuate.initialize( "http://localhost:8900/iportal",
        requestOpts, 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 dashboards and gadgets

The `actuate.Dashboard` class loads and displays dashboards and provides access to the gadgets contained in dashboards. To use `actuate.Dashboard`, load the class with `actuate.load( )`, as shown in the following code:

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

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

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

Load the dashboard and dialog components to use the dashboard on the page. Call the `actuate.Dashboard` functions to prepare a dashboard and call the dashboard's `submit( )` function to display the contents in the assigned `<div>` element.

The `actuate.Dashboard` class is a container for a dashboard file. Create an instance of the class using JavaScript, as shown in the following code:

```
dashboard = new actuate.Dashboard("containerID");
```

The value of `"containerID"` is the name value for the `<div>` element that displays the dashboard content. The page body must contain a `<div>` element with the `containerID` id, as shown in the following code:

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

To set the dashboard file to display, use `setDashboardName( )` as shown in the following code:

```
dashboard.setDashboardName("/sharedtab.dashboard");
```

The `setReportName( )` function accepts 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 display the dashboard, call `dashboard.submit( )` as shown in the following code:

```
dashboard.submit(submitCallback);
```

The `submit( )` function submits all of the asynchronous operations that previous viewer functions prepare and triggers an AJAX request for the dashboard. The Actuate web application returns the dashboard and the page displays the dashboard in the assigned `<div>` element. The `submitCallback( )` callback function triggers after the submit operation completes.

This is a complete example that displays a dashboard:

```
<!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>Dashboard Page</title>
</head>
<body onload="init( )">
<div id="dashboardpane">
    <script type="text/javascript" language="JavaScript"
        src="http://localhost:8700/iportal/jsapi"></script>

    <script type="text/javascript" language="JavaScript">
function init( ){
    actuate.load("dashboard");
    actuate.initialize( "http://localhost:8900/iportal", null,
        null, null, runDashboard);
} function runDashboard( ){
    var dash = new actuate.Dashboard("dashboardpane");
    dash.setDashboardName("/Dashboard/Contents
        /Documents.DASHBOARD");
    dash.setIsDesigner(false);
    dash.submit( );
}
    </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 `<div>` 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 `<div>` element that holds the report parameters display. The page body must contain a `<div>` 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 <div> 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://localhost:8700/iportal/jsapi"></script>

  <script type="text/javascript" language="JavaScript">
function init( ){
  actuate.load("viewer");
  actuate.load("parameter");
  actuate.initialize( "http://localhost:8900/iportal", null,
    null, null, displayParams);
}
function displayParams( ) {
  param = new actuate.Parameter("param1");
  param.setReportName("/Public/BIRT and BIRT Studio
    Examples/Custom 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/Custom 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://localhost:8700/iportal/jsapi"></script>
        <script type="text/javascript" language="JavaScript">
            function init( ){
                actuate.load("viewer");
                actuate.load("parameter");
                actuate.initialize( "http://localhost:8700/iportal", null,
                    null, null, displayParams);
            }
            function displayParams( ) {
                param = new actuate.Parameter("parampane");
                param.setReportName("/Public/BIRT and BIRT Studio
                    Examples/Custom Order History.rptdesign");
                param.submit(
                    function ( ) {this.run.style.visibility = 'visible';});
            }
            function processParameters( ) {
                param.downloadParameterValues(runReport);
            }
        </script>
    </div>
</body>
</html>
```

```

</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://localhost:8700/portal/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/Custom 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 Information Console. The default report is at the following path:

```

/Public/BIRT and BIRT Studio Examples/Custom 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.initialize`.

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.



# Creating dynamic report content using the Actuate JavaScript API

This chapter contains the following topics:

- 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
- Using the Actuate JavaScript API in the BIRT script editor

---

## 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:

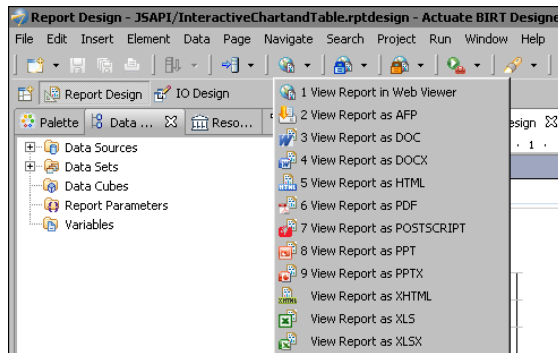
- Using the Actuate JavaScript API in an HTML button
- Using the Actuate JavaScript API in chart interactive features
- Using the Actuate JavaScript API in the BIRT script editor

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 2-1.

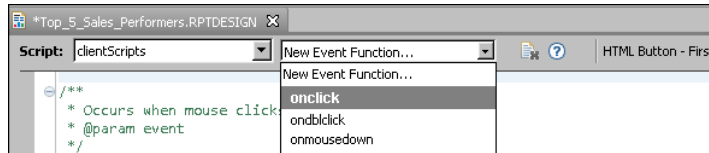


**Figure 2-1** Accessing Web Viewer in Actuate BIRT Designer

Most of the classes and functions in the `actuate.Viewer` class can be used in a BIRT report without loading or initializing the `actuate.Viewer` class and dialog support. 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 2-2.

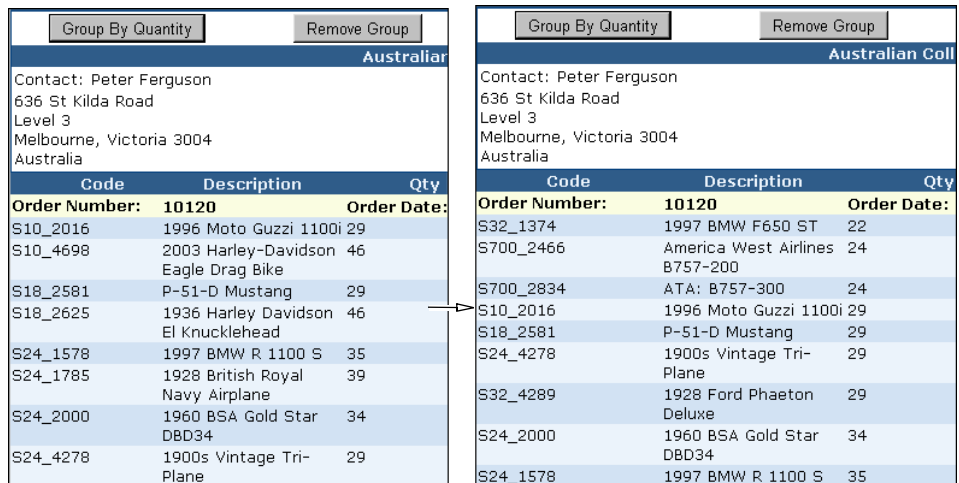


**Figure 2-2** Choosing HTML button event handlers

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 2-3.

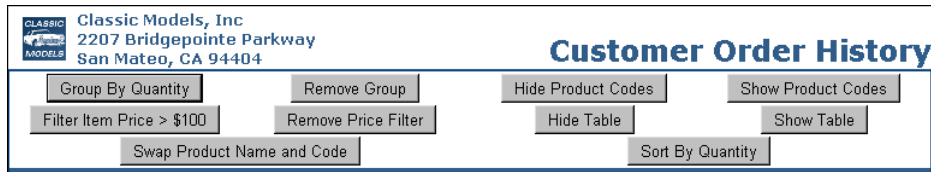


Code	Description	Qty
Order Number: 10120	Order Date:	
S10_2016	1996 Moto Guzzi 1100i	29
S10_4698	2003 Harley-Davidson Eagle Drag Bike	46
S18_2581	P-51-D Mustang	29
S18_2625	1936 Harley Davidson El Knucklehead	46
S24_1578	1997 BMW R 1100 S	35
S24_1785	1928 British Royal Navy Airplane	39
S24_2000	1960 BSA Gold Star DBD34	34
S24_4278	1900s Vintage Tri-Plane	29

Code	Description	Qty
Order Number: 10120	Order Date:	
S32_1374	1997 BMW F650 ST	22
S700_2466	America West Airlines B757-200	24
S700_2834	ATA: B757-300	24
S10_2016	1996 Moto Guzzi 1100i	29
S18_2581	P-51-D Mustang	29
S24_4278	1900s Vintage Tri-Plane	29
S32_4289	1928 Ford Phaeton Deluxe	29
S24_2000	1960 BSA Gold Star DBD34	34
S24_1578	1997 BMW R 1100 S	35

**Figure 2-3** Using a GroupBy HTMLButton control

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 2-4.

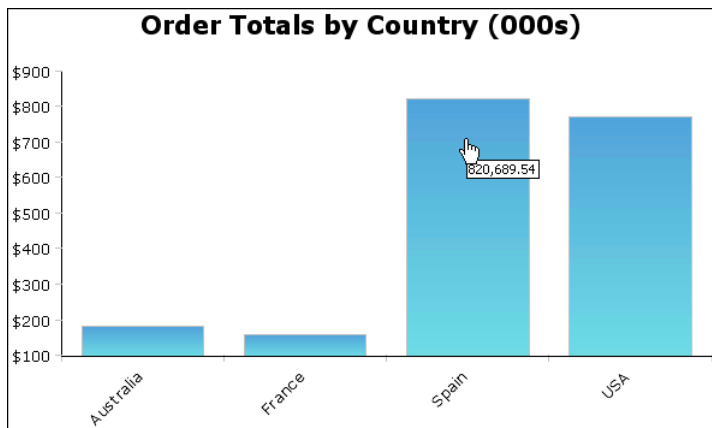


**Figure 2-4** HTML button header

## Using the Actuate JavaScript API in chart interactive features

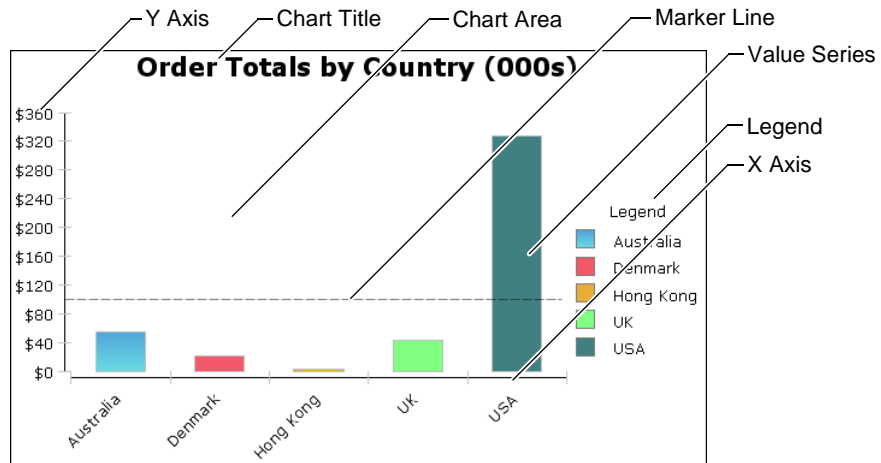
BIRT reports support adding interactive features to a chart to enhance the behavior of a chart in the viewer. The interactive chart features are available through the chart builder. Implement Actuate JavaScript API functions within interactive features.

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 2-5.



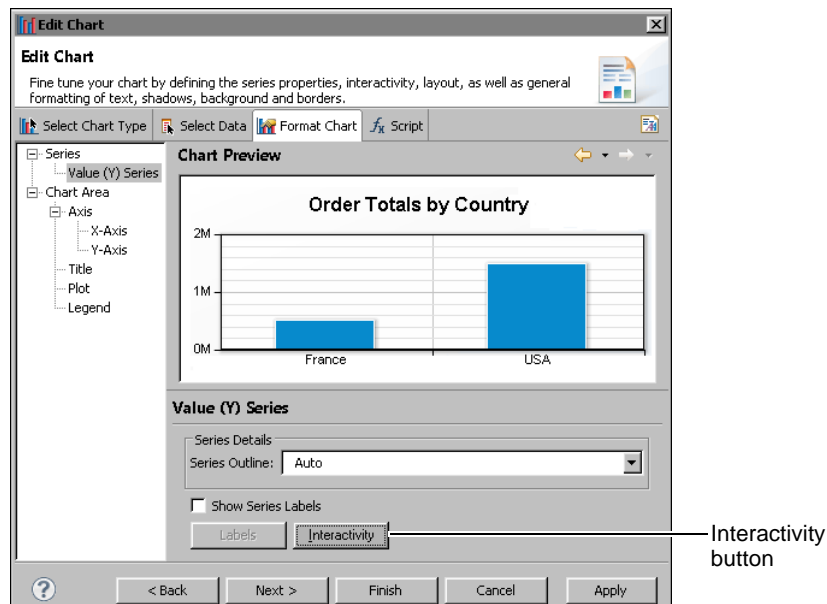
**Figure 2-5** Chart showing a tooltip

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 2-6 identifies these elements.



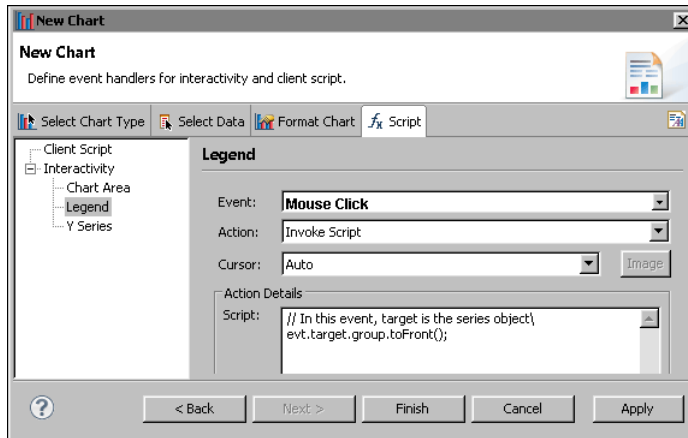
**Figure 2-6** Elements selectable for chart interactivity

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 2-7 shows the location of the Interactivity button for a value series.



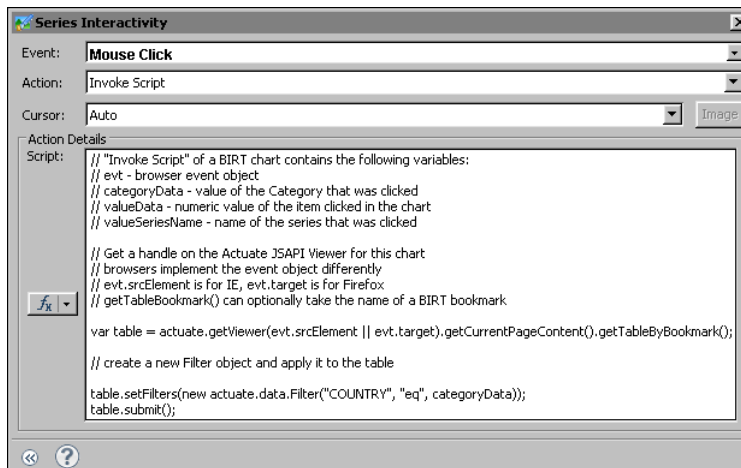
**Figure 2-7** Accessing interactivity for a value series

Figure 2-8 shows the elements accessible using the script feature.



**Figure 2-8** 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 2-9 shows the interactivity editor.



**Figure 2-9** 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 table = actuate.getViewer(evt.srcElement ||
    evt.target).getCurrentPageContent( ).getTableByBookmark( );
```

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.target` contains the name of the viewer object. For Internet Explorer, `evt.srcElement` contains the name of the viewer object.

The `getCurrentPageContent.getTableByBookmark()` 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:

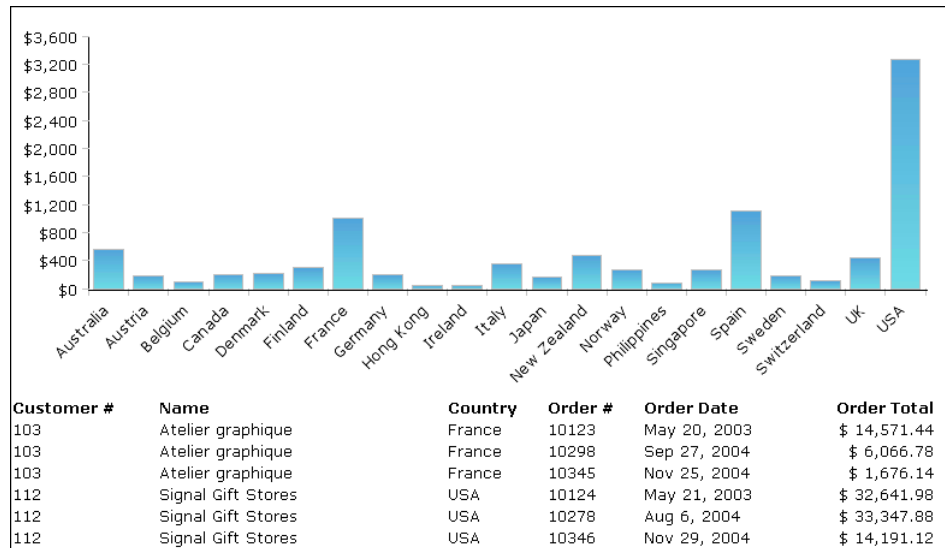
- **column name**: The column name is the name of the series. In this case, the *y*-axis is a list of countries, so a mouse click filters the table according to the `COUNTRY` column.
  - **operator**: `eq` is the reserved operator for equal to.
  - **value**: the value of the `categoryData` object generated 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 2-10 shows the chart before interaction.

When the user selects the bar for Australia in the value series, the table is filtered for Australia, as shown in Figure 2-11.



**Figure 2-10** An interactive chart and table before any user action



**Figure 2-11** An interactive chart and table after the user selects Australia

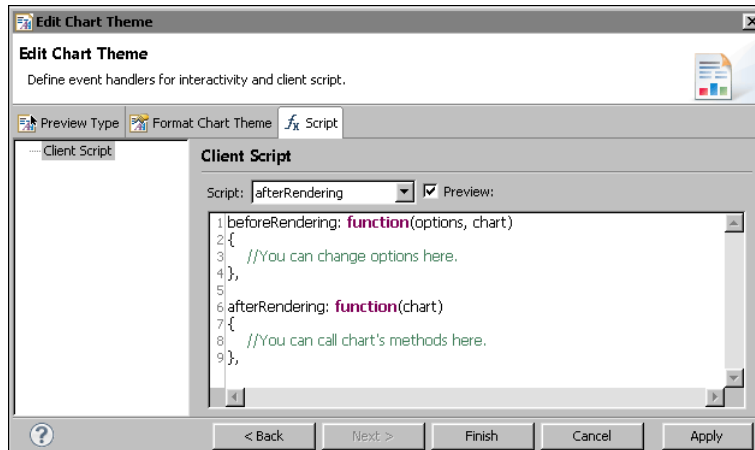
## Using the Actuate JavaScript API in chart themes

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 2-12.



**Figure 2-12** Adding script elements in edit chart theme

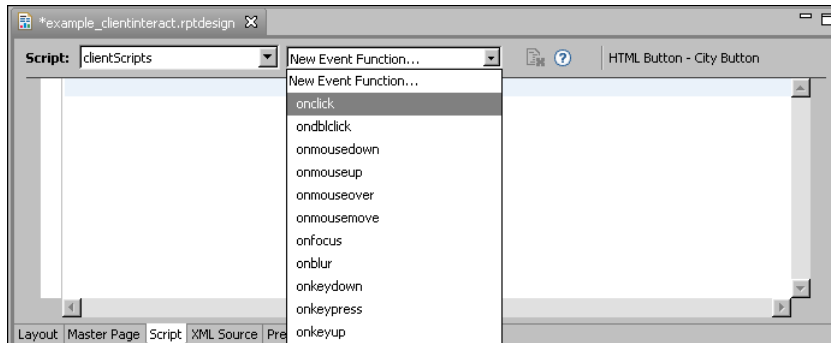
In an HTML5 chart, you can use the `actuate.report.HTML5Chart` classes to alter the report display. For example, to render every data point in the series that is greater than `avgValue` in a green color, use code similar to the following:

```
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';
        }
    }
}
```

---

## Using the Actuate JavaScript API in the BIRT script editor

Use the Script tab to open the BIRT script editor in Actuate BIRT Designer Professional, as shown in Figure 2-13.



**Figure 2-13** The BIRT Report Designer Professional script editor  
Assign a Actuate JavaScript API script to run when a particular event occurs, such as onRender.

## **Working with BIRT Data Analyzer and cross tabs**

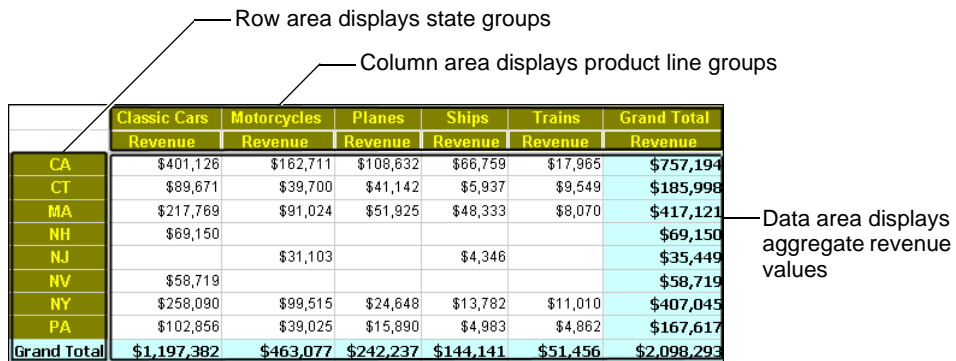
This chapter contains the following topics:

- About cross tabs
- About cubes
- Handling Data Analyzer 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 Data Analyzer 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.

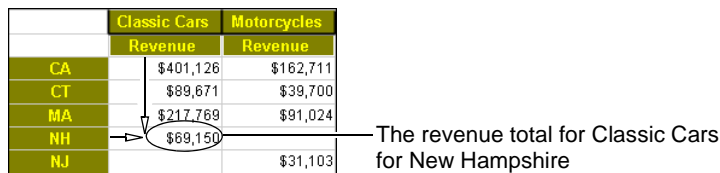
Figure 3-1 shows a cross tab that organizes state groups in the row area and product line groups in the column area. Aggregate revenue values appear in the cells of the data area.



	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

**Figure 3-1** Viewing a cross tab

A cell displays a revenue value by product line and by state, as shown in Figure 3-2.



	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

**Figure 3-2** A cell displaying a revenue total

A cross tab uses data from at least three fields. The cross tab in Figure 3-1 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 3-1, 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 3-1, the cross tab displays eight unique values from the state field: CA, CT, MA, NH, NJ, NV, NY, and PA.

- BIRT Data Analyzer 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. Data Analyzer calculates the revenue total using the SUM function on the values in the extendedprice field.

---

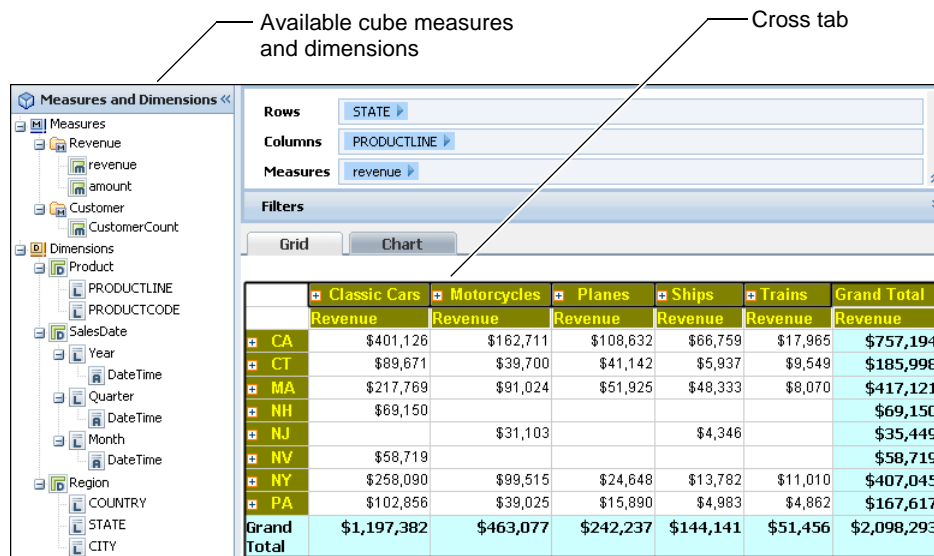
## About cubes

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 the Data Analyzer typically displays a portion of the available cube data in a simple layout. Figure 3-3 shows a cross tab and all of the cube measures and dimensions that are available for analysis.



**Figure 3-3** Data Analyzer displaying a cross tab and available measures and dimensions

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

## Handling Data Analyzer viewer events

The Data Analyzer 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 Data Analyzer 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. Data Analyzer 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 salesdate 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( );
```

---

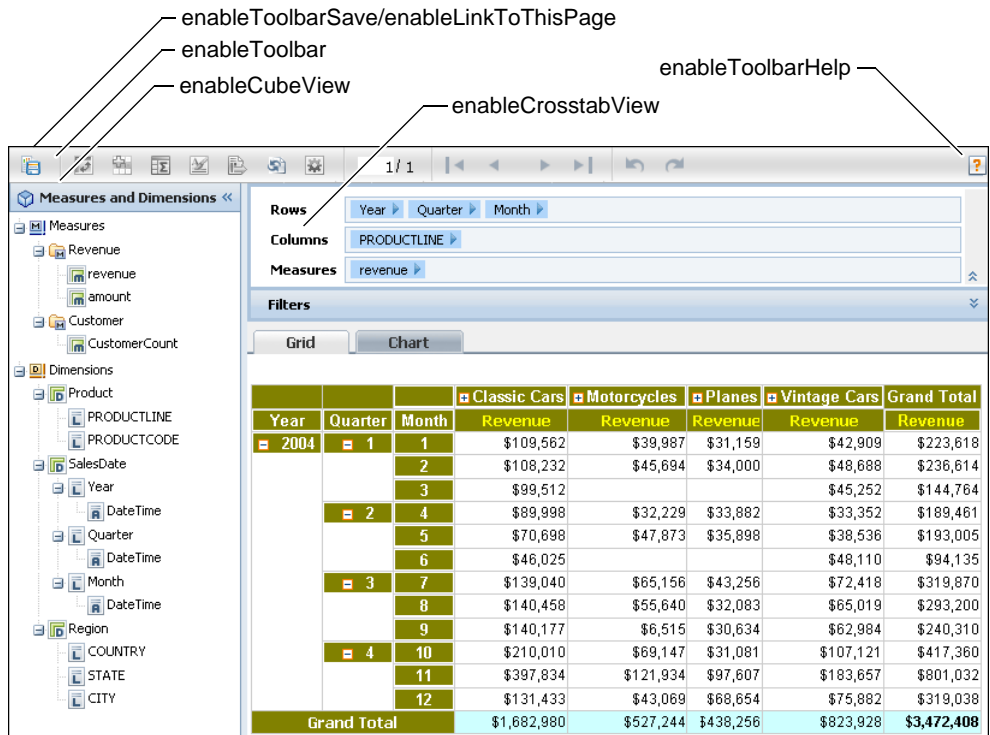
## Controlling the Data Analyzer viewer user interface

Show or hide Data Analyzer 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 3-4 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 3-4** Data Analyzer viewer showing areas altered with UIOptions

This code produces a viewer similar to Figure 3-5.

			Classic Cars	Motorcycles	Planes	Vintage Cars	Grand Total
Year	Quarter	Month	Revenue	Revenue	Revenue	Revenue	Revenue
2004	1	1	\$109,562	\$39,987	\$31,159	\$42,909	\$223,618
		2	\$108,232	\$45,694	\$34,000	\$48,688	\$236,614
		3	\$99,512			\$45,252	\$144,764
	2	4	\$89,998	\$32,229	\$33,882	\$33,352	\$189,461
		5	\$70,698	\$47,873	\$35,898	\$38,536	\$193,005
		6	\$46,025			\$48,110	\$94,135
	3	7	\$139,040	\$65,156	\$43,256	\$72,418	\$319,870
		8	\$140,458	\$55,640	\$32,083	\$65,019	\$293,200
		9	\$140,177	\$6,515	\$30,634	\$62,984	\$240,310
	4	10	\$210,010	\$69,147	\$31,081	\$107,121	\$417,360
		11	\$397,834	\$121,934	\$97,607	\$183,657	\$801,032
		12	\$131,433	\$43,069	\$68,654	\$75,882	\$319,038
Grand Total			\$1,682,980	\$527,244	\$438,256	\$823,928	\$3,472,408

**Figure 3-5** Data Analyzer viewer with settable UIOptions off

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

For example, the cross tab in Figure 3-5 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();
```

The code in this example modifies the cross tab so it longer shows the month detail level, as shown in Figure 3-6.

		Classic Cars	Motorcycles	Planes	Vintage Cars	Grand Total
Year	Quarter	Revenue	Revenue	Revenue	Revenue	Revenue
2004	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 3-6** 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();
```

# 4

## **Actuate JavaScript API classes**

This chapter contains the following topics:

- 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 Information Console 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 Information Console installation or Actuate Deployment Kit for BIRT reports. The URL for the library is:

```
http://localhost:8700/portal/jsapi
```

where

- localhost:8700 is the host name and TCP port for an available Actuate web application host.
- /portal 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://localhost:8700  
  /portal/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 4-1 lists the Actuate JavaScript API classes.

**Table 4-1** 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.Dashboard	Dashboard class
actuate.dashboard.DashboardDefinition	Dashboard wrapper class
actuate.dashboard.EventConstants	Global constants for the dashboard events class
actuate.dashboard.GadgetScript	Dashboard gadget script class
actuate.dashboard.Tab	Dashboard tab class
Actuate.data	Container for actuate data class
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

*(continues)*

**Table 4-1** Actuate JavaScript API classes (continued)

JavaScript class	Description
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
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 Highchart object
actuate.report.HTML5Chart.Renderer	A Highchart 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

**Table 4-1** Actuate JavaScript API classes (continued)

JavaScript class	Description
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
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 `<script>` element loads the Actuate JavaScript API.

## Function summary

Table 4-2 lists `actuate` functions.

**Table 4-2**      `actuate` functions

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

## actuate.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, Encyclopedia 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.

**password**

The password parameter is an optional string parameter that contains the login password when the login password is not provided in the session.

**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 URL
- volume: The Encyclopedia volume name

**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:8700/iportal", null, myname,
    mypassword, null, null, null);
```

**actuate.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 " + getDefaultIportalUrl( ));
```

**actuate.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( ));
```

**actuate.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( ));
```

## actuate.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 `<div>` element.
- viewer**  
This parameter is the DOM `htmlelement` object for the HTML `<div>` 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` `<div>` element on the page, use code similar to the following:
- ```
currentViewer = actuate.getViewer("first_viewer");
```

## actuate.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, Encyclopedia 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.
- 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
- `volume`: The Encyclopedia volume name

**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:8700/iportal", null, null, null,
    init, null);
```

## **actuate.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, Encyclopedia 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:8700/iportal", null)){
    actuate.authenticate("http://digits:8700/iportal", null,
        myname, mypassword, null, null, null);
}
```



## actuate.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:8700/iportal", null, null,
        null, init, null);
}
```

## actuate.load

**Syntax** void load(string componentName)

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

- dashboard: The dashboard component including the actuate.Dashboard package
- 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 data analyzer component, including the actuate.XTabAnalyzer package

**Parameters** **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");
```

## actuate.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, Encyclopedia 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:8700/iportal", null)){
    actuate.logout("http://digits:8700/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 4-3 lists actuate.AuthenticationException functions.

**Table 4-3** actuate.AuthenticationException functions

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

### **actuate.AuthenticationException.getIportalUrl**

**Syntax** string AuthenticationException.getIportalUrl( )

Returns the Deployment Kit for BIRT reports or Information Console URL.

**Returns** String.

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

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

### **actuate.AuthenticationException.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, Encyclopedia 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( );
```

## **actuate.AuthenticationException.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 4-4 describes actuate.ConnectionException functions.

**Table 4-4** actuate.ConnectionException function

| Function  | Description           |
|-----------|-----------------------|
| getUrl( ) | Returns the whole URL |

### **actuate.ConnectionException.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.Dashboard**

**Description** Represents a dashboard object.

### Constructor

**Syntax** `actuate.Dashboard(string container)`

Constructs a dashboard object.

**Parameters** **container**  
String. Optional. Container object or name of a container in the current document ID of container where controls are to be rendered.

### Function summary

Table 4-5 describes `actuate.Dashboard` functions.

**Table 4-5** `actuate.Dashboard` functions

| Function                                | Description                                                              |
|-----------------------------------------|--------------------------------------------------------------------------|
| <code>downloadDashboard()</code>        | Downloads the dashboard definitions.                                     |
| <code>embedTemplate()</code>            | The personal dashboard uses an embedded template file.                   |
| <code>getActiveTab()</code>             | Returns the active tab name.                                             |
| <code>getDashboardName()</code>         | Returns the dashboard name used by the dashboard object.                 |
| <code>getTemplate()</code>              | Returns the iHub volume repository path.                                 |
| <code>isAutoSaveEnabled()</code>        | Returns whether autosave is enabled.                                     |
| <code>isSavingNeeded()</code>           | Returns whether there are unsaved changes on the dashboard.              |
| <code>isUsingPersonalDashboard()</code> | Returns whether the dashboard is a personal dashboard.                   |
| <code>onUnload()</code>                 | Unloads JavaScript variables that are no longer needed by the dashboard. |
| <code>registerEventHandler()</code>     | Registers an event handler.                                              |
| <code>removeEventHandler()</code>       | Removes an event handler.                                                |
| <code>renderContent()</code>            | Renders the dashboard.                                                   |
| <code>save()</code>                     | Saves the dashboard as a .dashboard file.                                |
| <code>saveAs()</code>                   | Saves the dashboard in non-default location.                             |
| <code>setActiveTab()</code>             | Sets a specific tab as the active tab.                                   |

**Table 4-5**      `actuate.Dashboard` functions

| Function                            | Description                                                                         |
|-------------------------------------|-------------------------------------------------------------------------------------|
| <code>setAutoSaveDelay()</code>     | Sets the time interval before executing an automatic save for a personal dashboard. |
| <code>setContainer()</code>         | Sets the container for rendering the dashboard page HTML fragment.                  |
| <code>setDashboardName()</code>     | Sets the dashboard name to view.                                                    |
| <code>setHeight()</code>            | Sets the dashboard height.                                                          |
| <code>setService()</code>           | Sets the connection to the Actuate web service.                                     |
| <code>setSize()</code>              | Sets the dashboard size.                                                            |
| <code>setTemplate()</code>          | Sets the template path.                                                             |
| <code>setWidth()</code>             | Sets the dashboard width.                                                           |
| <code>showGallery()</code>          | Shows the dashboard gallery.                                                        |
| <code>showTabNavigation()</code>    | Shows the tab toolbar.                                                              |
| <code>submit()</code>               | Submits the dashboard page component request.                                       |
| <code>usePersonalDashboard()</code> | Forces the dashboard framework to use a personal dashboard.                         |

## **`actuate.Dashboard.downloadDashboard`**

**Syntax**    `void Dashboard.downloadDashboard(function callback)`

Downloads the dashboard metadata definitions.

**Parameters**    **callback**

Function. The callback function to use after the dashboard finishes downloading. This function must take the returned dashboard object as an input parameter.

**Example**    This example specifies a function to call after the dashboard object finishes downloading:

```
myDashboard.downloadDashboard(runNext);  
function runNext(dashobject) {  
    myDashboard.getDashboardName(dashobject);  
}
```

## **`actuate.Dashboard.embedTemplate`**

**Syntax**    `void Dashboard.embedTemplate(boolean isEmbedded)`

A personal dashboard can use a shared template file or embed a template file.

**Parameters**    **isEmbedded**  
Boolean. When the isEmbedded parameter is true, the personal dashboard uses an embedded template file. The default value is false.

**Example**    This example specifies that the personal dashboard myDashboard uses an embedded template file:

```
myDashboard.embedTemplate(true);
```

## **actuate.Dashboard.getActiveTab**

**Syntax**    string Dashboard.getActiveTab  
Returns the name of the current active tab for the dashboard.

**Returns**    String. The name of the current active dashboard tab.

**Example**    This example displays the name of the active tab for the myDashboard dashboard object in an alert box:

```
alert(myDashboard.getActiveTab());
```

## **actuate.Dashboard.getDashboardName**

**Syntax**    string Dashboard.getDashboardName()  
Returns the dashboard name used by the dashboard object.

**Returns**    String. The dashboard's name.

**Example**    This example displays the dashboard object's dashboard name in an alert box:

```
alert(myDashboard.getDashboardName());
```

## **actuate.Dashboard.getTemplate**

**Syntax**    string Dashboard.getTemplate()  
Returns the repository path for the iHub volume.

**Returns**    String. The repository path for the iHub volume.

**Example**    This example displays the repository path for the iHub volume in an alert box:

```
alert(myDashboard.getTemplate());
```

## **actuate.Dashboard.isAutoSaveEnabled**

**Syntax**    boolean Dashboard.isAutoSaveEnabled()  
Returns whether the autosave feature is enabled.

**Returns**    Boolean. True indicates that autosave is enabled.



**Example** This example informs the user of the status of the autosave feature:

```
if (dashboard.isAutoSavEnabled()) {  
    alert("Autosave is enabled.");  
}else{  
    alert("Autosave is disabled.");  
}
```

## **actuate.Dashboard.isSavingNeeded**

**Syntax** boolean Dashboard.isSavingNeeded( )

Returns whether there are unsaved changes on the dashboard.

**Returns** Boolean. True indicates that there are unsaved changes on the dashboard.

**Example** This example informs the user of unsaved changed:

```
if (dashboard.isSavingNeeded()) {  
    alert("The dashboard contains unsaved changes.");  
}
```

## **actuate.Dashboard.isUsingPersonalDashboard**

**Syntax** boolean Dashboard.isUsingPersonalDashboard( )

Returns whether this dashboard is a personal dashboard.

**Returns** Boolean. True indicates that this dashboard is a personal dashboard.

**Example** This example informs the user that they are using a personal dashboard:

```
if (dashboard.isUsingPersonalDashboard()) {  
    alert("This is a personal dashboard.");  
}
```

## **actuate.Dashboard.onUnload**

**Syntax** void Dashboard.onUnload( )

Unloads JavaScript variables that are no longer needed by the dashboard.

**Example** This example unloads JavaScript variables and displays the dashboard object's dashboard name in an alert box:

```
myDashboard.onUnload;  
alert("JS variables unloaded for " +  
    myDashboard.getDashboardName( ));
```

## **actuate.Dashboard.registerEventHandler**

**Syntax** void Dashboard.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 dashboard 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:

```
myDashboard.registerEventHandler(actuate.dashboard.EventConstants.  
                                ON_EXCEPTION, errorHandler);
```

## actuate.Dashboard.removeEventHandler

**Syntax** void Dashboard.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:

```
myDashboard.removeEventHandler(actuate.dashboard.EventConstants.  
                                ON_EXCEPTION, errorHandler);
```

## actuate.Dashboard.renderContent

**Syntax** void Dashboard.renderContent(object[ ] dashboardDefinitions, function callback)  
Renders the dashboard definitions content to the container. The submit API calls the renderContent API internally. The renderContent() function assumes that the user has already a list of DashboardDefinition to process.

**Parameters** **dashboardDefinitions**  
Array of objects. Each object is some piece of dashboard metadata and as many can be added as needed. Typically, this array contains the following metadata:

- Number of tabs in a dashboard file
- Number of sections/columns in a dashboard tab
- Number of gadgets in each section/column

- Attributes of each gadget
- Attributes of each tab
- Dependency information between gadgets to support publishing and subscribing mechanism

**callback**

Function. The callback function to call after `renderContent()` finishes.

**Example** This example renders the `myDash` dashboard object using the `dashboardDefinition` array `defs` and calls `afterRender()` once complete:

```
myDash.renderContent(defs, afterRender);
```

## actuate.Dashboard.save

**Syntax** `void Dashboard.save(function callback, boolean flag)`

Saves the dashboard as a `.dashboard` file.

**Parameters** **callback**

Function. Optional. The function to execute after the save operation completes.

**flag**

Boolean. Optional. True indicates a synchronous save operation.

**Example** This example saves the dashboard as a `.dashboard` file:

```
myDash.save();
```

## actuate.Dashboard.saveAs

**Syntax** `void Dashboard.saveAs(function callback, string saveAsPath, boolean replace, boolean flag)`

Saves the dashboard as a `.dashboard` file to a specific path.

**Parameters** **callback**

Function. Optional. The function to execute after the save operation completes.

**saveAsPath**

String. Optional. Fully qualified path in which to save the dashboard. The default value is the path for the original dashboard file, if one exists, or the path for the user's home directory.

**replace**

Boolean. Optional. True indicates to replace the latest version of the file. False indicates to create a new version.

**flag**

Boolean. Optional. True indicates a synchronous save operation.

**Example** This example saves the dashboard as a .dashboard file, replacing the latest version:

```
myDash.saveAs(null, null, true, true);
```

## **actuate.Dashboard.setActiveTab**

**Syntax** void Dashboard.setActiveTab(string tabName)

Sets a specified tab as the active tab. Only one tab can be active at a time.

**Parameters** **tabName**  
String. The name of the tab to set as the active tab.

**Example** This example sets the Files tab as the active tab for this dashboard:

```
myDash.setActiveTab("Files");
```

## **actuate.Dashboard.setAutoSaveDelay**

**Syntax** void Dashboard.setAutoSaveDelay(integer seconds)

Sets the amount of time before executing an automatic save for a personal dashboard.

**Parameters** **seconds**  
Integer. The number of seconds to delay the automatic save.

**Example** This example sets the delay for the automatic save for dashboard myDash to 5 minutes:

```
myDash.setAutoSaveDelay(300);
```

## **actuate.Dashboard.setContainer**

**Syntax** void Dashboard.setContainer(string containerID)

The container that will be used for rendering the dashboard page HTML fragment.

**Parameters** **containerID**  
String. The container ID.

**Example** This example sets the container where the myDash dashboard object renders:

```
myDash.setContainer("leftpane");
```

## **actuate.Dashboard.setDashboardName**

**Syntax** void Dashboard.setDashboardName(string dashboardName)

Sets the dashboard name to view.

**Parameters**    **dashboardName**

String. A fully qualified repository path and file name.

**Example**    This example sets the path for the myDash dashboard object:

```
myDash.setDashboardName("/Dashboard/Contents/Hello.DASHBOARD");
```

## **actuate.Dashboard.setHeight**

**Syntax**    void Dashboard.setHeight(integer height)

Sets the dashboard's startup height.

**Parameters**    **height**

Integer. Specifies the height in pixels.

**Example**    To set the dashboard height to 400 pixels, use code similar to the following:

```
myDashboard.setHeight(400);
```

## **actuate.Dashboard.setService**

**Syntax**    void Dashboard.setService(string iportalURL, actuate.RequestOptions requestOptions)

Sets the web service this dashboard component connects to.

**Parameters**    **iportalURL**

String. The URL of the web service to connect to.

**requestOptions**

actuate.RequestOptions object. Request options, if any, to apply to the connection. See actuate.RequestOptions for details on the options that this parameter can set.

**Example**    This example connects a dashboard component to the iPortal service and adds a custom URL parameter:

```
function setDashboardService( ){
    myDashboard.setService("http://localhost:8700/iportal",
        myRequestOptions.setCustomParameters({myParam: "myValue"}));
}
```

## **actuate.Dashboard.setSize**

**Syntax**    void Dashboard.setSize(integer height, integer width)

Sets the dashboard's startup size.

**Parameters**    **height**

Integer. Height in pixels.

**width**

Integer. Width in pixels.

**Example** To set the dashboard height to 400 pixels and the width to 800 pixels, use code similar to the following:

```
myDashboard.setSize(400, 800);
```

## **actuate.Dashboard.setTemplate**

**Syntax** void Dashboard.setTemplate(string path)

Sets the template path. This function overwrites the template path that is used by Information Console.

**Parameters** **path**  
String. Specifies a new template path. Use an iHub volume repository path.

**Example** This example sets the template path for myDashboard to /iportal/jsapi/template/path:

```
myDashboard.setTemplate("/iportal/jsapi/template/path");
```

## **actuate.Dashboard.setWidth**

**Syntax** void Dashboard.setWidth(integer width)

Sets the dashboard's startup width.

**Parameters** **width**  
Integer. Specifies the width in pixels.

**Example** To set the dashboard width to 800 pixels, use code similar to the following:

```
myDashboard.setWidth(800);
```

## **actuate.Dashboard.showGallery**

**Syntax** void Dashboard.showGallery(boolean show)

Shows the gadget gallery.

**Parameters** **show**  
Boolean. The gadget gallery is visible when this parameter is set to true.

**Example** To show the gadget gallery for the myDashboard dashboard object, use code similar to the following:

```
myDashboard.showGallery(true);
```

## **actuate.Dashboard.showTabNavigation**

**Syntax** void Dashboard.showTabNavigation(boolean show)

Shows the tab toolbar.

- Parameters** **show**  
Boolean. The tab toolbar is visible when this parameter is set to true.
- Example** To show the tab toolbar for the myDashboard dashboard object, use code similar to the following:

```
myDashboard.showTabNavigation(true);
```

## **actuate.Dashboard.submit**

- Syntax** void Dashboard.submit(function callback)
- Submits requests to the server for the dashboard. When this function is called, an AJAX request is triggered to submit all pending operations. When the server finishes the processing, it returns a response and the results are rendered on the page in the dashboard container.
- Parameters** **callback**  
Function. The function to execute after the asynchronous call processing is done.
- Example** This example submits the dashboard name that was set with `setDashboardName()`:

```
dash.setDashboardName("/Dashboard/Contents/Hello.DASHBOARD");  
dash.submit();
```

## **actuate.Dashboard.usePersonalDashboard**

- Syntax** void Dashboard.usePersonalDashboard(boolean true|false)
- Forces the dashboard framework to use the user's personal dashboard.
- Parameters** **true|false**  
Boolean. A value of true sets the dashboard framework to ignore any value set by the `setDashboardName()` method. The dashboard framework creates a new personal dashboard file for the logged in user when no personal dashboard file is present.
- Example** To force the use of a personal dashboard for the myDashboard object, use code similar to the following:

```
myDashboard.usePersonalDashboard(true);
```

---

## Class **actuate.dashboard.DashboardDefinition**

**Description** The DashboardDefinition class is a wrapper class for a dashboard file definition.

### Constructor

**Syntax** `actuate.dashboard.DashboardDefinition( )`  
Constructs a new DashboardDefinition object.

### Function summary

Table 4-6 lists the `actuate.dashboard.DashboardDefinition` functions.

**Table 4-6** `actuate.dashboard.DashboardDefinition` functions

| Function                            | Description                                                              |
|-------------------------------------|--------------------------------------------------------------------------|
| <code>getDefaultActiveTab( )</code> | Returns the name of the default active tab for this dashboard definition |
| <code>getTabs( )</code>             | Returns an array of the tabs in this dashboard definition                |

### **actuate.dashboard.DashboardDefinition** **.getDefaultActiveTab**

**Syntax** `string DashboardDefinition.getDefaultActivetab( )`  
Returns the name of the default active tab for this dashboard definition.

**Returns** String. The name of the default active tab.

**Example** This example displays the default active tab for the `myDashDef` DashboardDefinition object in an alert box:

```
alert(myDashboard.getDefaultActiveTab( ));
```

### **actuate.dashboard.DashboardDefinition.getTabs**

**Syntax** `array DashboardDefinition.getTabs( )`  
Returns an array of the tabs in this dashboard definition.

**Returns** Array. An array of `actuate.dashboard.Tab` objects.

**Example** This example assigns the array of tabs to the `mytabs` variable:

```
var mytabs = new Array(myDashDef.getTabs( ));
```



---

## Class `actuate.dashboard.EventConstants`

**Description** Defines the event constants supported by the Dashboard class. Table 4-7 lists the dashboard event constants.

**Table 4-7** Actuate JavaScript API dashboard event constants

| Event              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ON_EXCEPTION       | Event triggered when an exception occurs.<br>The event handler takes an <code>actuate.Exception</code> object as an input parameter.                                                                                                                                                                                                                                                                                                                                               |
| ON_SESSION_TIMEOUT | Session time-out event.<br>Whenever a session time-out event occurs and the user tries to perform any operation on the explorer, a prompt dialog appears to ask whether the user wants to log in again. If the user clicks yes, the <code>ON_SESSION_TIMEOUT</code> event fires. If no handler has been registered for this event, the viewer displays a default login dialog.<br>The event handler takes the current <code>actuate.Dashboard</code> object as an input parameter. |

---

## Class `actuate.dashboard.GadgetScript`

**Description** The `actuate.dashboard.GadgetScript` class is a container for the information passed to the `onChange` event function.

### Constructor

**Syntax** `onChange( string event, actuate.dashboard.GadgetScript publisher, object data, actuate.dashboard.GadgetScript thisGadget )`

Constructs a new `GadgetScript` object. This object contains the publisher and `thisGadget` for an `onChange` event signature.

**Parameters** **event**  
String. An event name.

**publisher**  
`actuate.dashboard.GadgetScript` object. The publisher gadget.

**data**  
Object. Data to pass to the subscriber.

**thisGadget**  
`actuate.dashboard.GadgetScript` object. `thisGadget` is this script gadget.

### Function summary

Table 4-8 lists the `actuate.dashboard.GadgetScript` functions.

**Table 4-8** `actuate.dashboard.GadgetScript` functions

| Function                                   | Description                                                          |
|--------------------------------------------|----------------------------------------------------------------------|
| <code>getCurrentReportParameters( )</code> | Gets the current report parameter values for <code>thisGadget</code> |
| <code>getGadgetName( )</code>              | Returns <code>thisGadget</code> 's name                              |
| <code>getGadgetTitle( )</code>             | Returns <code>thisGadget</code> 's title                             |
| <code>getGadgetType( )</code>              | Returns <code>thisGadget</code> 's type                              |
| <code>getTabName( )</code>                 | Returns the name of the tab containing <code>thisGadget</code>       |
| <code>getTabTitle( )</code>                | Returns the title of the tab containing <code>thisGadget</code>      |

## **actuate.dashboard.GadgetScript .getCurrentReportParameters**

**Syntax** `actuate.parameter.ParameterValue[ ]  
GadgetScript.getCurrentReportParameters( )`

Returns the current report parameter values for report and Reportlet gadgets.

**Returns** Array of `actuate.parameter.ParameterValue` objects. Parameter values assigned to this gadget.

## **actuate.dashboard.GadgetScript.getGadgetName**

**Syntax** `string GadgetScript.getGadgetName( )`

Returns this gadget's name.

**Returns** String. The name of this gadget.

**Example** This example displays this gadget's name in an alert box:  
`alert(myGadgetScript.getGadgetName( ));`

## **actuate.dashboard.GadgetScript.getGadgetTitle**

**Syntax** `string GadgetScript.getGadgetTitle( )`

Returns this gadget's title.

**Returns** String. The title of this gadget.

**Example** This example displays this gadget's title in an alert box:  
`alert(myGadgetScript.getGadgetTitle( ));`

## **actuate.dashboard.GadgetScript.getGadgetType**

**Syntax** `string GadgetScript.getGadgetType( )`

Returns this gadget's type.

**Returns** String. This gadget's type.

**Example** This example displays this gadget's type in an alert box:  
`alert(myGadgetScript.getGadgetType( ));`

## **actuate.dashboard.GadgetScript.getTabName**

**Syntax** `string GadgetScript.getTabName( )`

Returns the name of the tab containing this gadget.

**Returns** String. The name of the tab containing this gadget.

**Example** This example displays the name of the tab containing this gadget in an alert box:

```
alert(myGadgetScript.getTabName( ));
```

## **actuate.dashboard.GadgetScript.getTabTitle**

**Syntax** string GadgetScript.getTabTitle( )

Returns the title of the tab containing this gadget.

**Returns** String. The title of the tab containing this gadget.

**Example** This example displays the title of the tab containing this gadget in an alert box:

```
alert(myGadgetScript.getTabTitle( ));
```

---

## Class **actuate.dashboard.Tab**

**Description** A wrapper class for the raw definition of a tab in a dashboard file.

### Constructor

**Syntax** `actuate.dashboard.Tab( )`  
Constructs a new tab object.

### Function summary

Table 4-9 lists the `actuate.dashboard.Tab` functions.

**Table 4-9** `actuate.dashboard.Tab` functions

| Function                   | Description             |
|----------------------------|-------------------------|
| <code>getName( )</code>    | Returns the tab's name  |
| <code>getTabType( )</code> | Returns the tab's type  |
| <code>getTitle( )</code>   | Returns the tab's title |

### **actuate.dashboard.Tab.getName**

**Syntax** `string Tab.getName( )`  
Returns the tab's name.

**Returns** String. The name of the tab.

**Example** This example displays the tab object's name in an alert box:  
`alert(myTab.getName( ));`

### **actuate.dashboard.Tab.getTabType**

**Syntax** `string Tab.getTabType( )`  
Returns the tab's type.

**Returns** String. The tab's type. The legal type values are `ISystemTabHandle` and `ITabHandle`.

**Example** This example displays the tab object's type in an alert box:  
`alert(myTab.getTabType( ));`

## **actuate.dashboard.Tab.getTitle**

**Syntax** `string Tab.getTitle( )`

Returns the tab's title.

**Returns** String. The title of the tab.

**Example** This example displays the tab object's title in an alert box:

```
alert(myTab.getTitle( ));
```

---

# 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 4-10 lists the valid filter operators and the number of arguments to pass to the constructor or `setValues()`.

**Table 4-10** 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                   |

*(continues)*

**Table 4-10** Filter operators (continued)

| Operator    | Description                                   | Number of arguments |
|-------------|-----------------------------------------------|---------------------|
| 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    | 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 4-11 lists `actuate.data.Filter` functions.

**Table 4-11**      `actuate.data.Filter` functions

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

### `actuate.data.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;  
}
```

### `actuate.data.Filter.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;  
}
```

## actuate.data.Filter.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;  
}
```

## actuate.data.Filter.setColumnName

**Syntax** void Filter.setColumnName(columnName)

Sets the name of the column to filter.

**Parameters** **columnName**  
String. The column name.

**Example** This example sets the name of the column to filter to Sales:

```
function setFilterSales( myfilter ) {  
    myfilter.setColumnName("Sales");  
}
```

## actuate.data.Filter.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.

**Parameters** **operator**  
String. The operator can be any valid operator. Table 4-10 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");  
}
```

## actuate.data.Filter.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 4-10 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.

**Parameters** **data**  
String. Content text.

### Function summary

Table 4-12 describes `actuate.data.ReportContent` functions.

**Table 4-12** `actuate.data.ReportContent` function

| Function                      | Description                                |
|-------------------------------|--------------------------------------------|
| <code>getTextContent()</code> | Returns the text in the downloaded content |

### **actuate.data.ReportContent.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 `ResultSet`s 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 4-13 lists `actuate.data.Request` functions.

**Table 4-13** `actuate.data.Request` functions

| Function                   | Description                                    |
|----------------------------|------------------------------------------------|
| <code>getBookmark()</code> | Returns the bookmark name                      |
| <code>getColumns()</code>  | Returns the column names                       |
| <code>getFilters()</code>  | Returns filters defined in this data condition |
| <code>getMaxRows()</code>  | Returns the max row number                     |
| <code>getSorters()</code>  | Returns sorters defined in this data condition |
| <code>getStartRow()</code> | Returns the start row number                   |
| <code>setBookmark()</code> | Sets the bookmark name                         |
| <code>setColumns()</code>  | Sets the columns to return                     |

*(continues)*

**Table 4-13**      `actuate.data.Request` functions (continued)

| Function                   | Description                            |
|----------------------------|----------------------------------------|
| <code>setFilters()</code>  | Sets the filters for the returned data |
| <code>setMaxRows()</code>  | Sets the max row number                |
| <code>setSorters()</code>  | Sets the sorters for the returned data |
| <code>setStartRow()</code> | Sets the start row number              |

## **`actuate.data.Request.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();
```

## **`actuate.data.Request.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];  
}
```

## **`actuate.data.Request.getFilters`**

**Syntax**    `actuate.data.Filter[] Request.getfilters()`

Returns filters set for this request.

**Returns**    Array of `actuate.data.Filter` objects.

## **`actuate.data.Request.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.

## **actuate.data.Request.getSorters**

**Syntax** `actuate.data.Sorter[ ] Request.getSorters( )`

Returns sorters assigned to this request.

**Returns** Array of `actuate.data.Sorter` objects.

## **actuate.data.Request.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.

## **actuate.data.Request.setBookmark**

**Syntax** `void Request.setBookmark(string bookmark)`

Sets the bookmark of the element from which to request values.

**Parameters** **bookmark**  
String. A bookmark.

**Example** This example sets the bookmark for the `myRequest` object to the string `myRequestStart`:

```
function setMyRequestBookmark(myRequest) {  
    myRequest.setBookmark("myRequestStart");  
}
```

## **actuate.data.Request.setColumns**

**Syntax** `void Request.setColumns(string[ ] columns)`

Sets the request column names.

**Parameters** **columns**  
An array of strings designating the columns of requested data. Use an array for this argument, even if there is only one value.

## **actuate.data.Request.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.

**Parameters** **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.

## **actuate.data.Request.setMaxRows**

**Syntax** `void Request.setMaxRows(integer maxrow)`

Sets the maximum number of rows to retrieve.

**Parameters** **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);
```

## **actuate.data.Request.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.

**Parameters** **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);
```

## **actuate.data.Request.setStartRow**

**Syntax** `void Request.setStartRow(integer startrow)`

Sets the requested first row.



**Parameters**    **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 4-14 lists `actuate.data.ResultSet` functions.

**Table 4-14**      `actuate.data.ResultSet` functions

| Function                      | Description                                |
|-------------------------------|--------------------------------------------|
| <code>getColumnNames()</code> | Returns the column names                   |
| <code>getValue()</code>       | Returns the data by the given column index |
| <code>next()</code>           | Increments the current row                 |

### **actuate.data.ResultSet.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];  
}
```

## **actuate.data.ResultSet.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.

**Parameters** **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);
```

## **actuate.data.ResultSet.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 4-15 lists `actuate.data.Sorter` functions.

**Table 4-15** `actuate.data.Sorter` functions

| Function                     | Description                                      |
|------------------------------|--------------------------------------------------|
| <code>getColumnName()</code> | Returns the column name                          |
| <code>isAscending()</code>   | Returns true if the current sorting is ascending |
| <code>setAscending()</code>  | Sets the sort order to ascending or descending   |
| <code>setColumnName()</code> | Sets the column to which this sorter applies     |

### **actuate.data.Sorter.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);  
}
```

## actuate.data.Sorter.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);  
    }  
}
```

## actuate.data.Sorter.setAscending

**Syntax** void Sorter.setAscending(boolean ascending)

Sets the sort order to ascending or descending.

**Parameters** **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);  
    }  
}
```

## actuate.data.Sorter.setColumnname

**Syntax** void Sorter.setColumnname(string columnName)

Sets the column to sort on.

**Parameters** **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 4-16 lists `actuate.DataService` functions.

**Table 4-16** `actuate.DataService` functions

| Function                          | Description                                                     |
|-----------------------------------|-----------------------------------------------------------------|
| <code>downloadResultSet( )</code> | Retrieves data from a report in a <code>ResultSet</code> object |

## `actuate.DataService.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://localhost:8900/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 4-17 lists `actuate.Exception` functions.

**Table 4-17** `actuate.Exception` functions

| Function                       | Description                                   |
|--------------------------------|-----------------------------------------------|
| <code>getDescription()</code>  | Returns details of the exception              |
| <code>getErrCode()</code>      | Returns error code for server-side exceptions |
| <code>getMessage()</code>      | Returns a short message about the exception   |
| <code>getType()</code>         | Returns the type of exception error           |
| <code>isExceptionType()</code> | Confirms exception type                       |

### `actuate.Exception.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( ));
```

## **actuate.Exception.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( ));
```

## **actuate.Exception.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( ));
```

## **actuate.Exception.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( ));
```

## **actuate.Exception.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.

- Parameters** **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.

**Parameters** **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 4-18 lists `actuate.Parameter` functions.

**Table 4-18** `actuate.Parameter` functions

Function	Description
<code>downloadParameters( )</code>	Returns an array of <code>ParameterDefinition</code> objects
<code>downloadParameterValues( )</code>	Returns an array list of <code>ParameterValue</code> objects
<code>getLayout( )</code>	Returns the parameter layout
<code>getParameterGroupNames( )</code>	Returns the names of the groups of parameters
<code>getReportName( )</code>	Returns the name of the report file
<code>getTransientDocumentName( )</code>	Returns the name of the transient document
<code>hideNavBar( )</code>	Hides the navigation bar
<code>hideParameterGroup( )</code>	Hides report parameters by group
<code>hideParameterName( )</code>	Hides parameters by name
<code>navigate( )</code>	Navigates the parameter page
<code>onUnload( )</code>	Unloads unused JavaScript variables

**Table 4-18**      `actuate.Parameter` functions

Function	Description
<code>registerEventHandler()</code>	Registers an event handler
<code>removeEventHandler()</code>	Removes an event handler
<code>renderContent()</code>	Renders the parameter content to the container
<code>setAutoSuggestDelay()</code>	Sets the autosuggest delay time
<code>setAutoSuggestFetchSize()</code>	Sets the fetch size of the autosuggestion list
<code>setAutoSuggestListSize()</code>	Sets the size of the autosuggestion list
<code>setContainer()</code>	Sets the HTML container for the parameter content
<code>setExpandedGroups()</code>	Sets the groups to expand by default
<code>setFont()</code>	Sets the font of the parameter page
<code>setGroupContainer()</code>	Sets the HTML container for the group
<code>setLayout()</code>	Sets the parameter layout type
<code>setReadOnly()</code>	Sets the parameter UI to read-only
<code>setReportName()</code>	Sets the remote report path and name
<code>setService()</code>	Sets the Actuate web application service
<code>setShowDisplayType()</code>	Sets the parameter page to display localized content
<code>submit()</code>	Submits all the asynchronous operations that the user has requested on this <code>Parameter</code> object and renders the parameter component on the page

## **`actuate.Parameter.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.

**Parameters**    **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( ));  
}
```

## **actuate.Parameter.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.

**Parameters** **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);  
}
```

## **actuate.Parameter.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( ));
```

## **actuate.Parameter.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]);
```

## **actuate.Parameter.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( ));
```

## **actuate.Parameter.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( ));
```

## **actuate.Parameter.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");
```

## actuate.Parameter.hideParameterGroup

**Syntax** void Parameter.hideParameterGroup(string[ ] groupNames)

Hides all report parameters that belongs to a group or to a list of groups.

**Parameters** **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");
```

## actuate.Parameter.hideParameterName

**Syntax** void Parameter.hideParameterName(string[ ] parameterNames)

Hides report parameters as specified by name.

**Parameters** **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");
```

## actuate.Parameter.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 <div> 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 <div> element with the myParams ID:

```
function myParamsLast(myParameter) {  
    myParameter.navigate("myParams", NAV_LAST);  
}
```



## actuate.Parameter.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.");
```

## actuate.Parameter.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\_EXCEPTION 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);  
}
```

## actuate.Parameter.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);  
}
```

## actuate.Parameter.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));  
}
```

## actuate.Parameter.setAutoSuggestDelay

**Syntax**    void Parameter.setAutoSuggestDelay(long delay)

Sets the autosuggest delay time.

**Parameters**    **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);  
}
```

## **actuate.Parameter.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.

**Parameters** **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);  
}
```

## **actuate.Parameter.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.

**Parameters** **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);  
}
```

## **actuate.Parameter.setContainer**

**Syntax** void Parameter.setContainer(string containerId)

Sets the HTML element container for the parameter content.

**Parameters**    **containerID**

String. The name of the HTML element that displays the group of rendered parameter components.

**Example**    This example sets the container where the parameter components render:

```
paramObj.setContainer("leftpane");
```

## **actuate.Parameter.setExpandedGroups**

**Syntax**    void Parameter.setExpandedGroups(groupNames)

Defines a set of groups that are expanded by default.

**Parameters**    **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);
```

## **actuate.Parameter.setFont**

**Syntax**    void Parameter.setFont(string fontStyleString)

Sets the font of the parameter page content after the page is rendered.

**Parameters**    **fontStyleString**

String. The name of a font.

**Example**    This example sets the font to Arial for the parameters page:

```
paramObj.setFont("arial");
```

## **actuate.Parameter.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");
```

## **actuate.Parameter.setLayout**

**Syntax** void Parameter.setLayout(string layoutName)

Sets the parameter layout.

**Parameters** **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");
```

## **actuate.Parameter.setReadOnly**

**Syntax** void Parameter.setReadOnly(boolean readOnly)

Sets the parameters to read-only.

**Parameters** **readOnly**  
Boolean. True indicates that the parameters are read-only.

**Example** This example makes the parameters read-only:

```
paramObj.setReadOnly(true);
```

## **actuate.Parameter.setReportName**

**Syntax** void Parameter.setReportName(string reportFile)

Sets the report file from which to get report parameters.

**Parameters** **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);
```

## actuate.Parameter.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, Encyclopedia 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://localhost:8700/  
iportal",myRequestOptions);
```

## actuate.Parameter.setShowDisplayType

**Syntax** void Parameter.setShowDisplayType(boolean showDisplayType)

Sets whether localized data is shown or not.

**Parameters** **showDisplayType**

Boolean. True indicates that localized data is shown.

**Example** This example hides localized data:

```
paramObj.setShowDisplayType(false);  
paramObj.submit(alert("Localized data hidden."));
```

## actuate.Parameter.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.

**Parameters** **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 4-19 lists the constants used for the parameter class.

**Table 4-19** 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.

**Parameters**    **aParamVals**  
Array of `actuate.parameter.ParameterValue` objects to convert.

### Function summary

Table 4-20 lists `actuate.parameter.ConvertUtility` functions.

**Table 4-20**    `actuate.parameter.ConvertUtility` functions

Function	Description
<code>convert( )</code>	Converts the <code>ParameterValues</code> to an array of generic objects
<code>convertData( )</code>	Converts locale-neutral parameter values to the user's login locale
<code>getParameterMap( )</code>	Returns the <code>ParameterValues</code> as an associative array
<code>getParameterValues( )</code>	Returns an array of <code>ParameterValues</code>

### **actuate.parameter.ConvertUtility.convert**

**Syntax**    `void ConvertUtility.convert(function callback)`

Converts `ParameterValues` into an array of generic objects. The callback function takes the array as an argument.

**Parameters**    **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;
}
```

## **actuate.parameter.ConvertUtility.convertDate**

**Syntax** void ConvertUtility.convertDate(function callback)

Converts locale-neutral parameter values to the user's login locale.

**Parameters** **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;
}
```

## **actuate.parameter.ConvertUtility.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( );
```

## **actuate.parameter.ConvertUtility.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 4-21 lists the parameter event constants.

**Table 4-21** 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: <ul style="list-style-type: none"><li>■ <code>actuate.Parameter</code>: parameter component for which the event occurred</li></ul>
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: <ul style="list-style-type: none"><li>■ <code>actuate.Parameter</code>: parameter component for which the event occurred</li></ul>
ON_EXCEPTION	Event triggered when an exception occurs. The event handler must take an <code>actuate.Exception</code> object as an input argument. The <code>Exception</code> 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 <code>actuate.Parameter</code> 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 <code>ON_SESSION_TIMEOUT</code> 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: <ul style="list-style-type: none"><li>■ <code>actuate.Parameter</code>: component for which the event occurred</li></ul>

---

---

## 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 4-22 lists `actuate.parameter.NameValuePair` functions.

**Table 4-22** `actuate.parameter.NameValuePair` functions

Function	Description
<code>getName( )</code>	Gets the name for this <code>NameValuePair</code>
<code>getValue( )</code>	Gets the value for this <code>NameValuePair</code>
<code>setName( )</code>	Sets the name for this <code>NameValuePair</code>
<code>setValue( )</code>	Sets the value for this <code>NameValuePair</code>

### **actuate.parameter.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( ));
```

## **actuate.parameter.NameValuePair.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());
```

## **actuate.parameter.NameValuePair.setName**

**Syntax** `void NameValuePair.setName(string name)`

Sets the name for the NameValuePair.

**Parameters** **name**  
String.

**Example** This sample code sets the name component of the myNVPair NameValuePair object to "My hometown":

```
myNVPair.setName("My hometown");
```

## **actuate.parameter.NameValuePair.setValue**

**Syntax** `void NameValuePair.setValue(string value)`

Sets the value for the NameValuePair.

**Parameters** **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 4-23 lists the `actuate.parameter.ParameterData` functions.

**Table 4-23** `actuateparameter.ParameterData` functions

Function	Description
<code>getCascadingParentValues()</code>	Returns the cascading parent value
<code>getChildData()</code>	Returns the child <code>ParameterData</code> object
<code>getControlType()</code>	Returns the <code>controlType</code> UI value
<code>getCurrentValue()</code>	Returns the current UI value set by the UI control
<code>getDefaultValue()</code>	Returns the default value for this <code>ParameterData</code> object
<code>getHelpText()</code>	Returns the help text for this <code>ParameterData</code> object
<code>getNameValueList()</code>	Returns the list of name-value pairs for this <code>ParameterData</code> object
<code>getParameterName()</code>	Returns the parameter name for this <code>ParameterData</code> object
<code>getParentData()</code>	Returns the parent <code>ParameterData</code> object
<code>getPickList()</code>	Returns the pick list for the child <code>ParameterData</code> object

**Table 4-23**      `actuateparameter.ParameterData` functions

Function	Description
<code>getPromptText()</code>	Returns the prompt text for this <code>ParameterData</code> object
<code>getSuggestionList()</code>	Returns the filter-based suggestion list for this <code>ParameterData</code> object
<code>isAdhoc()</code>	Returns true when this parameter is ad hoc
<code>isCascadingParameter()</code>	Returns true when this parameter is a cascading parameter
<code>isDynamicFilter()</code>	Returns true when this parameter is a dynamic filter
<code>isMultiList()</code>	Returns true when this parameter is a multi-list
<code>isRequired()</code>	Returns true when this parameter is required
<code>setChildData()</code>	Indicates that the parameter data contains a child
<code>setCurrentValue()</code>	Sets the UI value of the UI control
<code>setParentData()</code>	Indicates that the parameter data contains a parent
<code>setWebService()</code>	Defines a web service to send SOAP messages

## **`actuate.parameter.ParameterData` `.getCascadingParentValues`**

**Syntax**    `actuate.parameter.ParameterValue[ ]`  
              `ParameterData.getCascadingParentValues(`  
              `actuate.parameter.ParameterValue[ ] parentValues)`

Returns the cascading parent value.

**Parameters**    **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);
```

## **actuate.parameter.ParameterData.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( );
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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( );
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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( );
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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( );
```

## **actuate.parameter.ParameterData.getPickList**

**Syntax** `actuate.parameter.ParameterValue[ ] ParameterData.getPickList(function callback)`

Gets the pick list for the child of this parameter data.

**Parameters** **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;  
}
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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;  
}
```

## **actuate.parameter.ParameterData.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());`

## **actuate.parameter.ParameterData.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());`

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.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( ));
```

## **actuate.parameter.ParameterData.setChildData**

**Syntax** void ParameterData.setChildData(actuate.parameter.ParameterData childData)

Adds a child parameter to this parameter.

**Parameters** **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);
```

## **actuate.parameter.ParameterData.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.

**Parameters** **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);
```

## **actuate.parameter.ParameterData.setParentData**

**Syntax** void ParameterData.setParentData(parentData)

Sets a parent ParameterData object, making this ParameterData object its child.

**Parameters** **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);
```

## **actuate.parameter.ParameterData.setWebService**

**Syntax** void ParameterData.setWebService(object webService)

Defines a web service to use to send SOAP messages.

**Parameters** **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 4-24 lists `actuate.parameter.ParameterDefinition` functions.

**Table 4-24** `actuate.parameter.ParameterDefinition` functions

Function	Description
<code>getAutoSuggestThreshold()</code>	Gets the auto suggest threshold value for this <code>ParameterDefinition</code>
<code>getCascadingParentName()</code>	Gets the cascadingParentName value for this <code>ParameterDefinition</code>
<code>getColumnName()</code>	Gets the columnName value for this <code>ParameterDefinition</code>
<code>getColumnType()</code>	Gets the columnType value for this <code>ParameterDefinition</code>
<code>getControlType()</code>	Gets the controlType value for this <code>ParameterDefinition</code>
<code>getCurrentDisplayName()</code>	Gets the auto suggest current display name for the current value of this <code>ParameterDefinition</code>
<code>getDataType()</code>	Gets the dataType value for this <code>ParameterDefinition</code>
<code>getDefaultValue()</code>	Gets the defaultValue value for this <code>ParameterDefinition</code>

**Table 4-24**      `actuate.parameter.ParameterDefinition` functions (continued)

Function	Description
<code>getDefaultValueIsNull()</code>	Gets a flag if the default value is null for this <code>ParameterDefinition</code>
<code>getDisplayName()</code>	Gets the <code>displayName</code> value for this <code>ParameterDefinition</code>
<code>getGroup()</code>	Gets the <code>group</code> value for this <code>ParameterDefinition</code>
<code>getHelpText()</code>	Gets the <code>helpText</code> value for this <code>ParameterDefinition</code>
<code>getName()</code>	Gets the <code>name</code> value for this <code>ParameterDefinition</code>
<code>getOperatorList()</code>	Gets the list of valid operators
<code>getPosition()</code>	Gets the <code>position</code> value for this <code>ParameterDefinition</code>
<code>getSelectNameValueList()</code>	Gets the <code>selectNameValueList</code> value for this <code>ParameterDefinition</code>
<code>getSelectValueList()</code>	Gets the <code>selectValueList</code> value for this <code>ParameterDefinition</code>
<code>isAdHoc()</code>	Gets the <code>isAdHoc</code> value for this <code>ParameterDefinition</code>
<code>isHidden()</code>	Gets the <code>isHidden</code> value for this <code>ParameterDefinition</code>
<code>isPassword()</code>	Gets the <code>isPassword</code> value for this <code>ParameterDefinition</code>
<code>isRequired()</code>	Gets the <code>isRequired</code> value for this <code>ParameterDefinition</code>
<code>isViewParameter()</code>	Gets the <code>isViewParameter</code> value for this <code>ParameterDefinition</code>
<code>setAutoSuggestThreshold()</code>	Sets the <code>auto suggest threshold</code> value for this <code>ParameterDefinition</code>
<code>setCascadingParentName()</code>	Sets the <code>cascadingParentName</code> value for this <code>ParameterDefinition</code>
<code>setColumnName()</code>	Sets the <code>columnName</code> value for this <code>ParameterDefinition</code>
<code>setColumnType()</code>	Sets the <code>columnType</code> value for this <code>ParameterDefinition</code>

*(continues)*

**Table 4-24**      `actuate.parameter.ParameterDefinition` functions (continued)

Function	Description
<code>setControlType( )</code>	Sets the <code>controlType</code> value for this <code>ParameterDefinition</code>
<code>setCurrentDisplayName( )</code>	Sets the current display name for this <code>ParameterDefinition</code>
<code>setDataType( )</code>	Sets the <code>dataType</code> value for this <code>ParameterDefinition</code>
<code>setDefaultValue( )</code>	Sets the <code>defaultValue</code> value for this <code>ParameterDefinition</code>
<code>setDefaultValueIsNull( )</code>	Sets the <code>defaultValue</code> to null for this <code>ParameterDefinition</code>
<code>setDisplayName( )</code>	Sets the <code>displayName</code> value for this <code>ParameterDefinition</code>
<code>setGroup( )</code>	Sets the group value for this <code>ParameterDefinition</code>
<code>setHelpText( )</code>	Sets the <code>helpText</code> value for this <code>ParameterDefinition</code>
<code>setIsAdHoc( )</code>	Sets the <code>isAdHoc</code> value for this <code>ParameterDefinition</code>
<code>setIsHidden( )</code>	Sets the <code>isHidden</code> value for this <code>ParameterDefinition</code>
<code>setIsPassword( )</code>	Sets the <code>isPassword</code> value for this <code>ParameterDefinition</code>
<code>setIsRequired( )</code>	Sets the <code>isRequired</code> value for this <code>ParameterDefinition</code>
<code>setIsViewParameter( )</code>	Sets the <code>isViewParameter</code> value for this <code>ParameterDefinition</code>
<code>setName( )</code>	Sets the name value for this <code>ParameterDefinition</code>
<code>setPosition( )</code>	Sets the position value for this <code>ParameterDefinition</code>
<code>setSelectNameValueList( )</code>	Sets the <code>selectNameValueList</code> value for this <code>ParameterDefinition</code>
<code>setSelectValueList( )</code>	Sets the <code>selectValueList</code> value for this <code>ParameterDefinition</code>



## **actuate.parameter.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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .getDefaultValuesNull**

**Syntax** `boolean ParameterDefinition.getDefaultValuesNull( )`

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!');  
}
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition.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( ));
```

## **actuate.parameter.ParameterDefinition.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( );
```

## **actuate.parameter.ParameterDefinition.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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition.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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition .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( );
```

## **actuate.parameter.ParameterDefinition.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.setDefaultValueHandling(true);
}
```

## **actuate.parameter.ParameterDefinition.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);
}
```

## **actuate.parameter.ParameterDefinition.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);
}
```

## **actuate.parameter.ParameterDefinition.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.");  
}
```

## **actuate.parameter.ParameterDefinition .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.");  
}
```

## **actuate.parameter.ParameterDefinition .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.

**Parameters** **threshold**  
Integer.

**Example** To always show the autosuggest dialog for the parameter definition `paramdef`, use code similar to the following:

```
paramdef.setAutoSuggestThreshold(0);
```

## **actuate.parameter.ParameterDefinition .setCascadingParentName**

**Syntax** void ParameterDefinition.setCascadingParentName(string cascadingParentName)

Sets the cascadingParentName for this ParameterDefinition. This sets another parameter as this parameter's parent.

**Parameters** **cascadingParentName**  
String.

**Example** To set the parent name of the parameter definition paramdef to "Clark", use code similar to the following:

```
paramdef.setCascadingParentName("Clark");
```

## **actuate.parameter.ParameterDefinition .setColumnName**

**Syntax** void ParameterDefinition.setColumnName(string columnName)

Sets the columnName for this ParameterDefinition. Used for queries.

**Parameters** **columnName**  
String.

**Example** To set the parameter definition paramdef to access the ProductName column, use code similar to the following:

```
paramdef.setColumnName("ProductName");
```

## **actuate.parameter.ParameterDefinition .setColumnType**

**Syntax** void ParameterDefinition.setColumnType(string columnType)

Sets the columnType for this ParameterDefinition. Used for queries.

**Parameters** **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");
```



## **actuate.parameter.ParameterDefinition .setControlType**

**Syntax** void ParameterDefinition.setControlType(string controlType)

Sets the control type of this ParameterDefinition.

**Parameters** **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");
```

## **actuate.parameter.ParameterDefinition .setCurrentDisplayName**

**Syntax** void ParameterDefinition.setCurrentDisplayName(string currentDiplayName)

Sets the displayed name for this parameter.

**Parameters** **currentDisplayName**  
String.

**Example** To set the display name for the parameter definition paramdef to "Year", use code similar to the following:

```
paramdef.setCurrentDisplayName("Year");
```

## **actuate.parameter.ParameterDefinition.setDataType**

**Syntax** void ParameterDefinition.setDataType(string dataType)

Sets the dataType for this ParameterDefinition.

**Parameters** **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");
```

## **actuate.parameter.ParameterDefinition .setDefaultValue**

**Syntax** void ParameterDefinition.setDefaultValue(string defaultValue)

Sets the default value for this ParameterDefinition.

**Parameters**    **defaultValue**  
String.

**Example**    To set the default value of parameter definition paramdef to "2010", use code similar to the following:

```
paramdef.setDefaultValue("2010");
```

## **actuate.parameter.ParameterDefinition .setDefaultValuesNull**

**Syntax**    void ParameterDefinition.setDefaultValuesNull(boolean value)

When true, sets the default value for this ParameterDefinition to null. Sets the default value to no value in all other cases.

**Parameters**    **value**  
Boolean.

**Example**    To set the default value of parameter definition paramdef to null, use code similar to the following:

```
paramdef.setDefaultValuesNull(true);
```

## **actuate.parameter.ParameterDefinition .setDisplayName**

**Syntax**    void ParameterDefinition.setDisplayName(string displayName)

Sets the name to display on the parameter page for this ParameterDefinition.

**Parameters**    **displayName**  
String.

**Example**    To set the displayed name of parameter definition paramdef to "Year", use code similar to the following:

```
paramdef.setDisplayName("Year");
```

## **actuate.parameter.ParameterDefinition.setGroup**

**Syntax**    void ParameterDefinition.setGroup(string group)

Sets the group value for this ParameterDefinition.

**Parameters**    **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");
```

## **actuate.parameter.ParameterDefinition.setHelpText**

**Syntax** `void ParameterDefinition.setHelpText(string helpText)`

Sets the `helpText` value for this `ParameterDefinition`.

**Parameters** **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.");  
}
```

## **actuate.parameter.ParameterDefinition.setIsAdHoc**

**Syntax** `void ParameterDefinition.setIsAdHoc(boolean isAdHoc)`

Sets this parameter as an ad hoc parameter.

**Parameters** **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);
```

## **actuate.parameter.ParameterDefinition.setIsHidden**

**Syntax** `void ParameterDefinition.setIsHidden(boolean isHidden)`

Sets the parameter to hidden.

**Parameters** **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);
```

## **actuate.parameter.ParameterDefinition .setIsMultiSelectControl**

**Syntax** void ParameterDefinition.setIsMultiSelectControl(boolean isMultiSelect)

Sets the parameter to accept multiple selected values.

**Parameters** **isHMultiSelect**

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);
```

## **actuate.parameter.ParameterDefinition .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.

**Parameters** **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);
```

## **actuate.parameter.ParameterDefinition.setIsRequired**

**Syntax** void ParameterDefinition.setIsRequired(boolean isRequired)

Sets the parameter to required.

**Parameters** **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);
```

## **actuate.parameter.ParameterDefinition .setIsViewParameter**

**Syntax** void ParameterDefinition.setIsViewParameter(boolean isViewParameter)

Sets the isViewParameter value for this ParameterDefinition.

**Parameters** **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);
```

## **actuate.parameter.ParameterDefinition.setName**

**Syntax** void ParameterDefinition.setName(string name)  
Sets the name to use internally for this ParameterDefinition.

**Parameters** **name**  
String.

**Example** To set the internal name of the parameter definition paramdef to Year, use code similar to the following:

```
paramdef.setName("Year");
```

## **actuate.parameter.ParameterDefinition.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.

**Parameters** **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());
```

## **actuate.parameter.ParameterDefinition .setSelectNameValueList**

**Syntax** void ParameterDefinition.setSelectNameValueList  
(actuate.parameter.NameValuePair[] selectNameValueList)  
Sets the selectNameValueList value for this ParameterDefinition.

**Parameters** **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());
```

## **actuate.parameter.ParameterDefinition .setSelectValueList**

**Syntax** void ParameterDefinition.setSelectValueList(array[ ] selectValueList)

Sets the selectValueList value for this ParameterDefinition.

**Parameters** **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 than `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 4-25 lists `actuate.parameter.ParameterValue` functions.

**Table 4-25**    `actuate.parameter.ParameterValue` functions

Function	Description
<code>getColumnName( )</code>	Gets the name of the column in this <code>ParameterValue</code>
<code>getColumnType( )</code>	Gets the data type value of the column for this <code>ParameterValue</code>
<code>getDataType( )</code>	Gets the <code>dataType</code> value for this <code>ParameterValue</code>
<code>getDisplayName( )</code>	Gets the displayed name for this <code>ParameterValue</code>
<code>getGroup( )</code>	Gets the group value for this <code>ParameterValue</code>
<code>getName( )</code>	Gets the name value for this <code>ParameterValue</code>
<code>getPosition( )</code>	Gets the position value for this <code>ParameterValue</code>
<code>getPromptParameter( )</code>	Gets the <code>promptParameter</code> value for this <code>ParameterValue</code>
<code>getValue( )</code>	Gets the value or values for this <code>ParameterValue</code>
<code>getValueIsNull( )</code>	Gets the <code>valueIsNull</code> value for this <code>ParameterValue</code>
<code>isViewParameter( )</code>	Gets the <code>isViewParameter</code> value for this <code>ParameterValue</code>

*(continues)*

**Table 4-25**      `actuate.parameter.ParameterValue` functions (continued)

Function	Description
<code>setColumnName( )</code>	Sets the name of the column in this <code>ParameterValue</code>
<code>setColumnType( )</code>	Sets the data type value of the column for this <code>ParameterValue</code>
<code>setDataType( )</code>	Sets the <code>dataType</code> value for this <code>ParameterValue</code>
<code>setDisplayName( )</code>	Sets the displayed name for this <code>ParameterValue</code>
<code>setGroup( )</code>	Sets the group value for this <code>ParameterValue</code>
<code>setIsViewParameter( )</code>	Sets the <code>isViewParameter</code> value for this <code>ParameterValue</code>
<code>setName( )</code>	Sets the name value for this <code>ParameterValue</code>
<code>setPosition( )</code>	Sets the position value for this <code>ParameterValue</code>
<code>setPromptParameter( )</code>	Sets the <code>promptParameter</code> value for this <code>ParameterValue</code>
<code>setValue( )</code>	Sets the value for this <code>ParameterValue</code>
<code>setValueIsNull( )</code>	Sets the <code>valueIsNull</code> value for this <code>ParameterValue</code>

## **`actuate.parameter.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( );
```

## **`actuate.parameter.ParameterValue`.`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( );
```

## **actuate.parameter.ParameterValue.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( );
```

## **actuate.parameter.ParameterValue.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 displayName, use code similar to the following:

```
var displayName = pvalue.getDisplayName( );
```

## **actuate.parameter.ParameterValue.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( );
```

## **actuate.parameter.ParameterValue.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( );
```

## **actuate.parameter.ParameterValue.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( );
```

## **actuate.parameter.ParameterValue.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( );
```

## **actuate.parameter.ParameterValue.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( );
```

## **actuate.parameter.ParameterValue.getValuesIsNull**

**Syntax** boolean ParameterValue.getValuesIsNull( )

Gets the valuesIsNull 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 (pvalue.getValueIsNull( )){  
    alert('Default value is null!');  
}
```

## **actuate.parameter.ParameterValue.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.");  
}
```

## **actuate.parameter.ParameterValue.setColumnName**

**Syntax** void ParameterValue.setColumnName(string columnName)

Sets the column name value for this ParameterValue.

**Parameters** **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");
```

## **actuate.parameter.ParameterValue.setColumnType**

**Syntax** void ParameterValue.setColumnType(string columnType)

Sets the data type of the column for this ParameterValue. Used for queries.

**Parameters** **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");
```

## actuate.parameter.ParameterValue.setDataType

**Syntax** void ParameterValue.setDataType(string dataType)

Sets the dataType value for this ParameterValue.

**Parameters** **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");
```

## actuate.parameter.ParameterValue.setDisplayName

**Syntax** void ParameterValue.setDisplayName(string name)

Sets the displayed name value for this ParameterValue.

**Parameters** **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");
```

## actuate.parameter.ParameterValue.setGroup

**Syntax** void ParameterValue.setGroup(string group)

Sets the group value for this ParameterValue.

**Parameters** **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");
```

## actuate.parameter.ParameterValue .setIsViewParameter

**Syntax** void ParameterValue.setIsViewParameter(boolean isViewParameter)

Sets the isViewParameter value for this ParameterValue.

**Parameters** **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);
```

## **actuate.parameter.ParameterValue.setName**

**Syntax** void ParameterValue.setName(string name)

Sets the name value for this ParameterValue.

**Parameters** **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");
```

## **actuate.parameter.ParameterValue.setPosition**

**Syntax** void ParameterValue.setPosition(integer position)

Sets the position value for this ParameterValue.

**Parameters** **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());
```

## **actuate.parameter.ParameterValue.setPromptParameter**

**Syntax** void ParameterValue.setPromptParameter(boolean promptParameter)

Sets the promptParameter value for this ParameterValue.

**Parameters** **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);
```

## **actuate.parameter.ParameterValue.setValue**

**Syntax** void ParameterValue.setValue(string[] value)

Sets the value or values for this `ParameterValue`.

**Parameters**    **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"});
```

## **actuate.parameter.ParameterValue.setValuesNull**

**Syntax**    `void ParameterValue.setValuesNull(boolean valuesNull)`

Sets the `valuesNull` value for this `ParameterValue`.

**Parameters**    **valuesNull**  
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 4-26 lists `actuate.report.Chart` functions.

**Table 4-26** `actuate.report.Chart` functions

Function	Description
<code>clearFilters( )</code>	Clears the filters applied to the given column
<code>drillDownCategory( )</code>	Drills down into a chart by category
<code>drillDownSeries( )</code>	Drills down into a chart by series
<code>drillUpCategory( )</code>	Drills up one level by category
<code>drillUpSeries( )</code>	Drills up one level by series
<code>getBookmark( )</code>	Returns the report element bookmark name
<code>getClientChart( )</code>	Returns an HTML5 instance of this chart
<code>getHtmlDom( )</code>	Returns the HTML element DOM object
<code>getInstanceId( )</code>	Returns the report element instance id
<code>getPageContent( )</code>	Returns the page content to which this element belongs
<code>getType( )</code>	Returns the report element type
<code>hide( )</code>	Hides this element
<code>setChartTitle( )</code>	Sets the title for this chart
<code>setDimension( )</code>	Sets the number of dimensions for the chart element
<code>setFilters( )</code>	Applies filters to this chart element
<code>setSize( )</code>	Sets the width and height of the chart element
<code>setSubType( )</code>	Sets a chart subtype to the chart element
<code>show( )</code>	Shows this element

*(continues)*

**Table 4-26**      `actuate.report.Chart` functions (continued)

Function	Description
<code>submit()</code>	Submits all the asynchronous operations that the user has requested on this report and renders the chart component on the page

## **`actuate.report.Chart.clearFilters`**

**Syntax**    `void Chart.clearFilters(string columnName)`

Clears the filters for a given column.

**Parameters**    **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();  
}
```

## **`actuate.report.Chart.drillDownCategory`**

**Syntax**    `void Chart.drillDownCategory(string categoryData)`

Drills down into a chart by category.

**Parameters**    **categoryData**  
String. The name of the data category to drill down to.

## **`actuate.report.Chart.drillDownSeries`**

**Syntax**    `void Chart.drillDownSeries(string seriesName)`

Drills down into a chart by series.

**Parameters**    **seriesName**  
String. The name of the data series to drill down to.

## **`actuate.report.Chart.drillUpCategory`**

**Syntax**    `void Chart.drillUpCategory()`

Drills up into a chart by one data category level.



## **actuate.report.Chart.drillUpSeries**

**Syntax** void Chart.drillUpSeries( )  
Drills up into a chart by one series level.

## **actuate.report.Chart.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( );  
}
```

## **actuate.report.Chart.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);  
}
```

## **actuate.report.Chart.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);  
}
```

## **actuate.report.Chart.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);  
}
```

## **actuate.report.Chart.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);  
}
```

## **actuate.viewer.Chart.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( ));
```

## **actuate.report.Chart.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( );
```

## **actuate.report.Chart.setChartTitle**

**Syntax** `void Chart.setChartTitle(string title)`  
Sets the title for this chart element.

**Parameters** **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( );  
}
```

## **actuate.report.Chart.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.

**Parameters** **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_D  
    EPTH );  
bchart.submit( );
```

## actuate.report.Chart.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( );  
}
```

## actuate.report.Chart.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( );
```

## **actuate.report.Chart.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.

**Parameters** **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( );
```

## **actuate.report.Chart.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( );
```

## **actuate.report.Chart.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.

**Parameters**    **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 4-27 lists `actuate.report.DataItem` functions.

**Table 4-27** `actuate.report.DataItem` functions

Function	Description
<code>getBookmark()</code>	Returns the bookmark name for this data item
<code>getData()</code>	Returns the data value on this data element
<code>getHtmlDom()</code>	Returns the HTML element for this data item
<code>getPageContent()</code>	Returns the page content to which this element belongs
<code>getType()</code>	Returns the report element type
<code>hide()</code>	Hides this element
<code>show()</code>	Shows this element
<code>submit()</code>	Applies the changes made to this <code>DataItem</code>

## `actuate.report.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());`

## `actuate.report.DataItem.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( ));
```

## **actuate.report.DataItem.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);
}
```

## **actuate.report.DataItem.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);
}
```

## **actuate.report.DataItem.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);  
}
```

## **actuate.report.DataItem.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");  
}
```

## **actuate.report.DataItem.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( );
```

## **actuate.report.DataItem.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( );
```

## **actuate.report.DataItem.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.

**Parameters**    **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 4-28 lists `actuate.report.FlashObject` functions.

**Table 4-28** `actuate.report.FlashObject` functions

Function	Description
<code>clearFilters( )</code>	Removes filters from this <code>FlashObject</code>
<code>getBookmark( )</code>	Returns the bookmark name for this <code>FlashObject</code>
<code>getHtmlDom( )</code>	Returns the HTML element for this <code>FlashObject</code>
<code>getInstanceId( )</code>	Returns the report element instance id
<code>getPageContent( )</code>	Returns the page content to which this element belongs
<code>getType( )</code>	Returns the <code>FlashObject</code> 's element type
<code>hide( )</code>	Hides this element
<code>setFilters( )</code>	Adds filters to this <code>FlashObject</code>
<code>show( )</code>	Shows this element
<code>submit( )</code>	Applies changes made to this <code>FlashObject</code>

## `actuate.report.FlashObject.clearFilters`

**Syntax** `void FlashObject.clearFilters(string columnName)`

Clears the filters of a given column.

**Parameters** **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( );  
}
```

## **actuate.report.FlashObject.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( ));  
}
```

## **actuate.report.FlashObject.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);  
}
```

## **actuate.report.FlashObject.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);  
}
```

## **actuate.report.FlashObject.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);  
}
```

## **actuate.report.FlashObject.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");  
}
```

## **actuate.report.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( );
```

## actuate.report.FlashObject.setFilters

**Syntax** void FlashObject.setFilters(actuate.data.Filter[ ] filters)

Sets the given filters.

**Parameters** **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);  
}
```

## actuate.report.FlashObject.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( );
```

## actuate.report.FlashObject.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.

**Parameters** **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 4-29 lists `actuate.report.Gadget` functions.

**Table 4-29** `actuate.report.Gadget` functions

Function	Description
<code>clearFilters()</code>	Removes filters from this gadget
<code>getBookmark()</code>	Returns the bookmark name for this gadget
<code>getHtmlDom()</code>	Returns the HTML element for this gadget
<code>getInstanceId()</code>	Returns the report element instance id
<code>getPageContent()</code>	Returns the page content to which this element belongs
<code>getType()</code>	Returns the gadget's element type, which is gadget
<code>hide()</code>	Hides this element
<code>setFilters()</code>	Adds filters to this gadget
<code>setGadgetType()</code>	Sets the gadget type
<code>setSize()</code>	Resizes the gadget's width and height
<code>show()</code>	Shows this element
<code>submit()</code>	Applies changes made to this gadget

### `actuate.report.Gadget.clearFilters`

**Syntax** `void Gadget.clearFilters(string columnName)`

Clears the filters of a given column.

**Parameters** **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( );  
}
```

## **actuate.report.Gadget.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( ));  
}
```

## **actuate.report.Gadget.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);  
}
```

## **actuate.report.Gadget.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);  
}
```

## **actuate.report.Gadget.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);  
}
```

## **actuate.report.Gadget.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");  
}
```

## **actuate.report.Gadget.hide**

**Syntax** `void Gadget.hide( )`

Hides this element.

**Example** Use `hide( )` to hide a gadget, as shown in the following code:

```
myGadget.show( );
```

## **actuate.report.Gadget.setFilters**

**Syntax** `void Gadget.setFilters(actuate.data.Filter[ ] filters)`

Sets the given filters.

**Parameters** **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);
}
```

## **actuate.report.Gadget.setGadgetType**

**Syntax** `void Gadget.setGadgetType(string chartType)`

Specifies the gadget type for the Gadget element. The chart type is a constant.

**Parameters** **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);
```

## **actuate.report.Gadget.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);
```

## **actuate.report.Gadget.show**

**Syntax** void Gadget.show( )

Shows this element.

**Example** Use show( ) to reveal a hidden gadget, as shown in the following code:

```
myGadget.show( );
```

## **actuate.report.Gadget.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.

**Parameters** **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 4-30 lists `actuate.report.HTML5Chart.ClientChart` functions.

**Table 4-30** `actuate.report.HTML5Chart.ClientChart` functions

Function	Description
<code>addSeries()</code>	Adds a series to the chart
<code>getCategoryCount()</code>	Returns the number of categories in the chart
<code>getChartHeight()</code>	Returns the height of the chart in pixels
<code>getChartWidth()</code>	Returns the width of the chart in pixels
<code>getClientOptions()</code>	Returns the chart options
<code>getCore()</code>	Returns the core Highcharts object
<code>getSeriesCount()</code>	Returns the number of run-time series in the chart
<code>getXAxisMax()</code>	Returns the maximum value of X-axis series
<code>getXAxisMin()</code>	Returns the minimum value of X-axis series
<code>getYAxisMax()</code>	Returns the maximum value of Y-axis series
<code>getYAxisMin()</code>	Returns the minimum value of Y-axis series
<code>isChartWithAxes()</code>	Returns whether chart has axes
<code>redraw()</code>	Redraws the chart according to chart options
<code>removeSeries()</code>	Removes specified series
<code>setSeriesVisible()</code>	Hides or displays specified series
<code>setTitle()</code>	Updates chart title
<code>setValues()</code>	Updates values of specified series
<code>setXAxisRange()</code>	Changes the minimum and maximum of the X-axis and zooms in on the new data range

**Table 4-30**      `actuate.report.HTML5Chart.ClientChart` functions

Function	Description
<code>setYAxisRange( )</code>	Changes the minimum and maximum of the Y-axis and zooms in on the new data range

## **`actuate.report.HTML5Chart.ClientChart.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]);
```

## **`actuate.report.HTML5Chart.ClientChart` `.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.");
```

## **`actuate.report.HTML5Chart.ClientChart` `.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( ));
```

## **actuate.report.HTML5Chart.ClientChart .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( ));
```

## **actuate.report.HTML5Chart.ClientChart .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( );
```

## **actuate.report.HTML5Chart.ClientChart.getCore**

**Syntax** actuate.report.HTML5Chart.Highcharts ClientChart.getCore( )  
Returns the Highcharts object contained in this ClientChart.

**Returns** actuate.report.HTML5Chart.Highcharts object. A highchart.

**Example** This example retrieves the Highcharts object from myClientChart and stores it in the myHighchart variable:  

```
var myHighchart = myClientChart.getCore( );
```

## **actuate.report.HTML5Chart.ClientChart .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( ));
```

## **actuate.report.HTML5Chart.ClientChart.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( ));
```

## **actuate.report.HTML5Chart.ClientChart.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( ));
```

## **actuate.report.HTML5Chart.ClientChart.getYAxisMax**

**Syntax** float ClientChart.getYAxisMax( integer axisIndex)

Returns the maximum value of a series associated with the Y-axis in this ClientChart.

**Parameters** **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( ));
```

## **actuate.report.HTML5Chart.ClientChart.getYAxisMin**

**Syntax** float ClientChart.getYAxisMin( integer axisIndex)

Returns the minimum value of a series associated with the Y-axis in this ClientChart.

**Parameters** **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( ));
```

## **actuate.report.HTML5Chart.ClientChart.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( ));
```

## **actuate.report.HTML5Chart.ClientChart.redraw**

**Syntax** void ClientChart.redraw(actuate.report.HTML5Chart.ClientOption chartOptions)

Redraws this ClientChart with options.

**Parameters** **chartOptions**  
actuate.report.HTML5Chart.ClientOption object. Optional. The chart options.

**Example** This example redraws myClientChart with the default options:

```
myClientChart.redraw( );
```

## **actuate.report.HTML5Chart.ClientChart.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);
```

## **actuate.report.HTML5Chart.ClientChart.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);
```

## **actuate.report.HTML5Chart.ClientChart.setTitle**

**Syntax** void ClientChart.setTitle(string title)

Sets the title of this ClientChart.

**Parameters** **title**

String. Chart title text.

**Example** This example sets the title of myClientChart to 'Annual Report':

```
myClientChart.setTitle('Annual Report');
```

## **actuate.report.HTML5Chart.ClientChart.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]);
```

## **actuate.report.HTML5Chart.ClientChart .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);
```

## **actuate.report.HTML5Chart.ClientChart .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 4-31 lists `actuate.report.HTML5Chart.ClientOption` functions.

**Table 4-31** `actuate.report.HTML5Chart.ClientOption` functions

Function	Description
<code>addSeries( )</code>	Adds a series to the chart
<code>explodePieSlice( )</code>	Explodes specified pie's slice
<code>isChartWithAxes( )</code>	Checks if current chart is chart with axes
<code>pivotChart( )</code>	Inverts chart
<code>setChartType( )</code>	Updates chart type
<code>setSeriesVisible( )</code>	Hides or shows specified series
<code>setTitle( )</code>	Updates chart title
<code>setXAxisTitle( )</code>	Updates X-axis title
<code>setYAxisTitle( )</code>	Updates Y-axis title

## `actuate.report.HTML5Chart.ClientOption.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]);
```

## **actuate.report.HTML5Chart.ClientOption .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');
```

## **actuate.report.HTML5Chart.ClientOption .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());
```

## **actuate.report.HTML5Chart.ClientOption.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);
```

## **actuate.report.HTML5Chart.ClientOption .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');`

## **actuate.report.HTML5Chart.ClientOption .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);`

## **actuate.report.HTML5Chart.ClientOption.setTitle**

**Syntax** `void ClientOption.setTitle(string title)`  
Sets the title of this ClientOption.

**Parameters** **title**  
String. Chart title text.

**Example** This example sets the title of myClientOption to 'Annual Report':  
`myClientOption.setTitle('Annual Report');`

## **actuate.report.HTML5Chart.ClientOption .setXAxisTitle**

**Syntax** `void ClientOption.setXAxisTitle(string title)`  
Sets the X-axis title of this ClientOption.

**Parameters** **title**  
String. X-axis title text.

**Example** This example sets the title of the X-axis in myClientOption to 'Month':  
`myClientOption.setXAxisTitle('Month');`

## **actuate.report.HTML5Chart.ClientOption .setYAxisTitle**

**Syntax** void ClientOption.setTitle(string title)

Sets the Y-axis title of this ClientOption.

**Parameters** **title**

String. Y-axis title text.

**Parameters** **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://www.actuate.com/documentation/R11SP4/actuatebirth/highcharts/Highcharts-Options-Reference.htm>

### Constructor

**Syntax** `void actuate.report.HTML5Chart.ClientPoint( )`

Generates a new ClientPoint object to manage a data point for a ClientChart.

### Function summary

Table 4-32 lists actuate.report.HTML5Chart.ClientPoint functions.

**Table 4-32** actuate.report.HTML5Chart.ClientPoint functions

Function	Description
<code>applyOptions( )</code>	Changes the point values or options
<code>destroy( )</code>	Destroys a point to clear memory
<code>remove( )</code>	Removes a point
<code>select( )</code>	Toggles the selection of a point
<code>update( )</code>	Updates the point with new options

### **actuate.report.HTML5Chart.ClientPoint.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.

**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.

**Example** This example changes the Y value of myClientPoint to 12:

```
myClientPoint.applyOptions(12);
```

## actuate.report.HTML5Chart.ClientPoint.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( );
```

## actuate.report.HTML5Chart.ClientPoint.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( );
```

## actuate.report.HTML5Chart.ClientPoint.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);
```

## actuate.report.HTML5Chart.ClientPoint.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();
```

**actuate.report.HTML5Chart.ClientPoint.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 4-33 lists actuate.report.HTML5Chart.ClientSeries functions.

**Table 4-33** 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

### **actuate.report.HTML5Chart.ClientSeries.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:

```
myClientSeries.addPoint(12);
```

## **actuate.report.HTML5Chart.ClientSeries.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( );
```

## **actuate.report.HTML5Chart.ClientSeries.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( );
```

## **actuate.report.HTML5Chart.ClientSeries.hide**

**Syntax** void ClientSeries.hide( )

Hides the graph of this series.

**Example** This example hides myClientSeries graph from the chart:

```
myClientSeries.hide( );
```

## **actuate.report.HTML5Chart.ClientSeries.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( );
```

## **actuate.report.HTML5Chart.ClientSeries.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( );
```

## **actuate.report.HTML5Chart.ClientSeries.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( );
```

## **actuate.report.HTML5Chart.ClientSeries.select**

**Syntax** void ClientSeries.select(boolean selected)

Selects this series.

**Parameters** **selected**

Boolean. Optional. Specifies whether to select or deselect the series. If undefined, toggles selection.

**Example** This example selects myClientSeries:

```
myClientSeries.select(true);
```

## **actuate.report.HTML5Chart.ClientSeries.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 Highchart 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]]);
```

## **actuate.report.HTML5Chart.ClientSeries.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);
```

## **actuate.report.HTML5Chart.ClientSeries.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 highchart documentation at the following URL:

<http://www.actuate.com/documentation/R11SP4/actuatebirt/highcharts/Highcharts-Options-Reference.htm>

### **Constructor**

**Syntax** void actuate.report.HTML5Chart.Highcharts( )

Generates a new Highcharts object to manage the Highchart for a ClientChart.

---

## Class **actuate.report.HTML5Chart.Renderer**

**Description** A container for a Highchart renderer object. Directly accesses the Highchart's 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 Highchart rendering options for a ClientChart.

### Function summary

Table 4-34 lists actuate.report.HTML5Chart.Renderer functions.

**Table 4-34** 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

### **actuate.report.HTML5Chart.Renderer.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  $-\text{Math.PI}/2$  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  $-\text{Math.PI}/2$  is directly upward.

**Returns** Highchart element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help or access the highchart documentation at the following URL:

<http://www.actuate.com/documentation/R11SP4/actuatebirt/highcharts/Highcharts-Options-Reference.htm#element>

**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);
```

## **actuate.report.HTML5Chart.Renderer.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** Highchart element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.

**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);
```

## **actuate.report.HTML5Chart.Renderer.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** Highchart element object. The Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.

**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);
```

## **actuate.report.HTML5Chart.Renderer.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()`;

## **actuate.report.HTML5Chart.Renderer.g**

**Syntax** object `Renderer.g(string name)`

Adds an SVG/VML group to the Renderer object.

- Parameters**    **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**    Highchart element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.
- Example**    This example creates a new group called myGroup:
- ```
myRenderer.g( 'myGroup' ) ;
```

## actuate.report.HTML5Chart.Renderer.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**    Highchart element object. The Highchart.Element class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.
- 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 );
```

## actuate.report.HTML5Chart.Renderer.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.

- Parameters** **path**  
Array of string and integer objects. An SVG path with attributes split up in array form.
- Returns** Highchart element object. The `Highchart.Element` class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.
- 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]);
```

## **actuate.report.HTML5Chart.Renderer.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** Highchart element object. The `Highchart.Element` class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.

**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);
```

## **actuate.report.HTML5Chart.Renderer.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);
```

## **actuate.report.HTML5Chart.Renderer.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** Highchart element object. The `Highchart.Element` class is a JavaScript wrapper for SVG elements used in the rendering layer of Highchart. For reference material for highcharts, consult the BIRT Designer Professional help.

**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 4-35 lists `actuate.report.Label` functions.

**Table 4-35** `actuate.report.Label` functions

| Function                      | Description                                            |
|-------------------------------|--------------------------------------------------------|
| <code>getBookmark()</code>    | Returns the bookmark name for this Label               |
| <code>getHtmlDom()</code>     | Returns the HTML element for this Label                |
| <code>getInstanceId()</code>  | Returns the report element instance id                 |
| <code>getLabel()</code>       | Returns the text of this Label element                 |
| <code>getPageContent()</code> | Returns the page content to which this element belongs |
| <code>getType()</code>        | Returns the Label's element type                       |
| <code>hide()</code>           | Hides this element                                     |
| <code>show()</code>           | Shows this element                                     |
| <code>submit()</code>         | Applies changes made to this gadget                    |

### **actuate.report.Label.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());
```

### **actuate.report.Label.getHtmlDom**

**Syntax** `HTMLInputElement 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);  
}
```

## **actuate.report.Label.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);  
}
```

## **actuate.report.Label.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( ));
```

## **actuate.report.Label.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);  
}
```

## **actuate.report.Label.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")  
}
```

## **actuate.report.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( );
```

## **actuate.report.Label.show**

**Syntax** void Label.show( )

Shows this element.

**Example** Use show( ) to reveal a report label, as shown in the following code:

```
myLabel.show( );
```

## **actuate.report.Label.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.



**Parameters**    **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 4-36 lists `actuate.report.Table` functions.

**Table 4-36** `actuate.report.Table` functions

| Function                      | Description                                                                                 |
|-------------------------------|---------------------------------------------------------------------------------------------|
| <code>clearFilters()</code>   | Clears the filters from the given column                                                    |
| <code>getBookmark()</code>    | Returns the bookmark name for this Table                                                    |
| <code>getColumn()</code>      | Gets the Table data by column index and returns only the data from the current visible page |
| <code>getHtmlDom()</code>     | Returns the HTML element for this Table                                                     |
| <code>getInstanceId()</code>  | Returns the report element instance id                                                      |
| <code>getPageContent()</code> | Returns the page content to which this element belongs                                      |
| <code>getRow()</code>         | Gets the Table data by row index                                                            |
| <code>getType()</code>        | Returns the report element type                                                             |
| <code>groupBy()</code>        | Adds an inner group to this Table                                                           |
| <code>hide()</code>           | Hides this element                                                                          |
| <code>hideColumn()</code>     | Hides a Table column by specifying the column name                                          |
| <code>hideDetail()</code>     | Hides detailed information for displayed groups                                             |
| <code>removeGroup()</code>    | Removes an inner group                                                                      |
| <code>setFilters()</code>     | Applies filters to this Table                                                               |
| <code>setSorters()</code>     | Adds sorters to this Table                                                                  |
| <code>show()</code>           | Shows this element                                                                          |
| <code>showColumn()</code>     | Shows a Table column by specifying the column name                                          |
| <code>showDetail()</code>     | Shows detailed information for displayed groups                                             |

**Table 4-36**      `actuate.report.Table` functions

| Function                   | Description                                                                                                                    |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <code>submit()</code>      | Submits all the asynchronous operations that the user has requested on this report and renders the Table component on the page |
| <code>swapColumns()</code> | Swaps two columns, reordering the columns                                                                                      |

## **`actuate.report.Table.clearFilters`**

**Syntax**    `void Table.clearFilters(string columnName)`

Clears the filters of a given column.

**Parameters**    **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();  
}
```

## **`actuate.report.Table.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());  
}
```

## **`actuate.report.Table.getColumn`**

**Syntax**    `array[ ] Table.getColumn(integer columnIndex)`

Gets the Table data by column index. Returns the data from the current visible page.

**Parameters**    **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( );  
}
```

## **actuate.report.Table.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);  
}
```

## **actuate.report.Table.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);  
}
```

## **actuate.report.Table.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);  
}
```

## **actuate.report.Table.getRow**

**Syntax** array[ ] Table.getRow(integer rowIndex)

Gets the Table data by row index. Returns the data from the current visible page.

**Parameters** **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( );  
}
```

## **actuate.report.Table.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( ));  
}
```

## **actuate.report.Table.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.

**Parameters** **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");  
}
```

## **actuate.report.Table.hide**

**Syntax** void Table.hide( )

Hides this element.

**Example** This example hides myTable:

```
myTable.hide( );
```

## **actuate.report.Table.hideColumn**

**Syntax** void Table.hideColumn(string columnName)

Hides a table column by specifying the column name.

**Parameters** **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( );  
}
```

## **actuate.report.Table.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.

**Parameters** **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( );  
}
```

## actuate.report.Table.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"));  
}
```

## actuate.report.Table.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);
```

## actuate.report.Table.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"]);  
}
```

## **actuate.report.Table.show**

**Syntax** void Table.show( )

Shows this element.

**Example** Use show( ) to reveal a report Table, as shown in the following code:

```
myTable.show( );
```

## **actuate.report.Table.showColumn**

**Syntax** void Table.showColumn(string columnName)

Shows the Table column by specifying the column name.

**Parameters** **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( );  
}
```

## **actuate.report.Table.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.

**Parameters** **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( );  
}
```



## actuate.report.Table.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.

**Parameters** **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"));  
}
```

## actuate.report.Table.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 4-37 lists `actuate.report.TextItem` functions.

**Table 4-37** `actuate.report.TextItem` functions

| Function                       | Description                                            |
|--------------------------------|--------------------------------------------------------|
| <code>getBookmark( )</code>    | Returns the bookmark name for this Text                |
| <code>getHtmlDom( )</code>     | Returns the HTML element for this Text                 |
| <code>getInstanceId( )</code>  | Returns the report element instance id                 |
| <code>getPageContent( )</code> | Returns the page content to which this element belongs |
| <code>getText( )</code>        | Returns the text in this Text element                  |
| <code>getType( )</code>        | Returns the Text element's type                        |
| <code>hide( )</code>           | Hides this element                                     |
| <code>show( )</code>           | Shows this element                                     |
| <code>submit( )</code>         | Applies changes made to this element                   |

### **actuate.report.TextItem.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( ));  
}
```

## actuate.report.TextItem.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);
}
```

## actuate.report.TextItem.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);
}
```

## actuate.report.TextItem.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);
}
```

## actuate.report.TextItem.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( ));
```

## actuate.report.TextItem.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");  
}
```

## actuate.report.TextItem.hide

**Syntax** `void TextItem.hide( )`  
Hides this element.

**Example** This example hides myTextItem:

```
myTextItem.hide( );  
myTextItem.submit( );
```

## actuate.report.TextItem.show

**Syntax** `void TextItem.show( )`  
Shows this element.

**Example** This example shows myTextItem:

```
myTextItem.show( );  
myTextItem.submit( );
```

## actuate.report.TextItem.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.

**Parameters**    **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.

**Parameters** **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 4-38 lists `actuate.ReportExplorer` functions.

**Table 4-38** `actuate.ReportExplorer` functions

| Function                                | Description                                                                |
|-----------------------------------------|----------------------------------------------------------------------------|
| <code>getFolderName()</code>            | Gets the root folder name                                                  |
| <code>getLatestVersionOnly()</code>     | Gets the <code>latestVersionOnly</code> flag                               |
| <code>getResultDef()</code>             | Gets the <code>resultDef</code> value for this <code>GetFolderItems</code> |
| <code>getSearch()</code>                | Gets the search value for this <code>GetFolderItems</code>                 |
| <code>onUnload()</code>                 | Unloads unused JavaScript variables                                        |
| <code>registerEventHandler()</code>     | Registers the event handler                                                |
| <code>removeEventHandler()</code>       | Removes the event handler                                                  |
| <code>setContainer()</code>             | Sets the div container                                                     |
| <code>setFolderName()</code>            | Sets the root folder name                                                  |
| <code>setLatestVersionOnly()</code>     | Sets the <code>latestVersionOnly</code> flag                               |
| <code>setResultDef()</code>             | Sets the <code>resultDef</code> value for this <code>GetFolderItems</code> |
| <code>setSearch()</code>                | Sets the search value for this <code>GetFolderItems</code>                 |
| <code>setService()</code>               | Sets the JSAPI web service                                                 |
| <code>setStartingFolder()</code>        | Sets the path for the initial folder selection                             |
| <code>setUseDescriptionAsLabel()</code> | Sets flag to use descriptions as file/folder labels                        |

**Table 4-38**      `actuate.ReportExplorer` functions

| Function                       | Description                           |
|--------------------------------|---------------------------------------|
| <code>showFoldersOnly()</code> | Sets the flag to only display folders |
| <code>submit()</code>          | Applies changes made to this element  |

## **`actuate.ReportExplorer.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());  
}
```

## **`actuate.ReportExplorer.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());  
}
```

## **`actuate.ReportExplorer.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 in an alert box:

```
function alertResultsDefinition(myReportExplorer) {  
    alert(myReportExplorer.getResultDef());  
}
```

## actuate.ReportExplorer.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( ));
```

## actuate.ReportExplorer.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( );
```

## actuate.ReportExplorer.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);
```

## actuate.ReportExplorer.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);
```

## **actuate.ReportExplorer.setContainer**

**Syntax** void ReportExplorer.setContainer(string containerId)

Sets the HTML element container for the ReportExplorer content.

**Parameters** **containerID**

String. The name of the HTML element that displays the group of rendered ReportExplorer components.

**Example** This example sets MyReportExplorer to render the <div> element labeled "History":

```
myReportExplorer.setContainer("History");
```

## **actuate.ReportExplorer.setFolderName**

**Syntax** void ReportExplorer.setFolderName(string folderName)

Sets the name of the root folder for this ReportExplorer.

**Parameters** **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.

**Example** This example sets the report explorer root folder to /Public:

```
myReportExplorer.setFolderName("/Public");
```

## **actuate.ReportExplorer.setLatestVersionOnly**

**Syntax** void ReportExplorer.setLatestVersionOnly(boolean latestVersionOnly)

Sets the latest version only flag for this ReportExplorer.

**Parameters** **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 );
```

## actuate.ReportExplorer.setResultDef

**Syntax** void ReportExplorer.setResultDef(string[] resultDef)

Sets the results definition.

**Parameters** **resultDef**

Array of strings. Valid values are: "Name", "FileType", "Version", "VersionName", "Description", "Timestamp", "Size", and "PageCount".

**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("|") );
```

## actuate.ReportExplorer.setSearch

**Syntax** void ReportExplorer.setSearch(actuate.ReportExplorer.FileSearch search)

Assigns a FileSearch object to this ReportExplorer.

**Parameters** **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( ));
```

## actuate.ReportExplorer.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 Information Console.

**requestOptions**

actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, Encyclopedia volume, or repository type. The URL can also include custom parameters.

**Example** This example sets the URL for the Actuate iPortal web application service:

```
myExplorer.setService("http://localhost:8700  
/iportal", myRequestOptions);
```

## actuate.ReportExplorer.setStartingFolder

**Syntax** void ReportExplorer.setStartingFolder(string strfoldername)

Sets the fully qualified path of the initially selected folder in the explorer tree.

**Parameters** **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:\Program Files\Actuate11\iHub2  
  \servletcontainer\iportal\WEB-INF\repository\Public");
```

## actuate.ReportExplorer.setUseDescriptionAsLabel

**Syntax** void ReportExplorer.setUseDescriptionAsLabel(boolean useDescription)

Sets the explorer to display the folder description as the folder label instead of the folder name.

**Parameters** **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);
```

## actuate.ReportExplorer.showFoldersOnly

**Syntax** void ReportExplorer.showFoldersOnly(boolean flag)

Sets ReportExplorer to display folders but not files.

**Parameters** **flag**

Boolean. True displays folders but not files.

**Example** This example displays folders in ReportExplorer but not files:

```
myExplorer.showFoldersOnly(true);
```

## actuate.ReportExplorer.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.

**Parameters** **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("/Dashboard/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 4-39 lists the constants used for the ReportExplorer class.

**Table 4-39** 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 4-40 lists the ReportExplorer event constants.

**Table 4-40** Actuate JavaScript API ReportExplorer event constants

| Event                | Description                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ON_EXCEPTION         | Event triggered when an exception occurs.<br>An event handler registered to this event must take an <code>actuate.Exception</code> object as an input argument. The <code>Exception</code> object contains the exception information.                                                                                                                                                  |
| ON_SELECTION_CHANGED | Event triggered when a selection change occurs.<br>For example, this event triggers if the value of a ReportExplorer list control changes.<br>An event handler registered to this event must take an <code>actuate.ReportExplorer.File</code> 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.<br>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 4-41 lists `actuate.reportexplorer.File` functions.

**Table 4-41** `actuate.reportexplorer.File` functions

| Function                           | Description                                               |
|------------------------------------|-----------------------------------------------------------|
| <code>getAccessType( )</code>      | Gets the <code>accessType</code> value for this File      |
| <code>getDescription( )</code>     | Gets the <code>description</code> value for this File     |
| <code>getFileType( )</code>        | Gets the <code>fileType</code> value for this File        |
| <code>getId( )</code>              | Gets the <code>id</code> value for this File              |
| <code>getName( )</code>            | Gets the <code>name</code> value for this File            |
| <code>getOwner( )</code>           | Gets the <code>owner</code> value for this File           |
| <code>getPageCount( )</code>       | Gets the <code>pageCount</code> value for this File       |
| <code>getSize( )</code>            | Gets the <code>size</code> value for this File            |
| <code>getTimeStamp( )</code>       | Gets the <code>timeStamp</code> value for this File       |
| <code>getUserPermissions( )</code> | Gets the <code>userPermissions</code> value for this File |
| <code>getVersion( )</code>         | Gets the <code>version</code> value for this File         |
| <code>getVersionName( )</code>     | Gets the <code>versionName</code> value for this File     |
| <code>setAccessType( )</code>      | Sets the <code>accessType</code> value for this File      |
| <code>setDescription( )</code>     | Sets the <code>description</code> value for this File     |
| <code>setFileType( )</code>        | Sets the <code>fileType</code> value for this File        |
| <code>setId( )</code>              | Sets the <code>id</code> value for this File              |
| <code>setName( )</code>            | Sets the <code>name</code> value for this File            |
| <code>setOwner( )</code>           | Sets the <code>owner</code> value for this File           |
| <code>setPageCount( )</code>       | Sets the <code>pageCount</code> value for this File       |
| <code>setSize( )</code>            | Sets the <code>size</code> value for this File            |

*(continues)*

**Table 4-41**      `actuate.reportexplorer.File` functions (continued)

| Function                           | Description                                               |
|------------------------------------|-----------------------------------------------------------|
| <code>setTimeStamp( )</code>       | Sets the <code>timeStamp</code> value for this File       |
| <code>setUserPermissions( )</code> | Sets the <code>userPermissions</code> value for this File |
| <code>setVersion( )</code>         | Sets the <code>version</code> value for this File         |
| <code>setVersionName( )</code>     | Sets the <code>versionName</code> value for this File     |

## **`actuate.reportexplorer.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;}
```

## **`actuate.reportexplorer.File.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;}
```

## **`actuate.reportexplorer.File.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( );
```

## **`actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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;}
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.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( );
```

## **actuate.reportexplorer.File.setAccessType**

**Syntax** void File.setAccessType(string accessType)

Sets the access type.

**Parameters** **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")
```

## **actuate.reportexplorer.File.setDescription**

**Syntax** void File.setDescription(string description)

Sets the description from the file.

**Parameters** **description**

String. The description.

**Example** To clear a file's description, use code similar to the following:

```
file.setDescription("");
```

## **actuate.reportexplorer.File.setFileType**

**Syntax** void File.setFileType(string fileType)

Sets the file type for this file.

**Parameters** **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");}
```

## **actuate.reportexplorer.File.setId**

**Syntax** void File.setId(integer id)

Sets the file ID value.

**Parameters** **id**

Integer. A file ID number.

**Example** To set a file's ID to 42, use code similar to the following:

```
file.setId("42");
```

## **actuate.reportexplorer.File.setName**

**Syntax** void File.setName(string name)

Sets the name of the file.

**Parameters** **name**  
String. The name.

**Example** To set a file's name to releasedates, use code similar to the following:

```
file.setName("releasedates");
```

## **actuate.reportexplorer.File.setOwner**

**Syntax** void File.setOwner(string owner)

Sets the name of the owner.

**Parameters** **owner**  
String. A user name.

**Example** To set a file's owner to Administrator, use code similar to the following:

```
file.setOwner("Administrator");
```

## **actuate.reportexplorer.File.setPageCount**

**Syntax** void File.setPageCount(integer pageCount)

Sets the number pages in the file.

**Parameters** **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);}
```

## **actuate.reportexplorer.File.setSize**

**Syntax** void File.setSize(integer size)

Sets the size of the file.

**Parameters** **size**  
Integer. File size in bytes.

**Example** To set a file's size to 0, use code similar to the following:

```
file.setSize(0);
```

## **actuate.reportexplorer.File.setTimeStamp**

**Syntax** void File.setTimeStamp(string timeStamp)

Sets the time stamp.

**Parameters** **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( ));
```

## **actuate.reportexplorer.File.setUserPermissions**

**Syntax** void File.setUserPermissions(string userPermissions)

Sets the user permissions.

**Parameters** **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( ));
```

## **actuate.reportexplorer.File.setVersion**

**Syntax** void File.setVersion(integer version)

Sets the version of the file.

**Parameters** **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);
```

## **actuate.reportexplorer.File.setVersionName**

**Syntax** void File.setVersionName(string versionName)

Sets the version name.

**Parameters** **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 in `actuate.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 4-42 lists `actuate.reportexplorer.FileCondition` functions.

**Table 4-42** `actuate.reportexplorer.FileCondition` functions

| Function                 | Description                                              |
|--------------------------|----------------------------------------------------------|
| <code>getField( )</code> | Gets the field for this <code>FileCondition</code>       |
| <code>getMatch( )</code> | Gets the match value for this <code>FileCondition</code> |
| <code>setField( )</code> | Sets the field for this <code>FileCondition</code>       |
| <code>setMatch( )</code> | Sets the match value for this <code>FileCondition</code> |

### **actuate.reportexplorer.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( );
```

### **actuate.reportexplorer.FileCondition.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( );
```

## **actuate.reportexplorer.FileCondition.setField**

**Syntax** void FileCondition.setField(string field)

Sets the field for the FileCondition.

**Parameters** **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");
```

## **actuate.reportexplorer.FileCondition.setMatch**

**Syntax** void FileCondition.setMatch(string match)

Sets the match value for the FileCondition.

**Parameters** **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 4-43 lists `actuate.reportexplorer.FileSearch` functions.

**Table 4-43** `actuate.reportexplorer.FileSearch` functions

| Function                               | Condition                                                           |
|----------------------------------------|---------------------------------------------------------------------|
| <code>getAccessType( )</code>          | Gets the <code>accessType</code> value for this FileSearch          |
| <code>getCondition( )</code>           | Gets the <code>condition</code> value for this FileSearch           |
| <code>getConditionArray( )</code>      | Gets the <code>ConditionArray</code> value for this FileSearch      |
| <code>getCountLimit( )</code>          | Gets the <code>countLimit</code> value for this FileSearch          |
| <code>getDependentFileId( )</code>     | Gets the <code>id</code> value for this FileSearch                  |
| <code>getDependentFileName( )</code>   | Gets the <code>file name</code> value for this FileSearch           |
| <code>getFetchDirection( )</code>      | Gets the <code>fetch direction</code> for this FileSearch           |
| <code>getFetchHandle( )</code>         | Gets the <code>fetchHandle</code> value for this FileSearch         |
| <code>getFetchSize( )</code>           | Gets the <code>fetchSize</code> value for this FileSearch           |
| <code>getIncludeHiddenObject( )</code> | Gets the <code>includeHiddenObject</code> value for this FileSearch |
| <code>getOwner( )</code>               | Gets the <code>owner</code>                                         |
| <code>getPrivilegeFilter( )</code>     | Gets the <code>privilegeFilter</code> value for this FileSearch     |
| <code>getRequiredFileId( )</code>      | Gets the <code>requiredFileId</code> for this FileSearch            |
| <code>getRequiredFileName( )</code>    | Gets the <code>requiredFileName</code> value for this FileSearch    |
| <code>setAccessType( )</code>          | Sets the <code>accessType</code> value for this FileSearch          |
| <code>setCondition( )</code>           | Sets the <code>condition</code> value for this FileSearch           |
| <code>setConditionArray( )</code>      | Sets the <code>ConditionArray</code> value for this FileSearch      |

*(continues)*

**Table 4-43**      `actuate.reportexplorer.FileSearch` functions (continued)

| Function                               | Condition                                           |
|----------------------------------------|-----------------------------------------------------|
| <code>setCountLimit( )</code>          | Sets the id value for this FileSearch               |
| <code>setDependentFileId( )</code>     | Sets the id value for this FileSearch               |
| <code>setDependentFileName( )</code>   | Sets the file name value for this FileSearch        |
| <code>setFetchDirection( )</code>      | Sets the owner value for this FileSearch            |
| <code>setFetchHandle( )</code>         | Sets the fetchHandle value for this FileSearch      |
| <code>setFetchSize( )</code>           | Sets the fetchSize value for this FileSearch        |
| <code>setIncludeHiddenObject( )</code> | Sets the timeStamp value for this FileSearch        |
| <code>setOwner( )</code>               | Sets the Owner                                      |
| <code>setPrivilegeFilter( )</code>     | Sets the PrivilegeFilter value for this FileSearch  |
| <code>setRequiredFileId( )</code>      | Sets the requiredFileId for this FileSearch         |
| <code>setRequiredFileName( )</code>    | Sets the requiredFileName value for this FileSearch |

## **`actuate.reportexplorer.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;}
```

## **`actuate.reportexplorer.FileSearch.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;}
```

## **`actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch .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( );
```

## **actuate.reportexplorer.FileSearch.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( ));
```

## **actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch.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;}
```

## **actuate.reportexplorer.FileSearch .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.getIncludeHiddenObject( )) {  
    alert("Hidden objects are enabled.");  
}
```

## **actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch.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( );
```

## **actuate.reportexplorer.FileSearch .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( );
```

## **actuate.reportexplorer.FileSearch.setAccessType**

**Syntax** void FileSearch.setAccessType(string accessType)

Sets the access type.

**Parameters** **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");
```

## **actuate.reportexplorer.FileSearch.setCondition**

**Syntax** void FileSearch.setCondition(actuate.reportexplorer.FileCondition condition)

Sets a search condition for this FileSearch.

**Parameters** **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);
```

## **actuate.reportexplorer.FileSearch.setConditionArray**

**Syntax** void FileSearch.setConditionArray(actuate.reportexplorer.FileCondition[ ] ConditionArray)

Sets multiple search conditions for this FileSearch.

**Parameters** **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);
```

## **actuate.reportexplorer.FileSearch.setCountLimit**

**Syntax** void FileSearch.setCountLimit(integer countlimit)

Sets the maximum number of match results to display.

**Parameters** **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);
```

## **actuate.reportexplorer.FileSearch.setDependentFileId**

**Syntax** void FileSearch.setDependentFileId(string dependentFileId)  
Sets the file ID of the FileSearch.

**Parameters** **dependentFileId**  
String. A file ID.

**Example** To set FileSearch fsearch's File ID to current, use code similar to the following:

```
fsearch.setDependentFileId("current");
```

## **actuate.reportexplorer.FileSearch.setDependentFileName**

**Syntax** void FileSearch.setDependentFileName(string dependentFileName)  
Sets the file name of FileSearch.

**Parameters** **dependentFileName**  
String. A file name.

**Example** To set FileSearch fsearch's file name to current, use code similar to the following:

```
fsearch.setDependentFileName("current");
```

## **actuate.reportexplorer.FileSearch.setFetchDirection**

**Syntax** void FileSearch.setFetchDirection(boolean fetchDirection)  
Sets the fetch direction for this FileSearch.

**Parameters** **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());
```

## **actuate.reportexplorer.FileSearch.setFetchHandle**

**Syntax** void FileSearch.setFetchHandle(string fetchHandle)

Sets the fetch handle for FileSearch.

**Parameters**    **fetchHandle**  
String. A fetch handle.

**Example**    To set FileSearch fsearch's fetch handle to ezsearch, use code similar to the following:

```
fsearch.setFetchHandle("ezsearch");
```

## **actuate.reportexplorer.FileSearch.setFetchSize**

**Syntax**    void FileSearch.setFetchSize(integer fetchSize)

Sets the fetch size.

**Parameters**    **fetchSize**  
Integer. The fetch size.

**Example**    To set FileSearch fsearch's fetch size to 12, use code similar to the following:

```
fsearch.setFetchSize(12);
```

## **actuate.reportexplorer.FileSearch .setIncludeHiddenObject**

**Syntax**    void FileSearch.setIncludeHiddenObject(boolean includeHiddenObject)

Sets the includeHiddenObject value for this FileSearch.

**Parameters**    **includeHiddenObject**  
Boolean. True includes hidden object.

**Example**    To prohibit FileSearch fsearch from including hidden objects, use code similar to the following:

```
fsearch.setIncludeHiddenObject(false);
```

## **actuate.reportexplorer.FileSearch.setOwner**

**Syntax**    void FileSearch.setOwner(string owner)

Sets the owner for this FileSearch.

**Parameters**    **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");
```



## **actuate.reportexplorer.FileSearch.setPrivilegeFilter**

**Syntax** void FileSearch.setPrivilegeFilter(actuate.reportexplorer.PrivilegeFilter  
privilegeFilter)

Sets the privilege filter.

**Parameters** **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);
```

## **actuate.reportexplorer.FileSearch.setRequiredFileId**

**Syntax** void FileSearch.setRequiredFileId(string requiredFileId)

Sets the requiredFileId for this FileSearch.

**Parameters** **requiredFileId**  
String. A file ID.

**Example** To set FileSearch fsearch's File ID to permanent, use code similar to the following:

```
fsearch.setRequiredFileId("permanent");
```

## **actuate.reportexplorer.FileSearch .setRequiredFileName**

**Syntax** void FileSearch.setRequiredFileName(string requiredFileName)

Sets the required file name.

**Parameters** **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 4-44 lists `actuate.reportexplorer.FolderItems` functions.

**Table 4-44** `actuate.reportexplorer.FolderItems` functions

| Function                       | Description                                                                     |
|--------------------------------|---------------------------------------------------------------------------------|
| <code>getFetchHandle( )</code> | Gets the <code>fetchHandle</code> value for <code>GetFolderItemsResponse</code> |
| <code>getItemList( )</code>    | Gets the <code>itemList</code> value for <code>GetFolderItemsResponse</code>    |
| <code>getTotalCount( )</code>  | Gets the <code>totalCount</code> value for <code>GetFolderItemsResponse</code>  |
| <code>setFetchHandle( )</code> | Sets the <code>fetchHandle</code> value for <code>GetFolderItemsResponse</code> |
| <code>setItemList( )</code>    | Sets the <code>itemList</code> value for <code>GetFolderItemsResponse</code>    |
| <code>setTotalCount( )</code>  | Sets the <code>totalCount</code> value for <code>GetFolderItemsResponse</code>  |

### **actuate.reportexplorer.FolderItems.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( );
```

### **actuate.reportexplorer.FolderItems.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( );
```

## **actuate.reportexplorer.FolderItems.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( );
```

## **actuate.reportexplorer.FolderItems.setFetchHandle**

**Syntax** void FolderItems.setFetchHandle(string fetchHandle)

Sets the fetch handle value for this FolderItems object.

**Parameters** **fetchHandle**

String. The fetch handle.

**Example** To set FolderItems fitems' fetch handle to dir, use code similar to the following:

```
fitems.setFetchHandle("dir");
```

## **actuate.reportexplorer.FolderItems.setItemList**

**Syntax** void FolderItems.setItemList(ctuate.reportexplorer.File[ ] itemList)

Sets the list of contents for this folder.

**Parameters** **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( ));
```

## **actuate.reportexplorer.FolderItems.setTotalCount**

**Syntax** void FolderItems.setDataTypes(string totalCount)

Sets the maximum number of list items to retrieve from this folder.

**Parameters** **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 4-45 lists `actuate.reportexplorer.PrivilegeFilter` functions.

**Table 4-45** `actuate.reportexplorer.PrivilegeFilter` functions

| Function                           | Description                                                          |
|------------------------------------|----------------------------------------------------------------------|
| <code>getAccessRights( )</code>    | Gets the <code>accessRights</code> value for this PrivilegeFilter    |
| <code>getGrantedRoleId( )</code>   | Gets the <code>grantedRoleId</code> value for this PrivilegeFilter   |
| <code>getGrantedRoleName( )</code> | Gets the <code>grantedRoleName</code> value for this PrivilegeFilter |
| <code>getGrantedUserId( )</code>   | Gets the <code>grantedUserId</code> value for this PrivilegeFilter   |
| <code>getGrantedUserName( )</code> | Gets the <code>grantedUserName</code> value for this PrivilegeFilter |
| <code>setAccessRights( )</code>    | Sets the <code>accessRights</code> value for this PrivilegeFilter    |
| <code>setGrantedRoleId( )</code>   | Sets the <code>grantedRoleId</code> value for this PrivilegeFilter   |
| <code>setGrantedRoleName( )</code> | Sets the <code>grantedRoleName</code> value for this PrivilegeFilter |
| <code>setGrantedUserId( )</code>   | Sets the <code>grantedUserId</code> value for this PrivilegeFilter   |
| <code>setGrantedUserName( )</code> | Sets the <code>grantedUserName</code> value for this PrivilegeFilter |

## **actuate.reportexplorer.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;}
```

## **actuate.reportexplorer.PrivilegeFilter .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( );
```

## **actuate.reportexplorer.PrivilegeFilter .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( );
```

## **actuate.reportexplorer.PrivilegeFilter .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( );
```

## **actuate.reportexplorer.PrivilegeFilter .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( );
```

## **actuate.reportexplorer.PrivilegeFilter .setAccessRights**

**Syntax** void PrivilegeFilter.setAccessRights(string accessRights)

Sets the repository access rights value for this PrivilegeFilter.

**Parameters** **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( ));
```

## **actuate.reportexplorer.PrivilegeFilter .setGrantedRoleId**

**Syntax** void PrivilegeFilter.setGrantedRoleId(string grantedRoleId)

Sets the grantedRoleId of the column for this PrivilegeFilter.

**Parameters** **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");
```

## **actuate.reportexplorer.PrivilegeFilter .setGrantedRoleName**

**Syntax** void PrivilegeFilter.setGrantedRoleName(string grantedRoleName)

Sets the grantedRoleName value for this PrivilegeFilter.

**Parameters** **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");
```

## **actuate.reportexplorer.PrivilegeFilter .setGrantedUserId**

**Syntax** void PrivilegeFilter.setGrantedUserId(string grantedUserId)

Sets the grantedUserId value for this PrivilegeFilter.

**Parameters** **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");
```

## **actuate.reportexplorer.PrivilegeFilter .setGrantedUserName**

**Syntax** void PrivilegeFilter.setGrantedUserName(string grantedUserName)

Sets the grantedUserName value for this PrivilegeFilter.

**Parameters** **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.

**Parameters** **requestOptions**  
actuate.RequestOptions object. Optional. Provides request option settings to copy into this RequestOptions object.

### Function summary

Table 4-46 lists actuate.RequestOptions functions.

**Table 4-46** actuate.RequestOptions functions

| Function                           | Description                                         |
|------------------------------------|-----------------------------------------------------|
| <code>getServerUrl()</code>        | Returns the BIRT iHub URL value                     |
| <code>getLocale()</code>           | Returns the current locale                          |
| <code>getRepositoryType()</code>   | Returns the repository type                         |
| <code>getVolume()</code>           | Returns the Encyclopedia volume                     |
| <code>getVolumeProfile()</code>    | Returns the volume profile                          |
| <code>setCustomParameters()</code> | Appends a custom report explorer to the request URL |
| <code>setServerUrl()</code>        | Sets the BIRT iHub URL value                        |
| <code>setLocale()</code>           | Sets the locale                                     |
| <code>setRepositoryType()</code>   | Sets the repository type: enterprise or workgroup   |
| <code>setVolume()</code>           | Sets the Encyclopedia volume                        |
| <code>setVolumeProfile()</code>    | Sets the volume profile                             |

### **actuate.RequestOptions.getServerUrl**

**Syntax** `string RequestOptions.getServerurl()`

Returns the BIRT iHub URL.



**Returns** String. The URL for BIRT iHub.

**Example** To retrieve the BIRT iHub URL from the RequestOptions object reqOpts, use code similar to the following:

```
var iHubUrl = reqOpts.getServerUrl( );
```

## **actuate.RequestOptions.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");  
}
```

## **actuate.RequestOptions.getRepositoryType**

**Syntax** string RequestOptions.getRepositoryType( )

Returns the repository type: enterprise or workgroup.

**Returns** String. Valid repository type values are enterprise or workgroup.

**Example** To retrieve the repository type for the RequestOptions object reqOpts, use code similar to the following:

```
var repositorytype = reqOpts.getRepositoryType( );
```

## **actuate.RequestOptions.getVolume**

**Syntax** string RequestOptions.getVolume( )

Returns the Encyclopedia volume.

**Returns** String. The name of the Encyclopedia volume.

**Example** To retrieve the Encyclopedia volume for the RequestOptions object reqOpts, use code similar to the following:

```
var encyVol = reqOpts.getVolume( );
```

## **actuate.RequestOptions.getVolumeProfile**

**Syntax** string RequestOptions.getVolumeProfile( )

Returns the volume profile by name. Valid volume profile names are listed in the service's WEB-INF\volumeProfile.xml file.

**Returns** String. The volume profile.

**Example** To retrieve the volume profile for the RequestOptions object reqOpts, use code similar to the following:

```
var volProfile = reqOpts.getVolumeProfile( );
```

## **actuate.RequestOptions.setCustomParameters**

**Syntax** void RequestOptions.setCustomParameters(object parameters)

Returns a custom parameter in the request URL.

**Parameters** **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"});
```

## **actuate.RequestOptions.setIserverUrl**

**Syntax** void RequestOptions.setIserverUrl(string iServerUrl)

Sets the BIRT iHub URL.

**Parameters** **Iserverurl**  
String. The BIRT iHub URL value.

**Example** This example sets the BIRT iHub URL for the reqOpts RequestOptions object:

```
reqOpts.setIserverUrl("http://localhost:8700");
```

## **actuate.RequestOptions.setLocale**

**Syntax** void RequestOptions.setLocale(string Locale)

Sets the locale.

**Parameters** **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:

```
reqOpts.setLocale( );
```

## actuate.RequestOptions.setRepositoryType

**Syntax** void RequestOptions.setRepositoryType(string repositoryType)

Sets the repository type, either enterprise or workgroup.

**Parameters** **repositoryType**

String. Valid repository type values are enterprise or standalone, as designated by the Actuate web application service with which to connect. Use the following constants:

- actuate.RequestOptions.REPOSITORY\_ENCYCLOPEDIA
- actuate.RequestOptions.REPOSITORY\_STANDALONE

**Example** This example sets the repository to workgroup:

```
reqOpts.setRepositoryType(  
    actuate.RequestOptions.REPOSITORY_STANDALONE);
```

## actuate.RequestOptions.setVolume

**Syntax** void RequestOptions.setVolume(string volume)

Sets the Encyclopedia volume.

**Parameters** **volume**

String. The Encyclopedia volume.

**Example** To set the Encyclopedia volume to marcom if the RequestOptions object reqOpts volume is null, use code similar to the following:

```
if( reqOpts.getVolume( ) == null){  
    reqOpts.setVolume("marcom");  
}
```

## actuate.RequestOptions.setVolumeProfile

**Syntax** void RequestOptions.setVolumeProfile(string volumeProfile)

Sets the volume profile to use. Valid volume profile names are listed in the service's WEB-INF\volumeProfile.xml file.

**Parameters** **volumeProfile**

String. The volume profile.

**Example** To set the volume profile to myServer if the RequestOptions object reqOpts volume profile is null, use code similar to the following:

```
if( reqOpts.getVolume( ) == null){  
    reqOpts.setVolumeProvile("myServer");  
}
```

---

## 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 `<div>` element that holds the viewer.

**viewContainerId**

String. The value of the `id` parameter for the `<div>` element that holds the viewer.

**Example** To assign the viewer to display in a `<div id='containerName' />` tag on the page, use the following constructor call:

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

### Function summary

Table 4-47 lists `actuate.Viewer` functions.

**Table 4-47** `actuate.Viewer` functions

| Function                             | Description                                 |
|--------------------------------------|---------------------------------------------|
| <code>disableIV()</code>             | Disables Interactive Viewer features        |
| <code>downloadReport()</code>        | Exports a report using the specified format |
| <code>downloadResultSet()</code>     | Exports data to an external file            |
| <code>enableIV()</code>              | Enables Interactive Viewer features         |
| <code>getChart()</code>              | Retrieves a chart by bookmark               |
| <code>getClientHeight()</code>       | Gets the viewer's height                    |
| <code>getClientWidth()</code>        | Gets the viewer's width                     |
| <code>getContentByBookmark()</code>  | Gets the report content by bookmark         |
| <code>getContentByPageRange()</code> | Gets the report content by page range       |

**Table 4-47**      **actuate.Viewer functions (continued)**

| <b>Function</b>                       | <b>Description</b>                                                |
|---------------------------------------|-------------------------------------------------------------------|
| <code>getContentMargin( )</code>      | Gets the margin dimensions of the content in pixels               |
| <code>getCurrentPageContent( )</code> | Returns the report content displayed in the viewer                |
| <code>getCurrentPageNum( )</code>     | Returns the current page number                                   |
| <code>getDataItem( )</code>           | Retrieves a data item by bookmark                                 |
| <code>getFlashObject( )</code>        | Retrieves a Flash object by bookmark                              |
| <code>getGadget( )</code>             | Retrieves a gadget by bookmark                                    |
| <code>getHeight( )</code>             | Returns the viewer height setting                                 |
| <code>getHelpBase( )</code>           | Gets the help URL                                                 |
| <code>getId( )</code>                 | Returns the ID of this object                                     |
| <code>getIportalUrl( )</code>         | Returns the Actuate web application URL that this Viewer accesses |
| <code>getLabel( )</code>              | Retrieves a label by bookmark                                     |
| <code>getReportletBookmark( )</code>  | Returns the bookmark of a Reportlet displayed in the viewer       |
| <code>getReportName( )</code>         | Returns the report file displayed in the viewer                   |
| <code>getRequestOptions( )</code>     | Returns the viewer's request options                              |
| <code>getTable( )</code>              | Retrieves a table by bookmark                                     |
| <code>getText( )</code>               | Retrieves a text element by bookmark                              |
| <code>getTotalPageCount( )</code>     | Returns the total number of pages                                 |
| <code>getUIConfig( )</code>           | Gets the UIConfig object assigned to the viewer                   |
| <code>getUIOptions( )</code>          | Returns the UIOptions object                                      |
| <code>getViewer( )</code>             | Returns a viewer object containing the given bookmarked element   |
| <code>getWidth( )</code>              | Returns the viewer width setting                                  |
| <code>gotoBookmark( )</code>          | Goes to the position in the report specified by the bookmark      |
| <code>gotoPage( )</code>              | Goes to the specified page                                        |
| <code>isInteractive( )</code>         | Returns whether interactive viewing features are enabled          |

*(continues)*

**Table 4-47**      `actuate.Viewer` functions (continued)

| Function                                   | Description                                                                                                                     |
|--------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <code>saveReportDesign()</code>            | Saves a report design to the repository                                                                                         |
| <code>saveReportDocument()</code>          | Saves a report document to the repository                                                                                       |
| <code>setContentMargin()</code>            | Sets the viewer content margin                                                                                                  |
| <code>setFocus()</code>                    | Sets the focus element on the viewer                                                                                            |
| <code>setHeight()</code>                   | Sets the viewer height                                                                                                          |
| <code>setHelpBase()</code>                 | Sets the base help URL                                                                                                          |
| <code>setParameters()</code>               | Sets the parameters to run a report using a list of literal string pairs                                                        |
| <code>setParameterValues()</code>          | Sets the parameters to run a report using a generated object                                                                    |
| <code>setReportletBookmark()</code>        | Sets bookmark name for a Reportlet                                                                                              |
| <code>setReportName()</code>               | Sets the report file to render within this Viewer                                                                               |
| <code>setService()</code>                  | Sets the target service URL                                                                                                     |
| <code>setSize()</code>                     | Sets the size of the viewer                                                                                                     |
| <code>setSupportSVG()</code>               | Sets the Scalable Vector Graphic support flag to enable Scalable Vector Graphics content                                        |
| <code>setUIOptions()</code>                | Sets UIOptions using a UIOptions object                                                                                         |
| <code>setViewingMode()</code>              | Sets the dashboard viewing mode                                                                                                 |
| <code>setWidth()</code>                    | Sets the width of the viewer                                                                                                    |
| <code>showDownloadReportDialog()</code>    | Enables the export report dialog window                                                                                         |
| <code>showDownloadResultSetDialog()</code> | Enables the download data dialog window                                                                                         |
| <code>showFacebookCommentPanel()</code>    | Shows the Facebook comments panel.                                                                                              |
| <code>showParameterPanel()</code>          | Shows the parameter panel                                                                                                       |
| <code>showPrintDialog()</code>             | Enables the print dialog window                                                                                                 |
| <code>showTocPanel()</code>                | Shows the table of contents panel                                                                                               |
| <code>submit()</code>                      | Submits all the asynchronous operations that the user has requested on this Viewer and renders the viewer component on the page |

## **`actuate.Viewer.disableIV`**

**Syntax**    `void Viewer.disableIV(function callback)`

Disables the Interactive Viewer features of this viewer object. This is an asynchronous setting committed by `submit()`.

**Parameters**    **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");});
```

## **actuate.Viewer.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);
```

## **actuate.Viewer.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);
```

## **actuate.Viewer.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);});  
}
```

**Parameters** **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);});  
}
```



## actuate.Viewer.getChart

- Syntax** `actuate.report.Chart Viewer.getChart(string bookmark)`  
Returns an instance of the chart referenced by a bookmark.
- Parameters** **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");  
}
```

## actuate.Viewer.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);  
}
```

## actuate.Viewer.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);  
}
```

## actuate.Viewer.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);
```

## actuate.Viewer.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);
```

## actuate.Viewer.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( ));
```

## **actuate.Viewer.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");
```

## **actuate.Viewer.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);
```

## **actuate.Viewer.getDataItem**

**Syntax** `actuate.report.DataItem Viewer.getDataItem(string bookmark)`

Returns an instance of report data referenced by a bookmark.

**Parameters** **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");
```

## actuate.Viewer.getFlashObject

**Syntax** `actuate.report.FlashObject Viewer.getFlashObject(string bookmark)`

Returns an instance of the Flash object referenced by a bookmark.

**Parameters** **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");
```

## actuate.Viewer.getGadget

**Syntax** `actuate.report.Gadget Viewer.getGadget(string bookmark)`

Returns an instance of the gadget referenced by a bookmark.

**Parameters** **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");
```

## actuate.Viewer.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);
```

## actuate.Viewer.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( ))
```

## **actuate.Viewer.getLabel**

**Syntax** `actuate.report.Label Viewer.getLabel(string bookmark)`  
Returns an instance of the label referenced by a bookmark.

**Parameters** **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");
```

## **actuate.Viewer.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( ));
```

## **actuate.Viewer.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( ));
```

## **actuate.Viewer.getTable**

**Syntax** `actuate.report.Table Viewer.getTable(string bookmark)`  
Returns an instance of the table referenced by a bookmark.

**Parameters** **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");
```

## **actuate.Viewer.getText**

**Syntax** actuate.report.Text Viewer.getText(string bookmark)

Returns an instance of the Text object referenced by a bookmark.

**Parameters** **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");
```

## **actuate.Viewer.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( ));
```

## **actuate.Viewer.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( );
```

## **actuate.Viewer.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( );
```

## **actuate.Viewer.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");  
}
```

## **actuate.Viewer.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);
```

## **actuate.Viewer.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.

**Parameters** **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);
```

## **actuate.Viewer.gotoPage**

**Syntax** void Viewer.gotoPage(integer pageNumber)

Goes to the specified page. The viewer throws an exception when the page is not found.

**Parameters** **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);
```

## **actuate.Viewer.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());
```

## **actuate.Viewer.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");
```

## **actuate.Viewer.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");
```

## **actuate.Viewer.setContentMarg**

**Syntax** void Viewer.setContentMargin(string[] margin)

void Viewer.setContentMargin(int margin)

Sets the viewer content margin.

**Parameters** **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);
```

## **actuate.Viewer.setFocus**

**Syntax** void setFocus(boolean focus)

Sets the focus for the viewer.

**Parameters** **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);
```

## **actuate.Viewer.setHeight**

**Syntax** void Viewer.setHeight(integer height)

Sets the viewer height.

**Parameters** **height**

Integer. The height in pixels.

**Example** To set the height of the viewer to 600 pixels, use the following code:

```
viewer.setHeight(600);
```

## **actuate.Viewer.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.

**Parameters** **helpBase**

String. The URL where the help documentation is located.

**Example** This example sets the help base URL to `http://www.actuate.com/documentation/R11`:

```
myViewer.setHelpBase("http://www.actuate.com/documentation/R11");  
myViewer.submit();
```

## **actuate.Viewer.setParameters**

**Syntax** void Viewer.setParameters(string[] params)

Sets parameters for executing report using literal string pairs.

**Parameters** **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"});
```

## **actuate.Viewer.setParameterValues**

**Syntax** void Viewer.setParameterValues(actuate.parameter.ParameterValue[] parameters)

Sets parameter values for executing a report using ParameterValue objects.

**Parameters**    **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);
```

## **actuate.Viewer.setReportletBookmark**

**Syntax**    `void Viewer.setReportletBookmark(string bookmark)`

Sets the bookmark for the Reportlet rendered.

**Parameters**    **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();
```

## **actuate.Viewer.setReportName**

**Syntax**    `void Viewer.setReportName(string reportFile)`

Sets the report file, either a report design or report document, to render in this Viewer.

**Parameters**    **reportFile**

String. The report file path for a report design file or report document file.

**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();
```

## actuate.Viewer.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 Information Console.

**requestOptions**

actuate.RequestOptions object. Optional. requestOptions defines URL parameters to send with the authentication request, such as the iHub URL, Encyclopedia volume, or repository type. The URL can also include custom parameters.

**Example** This example sets the URL for the Actuate iPortal web application service:

```
myViewer.setService("http://localhost:8700/iportal",  
    myRequestOptions);
```

## actuate.Viewer.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);
```

## actuate.Viewer.setSupportSVG

**Syntax** void Viewer.setSupportSVG(boolean usvgFlag)

Controls Scalable Vector Graphics support for the viewer.

**Parameters** **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);
```

## **actuate.Viewer.setUIOptions**

**Syntax** void Viewer.setUIOptions(actuate.viewer.UIOptions options)

Sets the UI options for the viewer using an actuate.viewer.UIOptions object.

**Parameters** **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();
```

## **actuate.Viewer.setViewingMode**

**Syntax** void Viewer.setViewingMode(string viewer)

Sets the dashboard viewing mode.

**Parameters** **viewer**  
actuate.Constant.ViewingMode constant. Legal values are NON\_DASHBOARD, DASHBOARD\_NORMAL, and DASHBOARD\_MAX.

**Example** To display content without dashboard features, use the following code:

```
viewer.setViewingMode(actuate.Constant.ViewingMode.NON_DASHBOARD);
```

## **actuate.Viewer.setWidth**

**Syntax** void Viewer.setWidth(string width)

Sets the viewer width.

**Parameters** **width**  
String.

**Example** To set the width of the viewer to 800 pixels, use the following code:

```
viewer.setWidth(800);
```

## **actuate.Viewer.showDownloadReportDialog**

**Syntax** void Viewer.showDownloadReportDialog()

Displays the export report dialog window.

**Example** Use this code to display the report dialog window:

```
myViewer.showDownloadReportDialog( );
```

## **actuate.Viewer.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( );
```

## **actuate.Viewer.showFacebookCommentPanel**

**Syntax** void Viewer.showFacebookCommentPanel( )

Displays the Facebook comments panel.

**Example** Use this code to display the Facebook comments panel:

```
viewer.showFacebookCommentPanel( );
```

## **actuate.Viewer.showParameterPanel**

**Syntax** void Viewer.showParameterPanel( )

Displays the parameter panel.

**Example** Use this code to display the parameter panel:

```
viewer.showParameterPanel( );
```

## **actuate.Viewer.showPrintDialog**

**Syntax** void Viewer.showPrintDialog( )

Displays the print dialog window.

**Example** Use this code to display the print dialog window:

```
viewer.showPrintDialog( );
```

## **actuate.Viewer.showTocPanel**

**Syntax** void Viewer.showTocPanel( )

Displays the table of contents panel.

**Example** Use this code to display the table of contents panel:

```
viewer.showTocPanel( );
```

## actuate.Viewer.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 4-48 lists the viewer event constants.

**Table 4-48** Actuate JavaScript API viewer event constants

Event	Description
ON_CONTENT_CHANGED	<p>Calls a registered event handler when the report content is reloaded.</p> <p>The event handler must take the viewer instance that fired the event as an input argument.</p>
ON_CONTENT_SELECTED	<p>Calls a registered event handler when the relevant part of the report content is selected. Supported selected contents are:</p> <ul style="list-style-type: none"><li>■ Column</li><li>■ Table</li><li>■ Data</li><li>■ Label</li><li>■ Text</li></ul> <p>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, <code>actuate.Viewer.Table</code> is passed into the event handler.</p> <p>The event handler must take the viewer instance that fired the event and an instance of <code>actuate.viewer.SelectedContent</code> as input arguments.</p>
ON_DIALOG_OK	<p>This event fires when the user clicks the OK button in a dialog.</p> <p>The event handler must take the viewer object that fired the event and a <code>dialog.AdvancedFilterDialog</code> object as input parameters.</p>
ON_EXCEPTION	<p>An exception event is broadcast when an error occurs.</p> <p>The event handler must take the viewer instance that fired the event and an instance of <code>actuate.viewer.Exception</code> as input arguments.</p>
ON_SESSION_TIMEOUT	<p>Calls a registered event handler when the session expires.</p> <p>The event handler must take the viewer instance that fired the event as an input argument.</p>

---

---

## 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 4-49 lists `actuate.viewer.PageContent` functions.

**Table 4-49** `actuate.viewer.PageContent` functions

Function	Description
<code>getChartByBookmark()</code>	Returns a chart element specified by the given bookmark
<code>getDataItemByBookmark()</code>	Returns a data element specified by the given bookmark
<code>getFlashObjectByBookmark()</code>	Returns a Flash object specified by the given bookmark
<code>getGadgetByBookmark()</code>	Returns a Flash gadget specified by the given bookmark
<code>getLabelByBookmark()</code>	Returns a label element specified by the given bookmark
<code>getTableByBookmark()</code>	Returns a table element specified by the given bookmark
<code>getTextByBookmark()</code>	Returns a text element specified by the given bookmark
<code>getViewerId()</code>	Returns the viewer ID

### **`actuate.viewer.PageContent.getChartByBookmark`**

**Syntax** `actuate.report.Chart PageContent.getChartByBookmark(string bookmark)`

Returns the chart element specified by the given bookmark.

**Parameters** **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( );
}
```

## **actuate.viewer.PageContent.getDataItemByBookmark**

**Syntax** actuate.report.DataItem PageContent.getDataItemByBookmark(string bookmark)

Returns the data element specified by the given bookmark.

**Parameters** **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");
```

## **actuate.viewer.PageContent.getFlashObjectByBookmark**

**Syntax** actuate.report.FlashObject PageContent.getFlashObjectByBookmark(string bookmark)

Returns the Flash object specified by the given bookmark.

**Parameters** **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");
```

## actuate.viewer.PageContent.getGadgetByBookmark

**Syntax** actuate.report.Gadget PageContent.getGadgetByBookmark(string bookmark)

Returns the gadget element specified by the given bookmark.

**Parameters** **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");
```

## actuate.viewer.PageContent.getLabelByBookmark

**Syntax** actuate.report.Label PageContent.getLabelByBookmark(string bookmark)

Returns the label element specified by the given bookmark.

**Parameters** **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");
```

## actuate.viewer.PageContent.getTableByBookmark

**Syntax** actuate.report.Table PageContent.getTableByBookmark(string bookmark)

Returns the table element specified by the given bookmark.

**Parameters** **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");
```

## **actuate.viewer.PageContent.getTextByBookmark**

**Syntax** `actuate.report.TextItem PageContent.getTextByBookmark(string bookmark)`

Returns the text element specified by the given bookmark.

**Parameters** **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");
```

## **actuate.viewer.PageContent.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 4-50 lists the actuate.viewer.ParameterValue functions.

**Table 4-50** actuate.viewer.ParameterValue functions

Function	Description
<code>getName( )</code>	Returns the name value
<code>getValue( )</code>	Returns the value value
<code>getValueIsNull( )</code>	Returns the valueIsNull value
<code>setName( )</code>	Sets the name value
<code>setValue( )</code>	Sets the value value
<code>setValueIsNull( )</code>	Sets the valueIsNull value

### **actuate.viewer.ParameterValue.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( );
```

### **actuate.viewer.ParameterValue.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( );
```

## **actuate.viewer.ParameterValue.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( ));
```

## **actuate.viewer.ParameterValue.setColumnName**

**Syntax** `void ParameterValue.setColumnName(string columnName)`

Sets the `columnName` value.

**Parameters** **columnName**  
String. The column name.

**Example** To set the column name to "Motorcycles", use code similar to the following:

```
myParamValue.setColumnName("Motorcycles");
```

## **actuate.viewer.ParameterValue.setValue**

**Syntax** `void ParameterValue.setValue(object value)`

Sets the value. A value can be a string or an array of strings.

**Parameters** **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);
```

## **actuate.viewer.ParameterValue.setValuesNull**

**Syntax** `void ParameterValue.setValuesNull(boolean valuesNull)`

Sets the `valueIsNull` value.

**Parameters** **valuesNull**  
Boolean. The `valueIsNull` value.

**Example** To set a `ParameterValue`'s `setValuesNull` 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 4-51 lists `actuate.viewer.RenderOptions` functions.

**Table 4-51** `actuate.viewer.RenderOptions` functions

Function	Description
<code>getOptions()</code>	Returns whether mouse scrolling is enabled
<code>setOption()</code>	Returns whether mouse panning is enabled

### **actuate.viewer.RenderOptions.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());
```

### **actuate.viewer.RenderOptions.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 4-1.



**Figure 4-1** 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 4-52 lists `actuate.viewer.ScrollPanel` functions.

**Table 4-52** `actuate.viewer.ScrollPanel` functions

Function	Description
<code>getMouseScrollingEnabled( )</code>	Returns whether mouse scrolling is enabled
<code>getPanInOutEnabled( )</code>	Returns whether mouse panning is enabled
<code>getScrollControlEnabled( )</code>	Returns whether scrolling is enabled
<code>setMouseScrollingEnabled( )</code>	Enables mouse scrolling
<code>setPanInOutEnabled( )</code>	Enables panning
<code>setScrollControlEnabled( )</code>	Enables scrolling

### `actuate.viewer.ScrollPanel.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( ));
```

## **actuate.viewer.ScrollPanel.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( ));
```

## **actuate.viewer.ScrollPanel.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( ));
```

## **actuate.viewer.ScrollPanel.setMouseScrollingEnabled**

**Syntax** `void ScrollPanel.setMouseScrollingEnabled(boolean enabled)`  
Enables mouse scrolling for this scroll panel.

**Parameters** **enabled**  
Boolean.

**Example** To disable mouse scrolling for sPanel, use code similar to the following:  

```
sPanel.setMouseScrollingEnabled(false);
```

## **actuate.viewer.ScrollPanel.setPanInOutEnabled**

**Syntax** `void ScrollPanel.setPanInOutEnabled(boolean enabled)`  
Enables panning in and out for this scroll panel.

**Parameters** **enabled**  
Boolean.

**Example** To disable panning for the sPanel object, use code similar to the following:  

```
sPanel.setPanInOutEnabled(false);
```

## **actuate.viewer.ScrollPanel.setScrollControlEnabled**

**Syntax** `void ScrollPanel.setScrollControlEnabled(boolean enabled)`

Enables scrolling for this scroll panel.

**Parameters** **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 4-53 lists actuate.viewer.SelectedContent functions.

**Table 4-53** actuate.viewer.SelectedContent functions

Function	Description
getColumnIndex( )	Returns the currently selected table column index number
getSelectedElement( )	Returns a copy of the currently selected element

### **actuate.viewer.SelectedContent.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( );
```

### **actuate.viewer.SelectedContent.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 4-54 lists actuate.viewer.UIConfig functions.

**Table 4-54** 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

### **actuate.viewer.UIConfig.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( );
```

### **actuate.viewer.UIConfig.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( )){ ...}
```

## **actuate.viewer.UIConfig.setContentPanel**

**Syntax** void UIConfig.setContentPanel(objectcontentPanel)

Sets the content panel for the viewer.

**Parameters** **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);
}
```

## **actuate.viewer.UIConfig.setShowToc**

**Syntax** void UIConfig.setShowToc(boolean showToc)

Sets the showToc flag.

**Parameters** **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 4-55 lists `actuate.viewer.UIOptions` functions.

**Table 4-55** `actuate.viewer.UIOptions` functions

Function	Description
<code>enableAdvancedSort( )</code>	Enables the advanced sort feature
<code>enableAggregation( )</code>	Enables the aggregation feature
<code>enableCalculatedColumn( )</code>	Enables the calculated column feature
<code>enableChartProperty( )</code>	Enables the chart properties feature
<code>enableChartSubType( )</code>	Enables the chart subtype selection
<code>enableCollapseExpand( )</code>	Enables the collapse/expand feature
<code>enableColumnEdit( )</code>	Enables the column editing feature
<code>enableColumnResize( )</code>	Enables the column resizing feature
<code>enableContentMargin( )</code>	Enables the content margin feature
<code>enableDataAnalyzer( )</code>	Enables the Data Analyzer feature
<code>enableDataExtraction( )</code>	Enables the data extraction feature
<code>enableEditReport( )</code>	Enables the report editing feature
<code>enableExportReport( )</code>	Enables the export report feature
<code>enableFilter( )</code>	Enables the filter feature
<code>enableFacebookComments( )</code>	Enables the Facebook comments feature.
<code>enableFlashGadgetType( )</code>	Enables the Flash gadget type change feature
<code>enableFormat( )</code>	Enables the format editing feature
<code>enableGroupEdit( )</code>	Enables the group editing feature
<code>enableHideShowItems( )</code>	Enables the hide/show item feature
<code>enableHighlight( )</code>	Enables the highlight feature

*(continues)*

**Table 4-55**      `actuate.viewer.UIOptions` functions (continued)

Function	Description
<code>enableHoverHighlight()</code>	Enables the hover highlight feature
<code>enableLaunchViewer()</code>	Enables the launch viewer feature
<code>enableLinkToThisPage()</code>	Enables the "link to this page" feature
<code>enableMainMenu()</code>	Enables the main menu feature
<code>enableMoveColumn()</code>	Enables column moving
<code>enablePageBreak()</code>	Enables the page break editing feature
<code>enablePageNavigation()</code>	Enables the page navigation feature
<code>enableParameterPage()</code>	Enables the parameter page feature
<code>enablePrint()</code>	Enables the print feature
<code>enableReorderColumns()</code>	Enables the column reordering
<code>enableRowResize()</code>	Enables row resizing
<code>enableSaveDesign()</code>	Enables the report design save feature
<code>enableSaveDocument()</code>	Enables the report document save feature
<code>enableShowToolTip()</code>	Enables the show tooltip feature
<code>enableSort()</code>	Enables the sort feature
<code>enableSuppressDuplicate()</code>	Enables the duplication suppression feature
<code>enableSwitchView()</code>	Enables the switch view feature
<code>enableTextEdit()</code>	Enables the text editing feature
<code>enableTOC()</code>	Enables the table of contents feature
<code>enableToolBar()</code>	Enables the toolbar feature
<code>enableToolBarContextMenu()</code>	Enables the toolbar context menu feature
<code>enableToolBarHelp()</code>	Enables the toolbar help feature
<code>enableTopBottomNFilter()</code>	Enables the top N and bottom N filter feature
<code>enableUndoRedo()</code>	Enables the undo and redo feature
<code>getFeatureMap()</code>	Returns a list of enabled and disabled features

## **`actuate.viewer.UIOptions.enableAdvancedSort`**

**Syntax**    `void UIOptions.enableAdvancedSort(boolean enabled)`

Enables or disables the advanced sort feature.

**Parameters**    **enabled**  
 Boolean. True enables this option.

**Example** To disable the advanced sort feature, use code similar to the following:

```
viewerOpts.enableAdvancedSort (false) ;
```

## **actuate.viewer.UIOptions.enableAggregation**

**Syntax** void UIOptions.enableAggregation(boolean enabled)

Enables or disables the aggregation feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the aggregation feature, use code similar to the following:

```
viewerOpts.enableAggregation (false) ;
```

## **actuate.viewer.UIOptions.enableCalculatedColumn**

**Syntax** void UIOptions.enableCalculatedColumn(boolean enabled)

Enables or disables the calculated column feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the calculated column feature, use code similar to the following:

```
viewerOpts.enableCalculatedColumn (false) ;
```

## **actuate.viewer.UIOptions.enableChartProperty**

**Syntax** void UIOptions.enableChartProperty(boolean enabled)

Enables or disables the chart properties feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the chart properties feature, use code similar to the following:

```
viewerOpts.enableChartProperty (false) ;
```

## **actuate.viewer.UIOptions.enableChartSubType**

**Syntax** void UIOptions.enableChartSubType(boolean enabled)

Enables or disables the chart subtype selection feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the chart subtype selection feature, use code similar to the following:

```
viewerOpts.enableChartSubType(false);
```

## **actuate.viewer.UIOptions.enableCollapseExpand**

**Syntax** void UIOptions.enableCollapseExpand(boolean enabled)

Enables or disables the collapse/expand feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the collapse/expand feature, use code similar to the following:

```
viewerOpts.enableCollapseExpand(false);
```

## **actuate.viewer.UIOptions.enableColumnEdit**

**Syntax** void UIOptions.enableColumnEdit(boolean enabled)

Enables or disables the column editing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the column editing feature, use code similar to the following:

```
viewerOpts.enableColumnEdit(false);
```

## **actuate.viewer.UIOptions.enableColumnResize**

**Syntax** void UIOptions.enableColumnResize(boolean enabled)

Enables or disables the column resizing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the column resizing feature, use code similar to the following:

```
viewerOpts.enableColumnResize(false);
```

## **actuate.viewer.UIOptions.enableContentMargin**

**Syntax** void UIOptions.enableContentMargin(boolean enabled)

Enables or disables the content margin feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the content margin feature, use code similar to the following:

```
viewerOpts.enableContentMargin(false);
```

## **actuate.viewer.UIOptions.enableDataAnalyzer**

**Syntax** void UIOptions.enableDataAnalyzer(boolean enabled)

Enables or disables the Data Analyzer feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the Data Analyzer feature, use code similar to the following:

```
viewerOpts.enableDataAnalyzer(false);
```

## **actuate.viewer.UIOptions.enableDataExtraction**

**Syntax** void UIOptions.enableDataExtraction(boolean enabled)

Enables or disables the data extraction feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the data extraction feature, use code similar to the following:

```
viewerOpts.enableDataExtraction(false);
```

## **actuate.viewer.UIOptions.enableEditReport**

**Syntax** void UIOptions.enableEditReport(boolean enabled)

Enables or disables the report editing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the report editing feature, use code similar to the following:

```
viewerOpts.enableEditReport(false);
```

## **actuate.viewer.UIOptions.enableExportReport**

**Syntax** void UIOptions.enableExportReport(boolean enabled)

Enables or disables the export report feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the export report feature, use code similar to the following:

```
viewerOpts.enableExportReport(false);
```

## **actuate.viewer.UIOptions.enableFilter**

**Syntax** void UIOptions.enableFilter(boolean enabled)

Enables or disables the filter feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the filter feature, use code similar to the following:

```
viewerOpts.enableFilter(false);
```

## **actuate.viewer.UIOptions.enableFacebookComments**

**Syntax** void UIOptions.enableFacebookComments(boolean enabled)

Enables or disables the Facebook comments feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the Facebook comments feature, use code similar to the following:

```
viewerOpts.enableFacebookComments(false);
```

## **actuate.viewer.UIOptions.enableFlashGadgetType**

**Syntax** void UIOptions.enableFlashGadgetType(boolean enabled)

Enables or disables the Flash gadget type change control.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the Flash gadget type change control, use code similar to the following:

```
viewerOpts.enableFlashGadgetType(false);
```

## **actuate.viewer.UIOptions.enableFormat**

**Syntax** void UIOptions.enableFormat(boolean enabled)

Enables or disables the format editing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the format editing feature, use code similar to the following:

```
viewerOpts.enableFormat(false);
```

## **actuate.viewer.UIOptions.enableGroupEdit**

**Syntax** void UIOptions.enableGroupEdit(boolean enabled)

Enables or disables the group editing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the group editing feature, use code similar to the following:

```
viewerOpts.enableGroupEdit(false);
```

## **actuate.viewer.UIOptions.enableHideShowItems**

**Syntax** void UIOptions.enableHideShowItems(boolean enabled)

Enables or disables the hide/show item feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the hide/show feature, use code similar to the following:

```
viewerOpts.enableHideShowItems(false);
```

## **actuate.viewer.UIOptions.enableHighlight**

**Syntax** void UIOptions.enableHighlight(boolean enabled)

Enables or disables the highlight feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the highlight feature, use code similar to the following:

```
viewerOpts.enableHighlight(false);
```

## **actuate.viewer.UIOptions.enableHoverHighlight**

**Syntax** void UIOptions.enableHoverHighlight(boolean enabled)

Enables or disables the hover highlight feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the hover highlight feature, use code similar to the following:

```
viewerOpts.enableHoverHighlight(false);
```

## **actuate.viewer.UIOptions.enableLaunchViewer**

**Syntax** void UIOptions.enableLaunchViewer(boolean enabled)

Enables or disables the launch viewer feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the launch viewer feature, use code similar to the following:

```
viewerOpts.enableLaunchViewer(false);
```

## **actuate.viewer.UIOptions.enableLinkToThisPage**

**Syntax** void UIOptions.enableLinkToThisPage(boolean enabled)

Enables or disables the "link to this page" feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the "link to this page" feature, use code similar to the following:

```
viewerOpts.enableLinkToThisPage(false);
```

## **actuate.viewer.UIOptions.enableMainMenu**

**Syntax** void UIOptions.enableMainMenu(boolean enabled)

Enables or disables the main menu feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the main menu feature, use code similar to the following:

```
viewerOpts.enableMainMenu(false);
```

## **actuate.viewer.UIOptions.enableMoveColumn**

**Syntax** void UIOptions.enableMoveColumn(boolean enabled)

Enables or disables the option to move columns.

**Parameters** **enabled**  
Boolean. True enables this option.



**Example** To disable the option to move columns, use code similar to the following:

```
viewerOpts.enableMoveColumn(false);
```

## **actuate.viewer.UIOptions.enablePageBreak**

**Syntax** void UIOptions.enablePageBreak(boolean enabled)

Enables or disables the page break editing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the page break editing feature, use code similar to the following:

```
viewerOpts.enablePageBreak(false);
```

## **actuate.viewer.UIOptions.enablePageNavigation**

**Syntax** void UIOptions.enablePageNavigation(boolean enabled)

Enables or disables the page navigation feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the page navigation feature, use code similar to the following:

```
viewerOpts.enablePageNavigation(false);
```

## **actuate.viewer.UIOptions.enableParameterPage**

**Syntax** void UIOptions.enableParameterPage(boolean enabled)

Enables or disables the parameter page feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the parameter page feature, use code similar to the following:

```
viewerOpts.enableParameterPage(false);
```

## **actuate.viewer.UIOptions.enablePrint**

**Syntax** void UIOptions.enablePrint(boolean enabled)

Enables or disables the print feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the print feature, use code similar to the following:

```
viewerOpts.enablePrint(false);
```

## **actuate.viewer.UIOptions.enableReorderColumns**

**Syntax** void UIOptions.enableReorderColumns(boolean enabled)

Enables or disables the column reordering feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the column reordering feature, use code similar to the following:

```
viewerOpts.enableReorderColumns(false);
```

## **actuate.viewer.UIOptions.enableRowResize**

**Syntax** void UIOptions.enableRowResize(boolean enabled)

Enables or disables row resizing.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable row resizing, use code similar to the following:

```
viewerOpts.enableRowResize(false);
```

## **actuate.viewer.UIOptions.enableSaveDesign**

**Syntax** void UIOptions.enableSaveDesign(boolean enabled)

Enables or disables the report design save feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the report design save feature, use code similar to the following:

```
viewerOpts.enableSaveDesign(false);
```

## **actuate.viewer.UIOptions.enableSaveDocument**

**Syntax** void UIOptions.enableSaveDocument(boolean enabled)

Enables or disables the report document save feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the report document save feature, use code similar to the following:

```
viewerOpts.enableSaveDocument (false) ;
```

## **actuate.viewer.UIOptions.enableShowToolTip**

**Syntax** void UIOptions.enableShowToolTip(boolean enabled)

Enables or disables the showing of tooltips.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the showing of tooltips, use code similar to the following:

```
viewerOpts.enableShowToolTip (false) ;
```

## **actuate.viewer.UIOptions.enableSort**

**Syntax** void UIOptions.enableSort(boolean enabled)

Enables or disables the sort feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the sort feature, use code similar to the following:

```
viewerOpts.enableSort (false) ;
```

## **actuate.viewer.UIOptions.enableSuppressDuplicate**

**Syntax** void UIOptions.enableSuppressDuplicate(boolean enabled)

Enables or disables the duplication suppression feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the duplication suppression feature, use code similar to the following:

```
viewerOpts.enableSuppressDuplicate (false) ;
```

## **actuate.viewer.UIOptions.enableSwitchView**

**Syntax** void UIOptions.enableSwitchView(boolean enabled)

Enables or disables the switch view feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the switch view feature, use code similar to the following:

```
viewerOpts.enableSwitchView(false);
```

## **actuate.viewer.UIOptions.enableTextEdit**

**Syntax** void UIOptions.enableTextEdit(boolean enabled)

Enables or disables the text editing feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the text editing feature, use code similar to the following:

```
viewerOpts.enableTextEdit(false);
```

## **actuate.viewer.UIOptions.enableTOC**

**Syntax** void UIOptions.enableTOC(boolean enabled)

Enables or disables the table of contents feature.

**Parameters** **enabled**  
Boolean. True enables this option.

**Example** To disable the table of contents feature, use code similar to the following:

```
viewerOpts.enableTOC(false);
```

## **actuate.viewer.UIOptions.enableToolBar**

**Syntax** void UIOptions.enableToolBar(boolean enabled)

Enables or disables the toolbar feature.

**Parameters** **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;
}
```

## **actuate.viewer.UIOptions.enableToolBarContextMenu**

**Syntax** `void UIOptions.enableToolBarContextMenu(boolean enabled)`

Enables or disables the context menu feature.

**Parameters** **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;
}
```

## **actuate.viewer.UIOptions.enableToolBarHelp**

**Syntax** `void UIOptions.enableToolBarHelp(boolean enabled)`

Enables or disables the toolbar help feature.

**Parameters**    **enabled**  
Boolean. True enables this option.

**Example**        To disable the toolbar help feature, use code similar to the following:

```
viewerOpts.enableToolbarHelp(false);
```

## **actuate.viewer.UIOptions.enableTopBottomNFilter**

**Syntax**        void UIOptions.enableTopBottomNFilter(boolean enabled)

Enables or disables the top N and bottom N filter feature.

**Parameters**    **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);
```

## **actuate.viewer.UIOptions.enableUndoRedo**

**Syntax**        void UIOptions.enableUndoRedo(boolean enabled)

Enables or disables the undo and redo feature.

**Parameters**    **enabled**  
Boolean. True enables this option.

**Example**        To disable the undo and redo feature, use code similar to the following:

```
viewerOpts.enableUndoRedo(false);
```

## **actuate.viewer.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.

---

## 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 4-56 lists actuate.viewer.ViewerException functions.

**Table 4-56** actuate.viewer.ViewerException functions

Function	Description
getElement( )	Returns the element for which the exception occurred
getErrorMessage( )	Returns the exception message

### **actuate.viewer.ViewerException.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( ));
```

## **actuate.viewer.ViewerException.getMessage**

**Syntax** string ViewerException.getMessage( )

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.getMessage( ));
```



# **BIRT Data Analyzer API classes**

This chapter contains the following topics:

- About the BIRT Data Analyzer JavaScript API
- Data Analyzer API reference
- Data Analyzer JavaScript classes quick reference

---

## About the BIRT Data Analyzer JavaScript API

The Data Analyzer 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 BIRT iHub. The Actuate JavaScript API functions that are described in this chapter invoke and control the Data Analyzer viewer and elements that are associated with the viewer. The Data Analyzer 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 Data Analyzer 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 `<div>` element on a web page.

Use the following JavaScript code to create an instance of a Data Analyzer viewer:

```
var ctViewer = new actuate.XTabAnalyzer("cTab");
```

In this example, `cTab` is the name value for the `<div>` element that holds the cross tab content. The web page body must contain a `<div>` element with an ID value of `cTab`, as shown in the following code:

```
<DIV ID="cTab"></DIV>
```

When no `<div>` element with the correct ID value exists in the web page body, the Data Analyzer 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 Data Analyzer viewer directly. This sample report document installs with iHub.

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 `<div>` element.

---

## Data Analyzer API reference

This section provides an alphabetic listing of the Data Analyzer 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 Data Analyzer viewer in the `acviewer` container. The `acviewer` container is a `<div>` 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 <div> 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( );">
```

---

## Data Analyzer JavaScript classes quick reference

Table 5-1 lists the Data Analyzer JavaScript classes.

**Table 5-1** Actuate Data Analyzer JavaScript classes

JavaScript class	Description
actuate.XTabAnalyzer	A Data Analyzer 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 Data Analyzer 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 Data Analyzer 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 Data Analyzer

---

---

## Class **actuate.XTabAnalyzer**

**Description** The XTabAnalyzer class represents a Data Analyzer viewer, used to view and operate a crosstab.

### Constructor

**Syntax** `actuate.XTabAnalyzer( )`

Constructs a new Data Analyzer object.

`actuate.XTabAnalyzer(object xtabContainer, actuate.xtabanalyzer.UIOptions uiOptions)`

`actuate.XTabAnalyzer(string xtabContainerId, actuate.xtabanalyzer.UIOptions uiOptions)`

Constructs a new Data Analyzer object in the specified container.

**Parameters** **xtabContainer**

Object. A document object referencing the HTML <div> element that contains the xTabAnalyzer viewer.

**xtabContainerId**

String. The value of the ID parameter for an HTML <div> 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 <div> element.

**uiOptions**

actuate.xtabanalyzer.UIOptions object. Optional. UIOptions references display options for the viewer.

### Function summary

Table 5-2 lists actuate.XTabAnalyzer functions.

**Table 5-2** actuate.XTabAnalyzer functions

Function	Description
<code>commit( )</code>	Commits all changes to the report design
<code>forceSoftRestart( )</code>	Forces the viewer to restart
<code>getCurrentPageContent( )</code>	Returns the Current Page Content object
<code>getCurrentPageNum( )</code>	Returns the current page number
<code>getGadgetId</code>	Returns the gadget ID of the shown cross tab
<code>getHeight( )</code>	Returns the viewer height

**Table 5-2**      **actuate.XTabAnalyzer functions (continued)**

<b>Function</b>	<b>Description</b>
<code>getLeft( )</code>	Returns the viewer left margin
<code>getParameterValues( )</code>	Returns the parameter values.
<code>getPosition( )</code>	Returns the CSS position attribute value
<code>getTop( )</code>	Returns the viewer top margin
<code>getTotalPageCount( )</code>	Returns the total page count
<code>getUIOptions( )</code>	Returns the <code>actuate.xtabanalyzer.UIOptions</code> object assigned to this viewer
<code>getViewer( )</code>	Gets a viewer within a container
<code>getWidth( )</code>	Returns the viewer width
<code>getXTabBookmark( )</code>	Returns the bookmark of the cross tab displayed in the viewer
<code>getXTabId( )</code>	Returns the instance ID of the cross tab displayed in the viewer
<code>isActive( )</code>	Checks if current viewer pop-up is active
<code>isDashboard( )</code>	Checks if the current viewer pop-up is a dashboard
<code>isInteractive( )</code>	Checks if the current viewer is interactive
<code>registerEventHandler( )</code>	Registers an event handler
<code>removeEventHandler( )</code>	Removes an event handler
<code>reset( )</code>	Resets the viewer object
<code>resizeTo( )</code>	Resizes the viewer
<code>rollback( )</code>	Rolls back all changes in the viewer and refreshes its content
<code>setGadgetId</code>	Sets the gadget id of the cross tab
<code>setHeight( )</code>	Sets the viewer height
<code>setIVMode( )</code>	Sets whether the viewer is in IV mode
<code>setLeft( )</code>	Sets the viewer left margin
<code>setOnClosed( )</code>	Sets callback when the pop-up window is closed
<code>setPageNum( )</code>	Sets the page number
<code>setPosition( )</code>	Sets the CSS position attribute
<code>setParameterValues( )</code>	Sets the parameter values

*(continues)*

**Table 5-2**      **actuate.XTabAnalyzer functions (continued)**

Function	Description
setReportletDocumentMode( )	Sets a Reportlet to document mode
setService( )	Sets the Information Console 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
setXTabId( )	Sets the instance ID of the cross tab
submit( )	Submits asynchronous operations and renders the requested components

## **actuate.XTabAnalyzer.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( );  
}
```

## **actuate.XTabAnalyzer.forceSoftRestart**

**Syntax**    `void XTabAnalyzer.forceSoftRestart( )`

Forces the viewer to restart.



**Example** This example restarts the viewer:

```
this.onclick = function(event){  
    forceSoftRestart( );  
}
```

## **actuate.XTabAnalyzer.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( );  
}
```

## **actuate.XTabAnalyzer.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( );  
}
```

## **actuate.XTabAnalyzer.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( );  
}
```

## actuate.XTabAnalyzer.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( );
    }
}
```

## actuate.XTabAnalyzer.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( );
    }
}
```

## actuate.XTabAnalyzer.getParameterValues

**Syntax** actuate.xtabanalyzer.ParameterValue[ ] XTabAnalyzer.getParameterValues( )

Returns the parameter values.

**Returns** actuate.xtabanalyzer.ParameterValue[ ] or actuate.parameter.ParameterValue[ ]. An array of parameter values.

## actuate.XTabAnalyzer.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( );
    }
}
```

## **actuate.XTabAnalyzer.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( );
    }
}
```

## **actuate.XTabAnalyzer.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( ));
```

## **actuate.XTabAnalyzer.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. Data Analyzer 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);
}
```

## actuate.XTabAnalyzer.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.

**Parameters** **container**  
HTMLElement. The container instance ID from which to retrieve the viewer.

**Returns** XTabAnalyzer object. The Data Analyzer viewer.

**Example** This example retrieves the viewer:

```
function retrieveViewer( ){
    return viewer.getViewer( );
}
```

## actuate.XTabAnalyzer.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( );
    }
}
```

## actuate.XTabAnalyzer.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( );
}
```

## **actuate.XTabAnalyzer.getXTabId**

**Syntax** string XTabAnalyzer.getXTabId( )

Returns the current instance ID of the data analyzer. This function is useful in integration with Interactive Viewer and supports the ability of Interactive Viewer to obtain and use the data analyzer instance ID.

**Returns** String. A data analyzer instance ID.

**Example** This example retrieves the data analyzer instance ID:

```
function retrieveXTabId( myviewer ){
    return myviewer.getXTabId( );
}
```

## **actuate.XTabAnalyzer.isActive**

**Syntax** boolean XTabAnalyzer.isActive( )

Returns true when a pop-up containing a data analyzer is active and false in all other cases.

**Returns** Boolean. True indicates an active data analyzer 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);
    }
}
```

## actuate.XTabAnalyzer.isDashboard

**Syntax** boolean XTabAnalyzer.isDashboard( )

Returns true when dashboard mode is active and false in all other cases.

**Returns** Boolean. True indicates dashboard mode.

## actuate.XTabAnalyzer.isInteractive

**Syntax** boolean XTabAnalyzer.isInteractive( )

Returns whether this Data Analyzer is in Interactive mode.

**Returns** Boolean. True indicates dashboard mode.

**Example** This example displays whether myDataAnalyzer is interactive:

```
alert("Interactive mode: " + myDataAnalyzer.isInteractive( ));
```

## actuate.XTabAnalyzer.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 );
}
```

## actuate.XTabAnalyzer.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 );
}
```

## **actuate.XTabAnalyzer.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( );
}
```

## **actuate.XTabAnalyzer.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);
    }
}
```

## actuate.XTabAnalyzer.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( );  
}
```

## actuate.XTabAnalyzer.setGadgetId

**Syntax** void XTabAnalyzer.setGadgetId(string gadgetId)

Sets the cross tab gadget ID. This function is used for dashboard integration.

**Parameters** **gadgetId**  
String. The gadget ID used to render the cross tab.

**Example** This example sets the gadget ID:

```
function setGadgetID(id){  
    viewer.setGadgetId(id);  
}
```

## actuate.XTabAnalyzer.setHeight

**Syntax** void XTabAnalyzer.setHeight(integer height)

Changes the height of the viewer.

**Parameters** **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( );  
    }  
}
```



## actuate.XTabAnalyzer.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.

**Parameters** **ivMode**  
Boolean. Set to true if using IV mode.

**Example** This example sets IVMode for the viewer:

```
function setViewerMode(mode) {  
    viewer.setIVMode(mode);  
}
```

## actuate.XTabAnalyzer.setLeft

**Syntax** void XTabAnalyzer.setLeft(integer left)

Sets the position of the viewer's left margin.

**Parameters** **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( );  
    }  
}
```

## actuate.XTabAnalyzer.setOnClosed

**Syntax** void XTabAnalyzer.setOnClosed(function callback)

Sets a callback function to call when a viewer pop-up closes.

**Parameters** **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);
    }
}
```

## **actuate.XTabAnalyzer.setPageNum**

**Syntax** void XTabAnalyzer.setPageNum(function pageNum)

Sets the page number.

**Parameters** **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);
    }
}
```

## **actuate.XTabAnalyzer.setPosition**

**Syntax** void XTabAnalyzer.setPosition(string position)

Sets the CSS position attribute.

**Parameters** **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( );
    }
}
```

## **actuate.XTabAnalyzer.setParameterValues**

**Syntax** void XTabAnalyzer.setParameterValues(actuate.xtabanalyzer.ParameterValue[ ] parameterValues)

Sets the CSS position attribute.

**Parameters** **parameterValues**  
actuate.xtabanalyzer.ParameterValue[ ] or actuate.parameter.ParameterValue[ ].  
An array of parameter values.

## **actuate.XTabAnalyzer.setReportletDocumentMode**

**Syntax** void XTabAnalyzer.setReportletDocumentMode(boolean reportletMode)  
Sets whether the viewer displays documents as Reportlets.

**Parameters** **reportletMode**  
Boolean. True indicates Reportlet display mode.

## **actuate.XTabAnalyzer.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.

**Parameters** **reportName**  
String. The name of the report file.

**Example** This example sets the report name to reportfile.rptdocument and reloads the Data Analyzer viewer with its content:

```
function run( ){  
    container = document.getElementById("acviewer");  
    viewer = new actuate.XTabAnalyzer(container);  
    viewer.setReportName("reportfile.rptdocument");  
    viewer.submit( );  
}
```

## **actuate.XTabAnalyzer.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);  
}
```

## actuate.XTabAnalyzer.setSupportSVG

**Syntax** void XTabAnalyzer.setSupportSVG(boolean svgFlag)

Sets a flag indicating whether or not the browser supports SVG.

**Parameters** **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);  
}
```

## actuate.XTabAnalyzer.setTop

**Syntax** void XTabAnalyzer.setTop(integer top)

Sets the top margin for the viewer.

**Parameters** **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( );  
    }  
}
```

## actuate.XTabAnalyzer.setUIOptions

**Syntax** void XTabAnalyzer.setUIOptions(actuate.xtabanalyzer.uioptions options)

Sets the user interface options enabled for the viewer.

**Parameters** **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);  
}
```

## **actuate.XTabAnalyzer.setWidth**

**Syntax** void XTabAnalyzer.setWidth(integer width)

Sets the width for the viewer.

**Parameters** **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( );  
    }  
}
```

## **actuate.XTabAnalyzer.setXTabBookmark**

**Syntax** void XTabAnalyzer.setXTabBookmark(string bookmark)

Sets the bookmark for a cross tab to render in the viewer.

**Parameters** **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( );  
}
```

## **actuate.XTabAnalyzer.setXTabId**

**Syntax** void XTabAnalyzer.setXTabId(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.

**Parameters**    **iid**  
String. The instance ID.

**Example**    This example sets the cross tab instance ID:

```
function setxtabInstance(id) {  
    viewer.setXTabId(id);  
}
```

## **actuate.XTabAnalyzer.submit**

**Syntax**    void XTabAnalyzer.submit(function callback, boolean rerun)

Submits requests to the server for the Data Analyzer 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 Data Analyzer container. The submit() function throws an exception when another submit() operation is pending. A CONTENT\_CHANGED event fires when the Data Analyzer 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 5-3 lists `actuate.xtabanalyzer.Crosstab` functions.

**Table 5-3** `actuate.xtabanalyzer.Crosstab` functions

Function	Description
<code>addDimension( )</code>	Adds a dimension to the cross tab
<code>addMeasure( )</code>	Adds a measure to the cross tab
<code>applyOptions( )</code>	Sets options for the cross tab
<code>changeMeasureDirection( )</code>	Switches measure direction
<code>clearFilters( )</code>	Clears cross tab filters
<code>drill( )</code>	Drills up or down measure levels, replacing drill and filter conditions
<code>drillDown( )</code>	Drills down a measure level, updating drill conditions
<code>drillUp( )</code>	Drills up a measure level, updating drill conditions
<code>editMeasure( )</code>	Edits a measure
<code>getBookmark( )</code>	Retrieves the cross tab element bookmark
<code>getColumn( )</code>	Retrieves table data by column index
<code>getData( )</code>	Returns the data from a cross tab
<code>getHtmlDom( )</code>	Retrieves the HTML DOM object
<code>getPageContent( )</code>	Retrieves the content of the page the cross tab belongs to
<code>getRow( )</code>	Retrieves table data by row index
<code>getType( )</code>	Retrieves the report element type
<code>hideDetail( )</code>	Hides the detail of a specified level

*(continues)*

**Table 5-3**      `actuate.xtabanalyzer.Crosstab` functions (continued)

Function	Description
<code>pivot()</code>	Pivots the cross tab
<code>removeDimension()</code>	Removes a dimension from the cross tab
<code>removeMeasure()</code>	Removes a measure from the cross tab
<code>reorderDimension()</code>	Reorders a dimension
<code>reorderMeasure()</code>	Reorders a measure
<code>setFilters()</code>	Sets the cross tab's filters
<code>setSorters()</code>	Sets the cross tab's sorters
<code>setTotals()</code>	Sets the cross tab's totals
<code>showDetail()</code>	Shows details to the lower level
<code>submit()</code>	Applies changes made to the cross tab

## **`actuate.xtabanalyzer.Crosstab.addDimension`**

**Syntax**    `void Crosstab.addDimension(actuate.xtabanalyzer.Dimension dimension)`

Adds a dimension to the cross tab object.

**Parameters**    **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( );
}
```



## actuate.xtabanalyzer.Crosstab.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*.

## actuate.xtabanalyzer.Crosstab.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.

**actuate.xtabanalyzer.Crosstab.  
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( );
    }
}
```

**actuate.xtabanalyzer.Crosstab.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( );
    }
}
```

**actuate.xtabanalyzer.Crosstab.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.

**Parameters** **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( );  
}
```

## **actuate.xtabanalyzer.Crosstab.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.

**Parameters** **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( );  
}
```

## **actuate.xtabanalyzer.Crosstab.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.

**Parameters** **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( );
}
```

## **actuate.xtabanalyzer.Crosstab.editMeasure**

**Syntax** void Crosstab.editMeasure(actuate.xtabanalyzer.Measure 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( );
    }
}
```

## **actuate.xtabanalyzer.Crosstab.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;  
}
```

## **actuate.xtabanalyzer.Crosstab.getColumn**

**Syntax** string[ ] Crosstab.getColumn(integer columnIndex)

Returns the table data by column index.

**Parameters** **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];  
}
```

## **actuate.xtabanalyzer.Crosstab.getData**

**Syntax** String[ ] Crosstab.getData(boolean forceReparse)

Returns the data in a cross tab.

**Parameters** **forceReparse**

Boolean. Forces a cache refresh when true.

**Returns** String[ ]. The data from the cross tab as an array of strings.

## **actuate.xtabanalyzer.Crosstab.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;  
}
```

## **actuate.xtabanalyzer.Crosstab.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( );  
}
```

## **actuate.xtabanalyzer.Crosstab.getRow**

**Syntax** string[ ] Crosstab.getRow(integer rowIndex)

Returns table data based on row index.

**Parameters** **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];  
}
```

## **actuate.xtabanalyzer.Crosstab.getType**

**Syntax** string Crosstab.getType( )  
Returns the report element type.

**Returns** String containing the value "Crosstab".

## **actuate.xtabanalyzer.Crosstab.hideDetail**

**Syntax** void Crosstab.hideDetail(string levelName)  
Hides details of the specified level.

**Parameters** **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( );  
    }  
}
```

## **actuate.xtabanalyzer.Crosstab.pivot**

**Syntax** void Crosstab.pivot( )  
Pivots the cross tab.

**Example** This example pivots a cross tab:

```
function pivot(crosstab){  
    crosstab.pivot( );  
    crosstab.submit( );  
}
```

## **actuate.xtabanalyzer.Crosstab.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( );
    }
}
```

**actuate.xtabanalyzer.Crosstab.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( );
}
```

## **actuate.xtabanalyzer.Crosstab.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.

**Parameters** **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( );
}
```

## **actuate.xtabanalyzer.Crosstab.reorderMeasure**

**Syntax** void Crosstab.reorderMeasure(actuate.xtabanalyzer.Measure 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( );
}
```

## **actuate.xtabanalyzer.Crosstab.setFilters**

**Syntax** void Crosstab.setFilters(actuate.xtabanalyzer.Filter[ ] filters)

Sets an array of filters for the cross tab.

**Parameters** **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( );
}
```

## **actuate.xtabanalyzer.Crosstab.setSorters**

**Syntax** void Crosstab.setSorters(actuate.xtabanalyzer.Sorter[ ] sorters)

Sets an array of sorters for the cross tab.

**Parameters** **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( );
}
```

## actuate.xtabanalyzer.Crosstab.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( );
}
```

## actuate.xtabanalyzer.Crosstab.showDetail

**Syntax** void Crosstab.showDetail(string axisType)

Shows a level of detail within a cross tab.

**Parameters** **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( );
}
```

## **actuate.xtabanalyzer.Crosstab.submit**

**Syntax** void Crosstab.submit(function callback)

Applies the changes made to this element. This is an asynchronous operation.

**Parameters** **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 5-4 lists `actuate.xtabanalyzer.Dimension` functions.

**Table 5-4** `actuate.xtabanalyzer.Dimension` functions

Function	Description
<code>addLevel( )</code>	Adds the level to the dimension
<code>getAxisType( )</code>	Returns the axis type
<code>getDimensionName( )</code>	Returns the dimension name
<code>getIndex( )</code>	Returns the index of the dimension
<code>getLevels( )</code>	Returns cross tab levels
<code>getNewAxisType( )</code>	Returns the new axis type
<code>getNewIndex( )</code>	Returns the new index
<code>setAxisType( )</code>	Sets the axis type
<code>setDimensionName( )</code>	Sets the dimension name
<code>setIndex( )</code>	Sets the index
<code>setLevels( )</code>	Sets the levels
<code>setNewAxisType( )</code>	Sets the new axis type
<code>setNewIndex( )</code>	Sets the new index axis type

## `actuate.xtabanalyzer.Dimension.addLevel`

**Syntax** `void Dimension.addLevel(actuate.xtabanalyzer.Level level)`  
Adds a level to the dimension.

**Parameters** **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);
}
```

## **actuate.xtabanalyzer.Dimension.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);
    }
}
```

## **actuate.xtabanalyzer.Dimension.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;
}
```

## **actuate.xtabanalyzer.Dimension.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);  
}
```

## **actuate.xtabanalyzer.Dimension.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;  
}
```

## **actuate.xtabanalyzer.Dimension.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;  
}
```

## actuate.xtabanalyzer.Dimension.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;  
}
```

## actuate.xtabanalyzer.Dimension.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.

**Parameters** **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);  
}
```

## actuate.xtabanalyzer.Dimension.setDimensionName

**Syntax** void Dimension.setDimensionName(string dimensionName)

Sets the name for a dimension during its creation.

**Parameters** **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);  
}
```



## **actuate.xtabanalyzer.Dimension.setIndex**

**Syntax** void Dimension.setIndex(integer index)

Sets the index for the dimension.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.Dimension.setLevels**

**Syntax** void Dimension.setLevels(xtabanalyzer.Level[ ] levels)

Sets levels for the dimension.

**Parameters** **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);  
    }  
}
```

## **actuate.xtabanalyzer.Dimension.setNewAxisType**

**Syntax** void Dimension.setNewAxisType(integer newAxisType)

Sets the new axis type.

**Parameters** **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);  
        }  
    }  
}
```

## **actuate.xtabanalyzer.Dimension.setNewIndex**

**Syntax** void Dimension.setNewIndex(integer newIndex)

Sets the new index.

**Parameters** **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 5-5 lists `actuate.xtabanalyzer.Driller` functions.

**Table 5-5** `actuate.xtabanalyzer.Driller` functions

Function	Description
<code>addMember()</code>	Adds a member to the drill condition
<code>getDimension()</code>	Retrieves the driller dimension
<code>getMembers()</code>	Retrieves the members used by the drill
<code>setDimension()</code>	Sets the driller dimension
<code>setMembers()</code>	Adds an array of members to the drill condition

## `actuate.xtabanalyzer.Driller.addMember`

**Syntax** `void Driller.addMember(actuate.xtabanalyzer.MemberValue member)`  
Adds a dimension member to the drill condition. Functional candidates are Dimension members with levels.

**Parameters** **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( );
}
```

## actuate.xtabalyzer.Driller.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;
}
```

## actuate.xtabalyzer.Driller.getMembers

**Syntax** actuate.xtabalyzer.MemberValue[ ] Driller.getMembers( )

returns the list of members assigned to the driller.

**Returns** Array of actuate.xtabalyzer.MemberValue. A dimension member.

**Example** This example retrieves the members that a driller uses:

```
function getDrillerMembers(driller){
    if (driller){
        return driller.getMembers( );
    }
    return null;
}
```

## actuate.xtabalyzer.Driller.setDimension

**Syntax** void Driller.setDimension(string dimension)

Sets the dimension for the driller by name.

**Parameters** **dimension**  
String. A dimension name.

**Example** This example sets the dimension name for the driller:

```
function setRowAxis(driller){
    if (driller){
        dimension.setDimension("Row");
    }
}
```

## **actuate.xtabalyzer.Driller.setMembers**

**Syntax** void Driller.setMembers(actuate.xtabalyzer.MemberValue[ ] member)

Sets an array of members to the drill condition.

**Parameters** **member**

Array of actuate.xtabalyzer.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 5-6 lists the cross tab analyzer event constants.

**Table 5-6** `actuate.xtabanalyzer.Dimension` constants

Constant	Description
<code>ON_CONTENT_CHANGED</code>	Content changed event. Triggers when the displayed content has changed, for example when changing cross tab report content. The event handler takes an <code>actuate.XTabAnalyzer</code> object that represents the viewer for which the event occurred, as the only parameter.
<code>ON_CONTENT_SELECTED</code>	Content selected event. Triggers when a user clicks on report elements. The event handler takes the following parameters: <ul style="list-style-type: none"><li>■ <code>actuate.XTabAnalyzer</code>: object viewer for which event occurred</li><li>■ <code>actuate.xtabanalyzer.SelectedContent</code>: the <code>SelectedContent</code> object</li></ul>
<code>ON_EXCEPTION</code>	Exception event. Triggers when an exception occurs during an asynchronous operation. The event handler takes the following arguments: <ul style="list-style-type: none"><li>■ <code>actuate.XTabAnalyzer</code>: viewer for which the event occurred</li><li>■ <code>actuate.Exception</code>: Exception object</li></ul>
<code>ON_SESSION_TIMEOUT</code>	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 <code>ON_SESSION_TIMEOUT</code> event triggers. When no handler is registered for this event, a default built-in login dialog will be displayed.  This event is only available if the <code>SessionReloginTimeout</code> flag is set to true in the server configuration from which the API was initialized.  The event handler takes one parameter: an <code>actuate.XTabAnalyzer</code> 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 5-7 lists `actuate.xtabanalyzer.Exception` functions.

**Table 5-7** `actuate.xtabanalyzer.Exception` functions

Function	Description
<code>getDescription()</code>	Returns details of the exception
<code>getElement()</code>	Returns the report element for which the exception occurred, if available
<code>getErrCode()</code>	Returns the error code for <code>ERR_SERVER</code>
<code>getMessage()</code>	Returns a short message about the error
<code>getType()</code>	Returns the type of error exception
<code>isExceptionType()</code>	Returns Boolean indicating whether exception is of certain type

## `actuate.xtabanalyzer.Exception.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( ));
}
```

## **actuate.xtabalyzer.Exception.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( ));
}
```

## **actuate.xtabalyzer.Exception.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( ));
}
```

## **actuate.xtabalyzer.Exception.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( ));
}
```

## **actuate.xtabanalyzer.Exception.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( ));
}
```

## **actuate.xtabanalyzer.Exception.isExceptionType**

**Syntax** `boolean Exception.isExceptionType(object exceptionType)`

Checks an exception's type for a match against a specified type.

**Parameters** **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 5-8 lists the valid filter operators and the number of arguments to pass to the constructor or `setValues()`.

**Table 5-8** 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 5-8** 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 5-9 lists `actuate.xtabanalyzer.Filter` functions.

**Table 5-9**      `actuate.xtabanalyzer.Filter` functions

Function	Description
<code>getFilterType()</code>	Returns the filter type
<code>getLevelAttributeName()</code>	Returns the dimension level attribute name
<code>getLevelName()</code>	Returns the name of the filtered level
<code>getOperator()</code>	Returns the filter operator
<code>getValues()</code>	Returns the set of values the filter is using
<code>setFilterType()</code>	Sets the filter type
<code>setLevelAttributeName()</code>	Sets the dimension level attribute name
<code>setLevelName()</code>	Sets the dimension level name
<code>setOperator()</code>	Sets the filter operator
<code>setValues()</code>	Sets the values for the filter

### `actuate.xtabanalyzer.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;  
    }  
}
```

### `actuate.xtabanalyzer.Filter.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;  
    }  
}
```

## **actuate.xtabanalyzer.Filter.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;  
    }  
}
```

## **actuate.xtabanalyzer.Filter.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;  
    }  
}
```

## actuate.xtabanalyzer.Filter.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;
    }
}
```

## actuate.xtabanalyzer.Filter.setFilterType

**Syntax** void Filter.setFilterType(string filterType)

Sets the filter type to filter.

**Parameters** **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( );
}
```

## actuate.xtabanalyzer.Filter.setLevelAttributeName

**Syntax** void Filter.setLevelAttributeName(string levelAttributeName)

Sets the dimension level attribute to filter on by name.

**Parameters** **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( );
}
```

## **actuate.xtabanalyzer.Filter.setLevelName**

**Syntax** void Filter.setLevelName(string level)

Sets the level to filter by name.

**Parameters** **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( );
}
```

## **actuate.xtabanalyzer.Filter.setOperator**

**Syntax** void Filter.setOperator(string operator)

Sets the filter operator.

**Parameters** **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( );
}
```

## actuate.xtabanalyzer.Filter.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 5-10 lists `actuate.xtabanalyzer.GrandTotal` functions.

**Table 5-10** `actuate.xtabanalyzer.GrandTotal` functions

Function	Description
<code>addTotal( )</code>	Adds a total
<code>getAxisType( )</code>	Returns the axis type
<code>getTotals( )</code>	Returns the totals array
<code>getType( )</code>	Returns the grand total type
<code>setAxisType( )</code>	Sets the axis type
<code>setTotals( )</code>	Sets the totals array

## `actuate.xtabanalyzer.GrandTotal.addTotal`

**Syntax** `void GrandTotal.addTotal(object total)`  
Adds a total to the cross tab.

**Parameters** **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);
    }
}

```

## actuate.xtabanalyzer.GrandTotal.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);
    }
}

```

## actuate.xtabanalyzer.GrandTotal.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( );
}

```

## **actuate.xtabanalyzer.GrandTotal.getType**

**Syntax** `string GrandTotal.getType()`  
Returns the type for the total.

**Returns** String. The total type.

## **actuate.xtabanalyzer.GrandTotal.setAxisType**

**Syntax** `void GrandTotal.setAxisType(integer axisType)`  
Sets the axis type for the total.

**Parameters** **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);
    }
}
```

## **actuate.xtabanalyzer.GrandTotal.setTotals**

**Syntax** `void GrandTotal.setTotals(actuate.xtabanalyzer.Total[ ] totals)`  
Sets totals as an array.

**Parameters** **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 5-11 lists `actuate.xtabanalyzer.Level` functions.

**Table 5-11** `actuate.xtabanalyzer.Level` functions

Function	Description
<code>addAttribute( )</code>	Adds the level attribute
<code>getAttributes( )</code>	Returns the level attributes
<code>getIndex( )</code>	Returns the index of the level
<code>getLevelName( )</code>	Returns the level name
<code>setIndex( )</code>	Sets the index level
<code>setLevelName( )</code>	Sets the level name

### **actuate.xtabanalyzer.Level.addAttribute**

**Syntax** `void Level.addAttribute(actuate.xtabanalyzer.LevelAttribute attr)`  
Adds the level attribute.

**Parameters** **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 );
```

### **actuate.xtabanalyzer.Level.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( );
```

## **actuate.xtabanalyzer.Level.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;  
}
```

## **actuate.xtabanalyzer.Level.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;  
}
```

## **actuate.xtabanalyzer.Level.setIndex**

**Syntax** `void Level.setIndex(integer index)`

Sets the level index.

**Parameters** **index**  
Integer. The level index.

**Example** This example sets the level index:

```
function assignIndex(level,index){
    if (level){
        return level.setIndex(index);
    }
}
```

## **actuate.xtabanalyzer.Level.setLevelName**

**Syntax** void Level.setLevelName(string levelName)

Sets the level name.

**Parameters** **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 5-12 lists `actuate.xtabanalyzer.LevelAttribute` functions.

**Table 5-12** `actuate.xtabanalyzer.Level` functions

Function	Description
<code>getName( )</code>	Returns the level attribute name
<code>setName( )</code>	Sets the level attribute name

## `actuate.xtabanalyzer.LevelAttribute.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( );
```

## `actuate.xtabanalyzer.LevelAttribute.setName`

**Syntax** `void LevelAttribute.setName(string attributeName)`  
Sets the level attribute name.

**Parameters** **`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 5-13 lists `actuate.xtabanalyzer.Measure` functions.

**Table 5-13** `actuate.xtabanalyzer.Measure` functions

Function	Description
<code>getAggregationFunction( )</code>	Returns the aggregation function name
<code>getDataType( )</code>	Returns the computed column data type
<code>getExpression( )</code>	Returns the computed measure expression
<code>getIndex( )</code>	Returns the measure index
<code>getMeasureName( )</code>	Returns the measure name
<code>getNewIndex( )</code>	Returns the new index
<code>setAggregationFunction( )</code>	Sets the aggregation function name
<code>setDataType( )</code>	Sets the computed column data type
<code>setExpression( )</code>	Sets the computed measure expression
<code>setIndex( )</code>	Sets the measure index
<code>setMeasureName( )</code>	Sets the measure name
<code>setNewIndex( )</code>	Sets the new index

### **actuate.xtabanalyzer.Measure** **.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");
  }
}
```

## **actuate.xtabanalyzer.Measure.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;
}
```

## **actuate.xtabanalyzer.Measure.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;
}
```

## **actuate.xtabanalyzer.Measure.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;  
}
```

## **actuate.xtabanalyzer.Measure.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;  
}
```

## **actuate.xtabanalyzer.Measure.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;  
}
```

## **actuate.xtabanalyzer.Measure .setAggregationFunction**

**Syntax** void Measure.setAggregationFunction(string aggregationFunction)

Sets the aggregation function name.

**Parameters**    **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");
    }
}
```

## **actuate.xtabanalyzer.Measure.setDataType**

**Syntax**    void Measure.setDataType(string dataType)  
Sets the computed column data type name.

**Parameters**    **dataType**  
String. The data type.

## **actuate.xtabanalyzer.Measure.setExpression**

**Syntax**    void Measure.setExpression(string expression)  
Sets the computed measure expression.

**Parameters**    **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( );
    }
}
```

## **actuate.xtabanalyzer.Measure.setIndex**

**Syntax** void Measure.setIndex(integer index)

Sets the index.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.Measure.setMeasureName**

**Syntax** void Measure.setMeasureName(string measureName)

Sets the measure name.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.Measure.setNewIndex**

**Syntax** void Measure.setNewIndex(integer newIndex)

Sets a new measure index.

**Parameters** **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 5-14 lists `actuate.xtabanalyzer.MemberValue` functions.

**Table 5-14** `actuate.xtabanalyzer.MemberValue` functions

Function	Description
<code>addMember( )</code>	Adds a member value object
<code>getLevelName( )</code>	Retrieves the level name
<code>getMembers( )</code>	Retrieves an array of members
<code>getValue( )</code>	Returns the level value
<code>setLevelName( )</code>	Sets the level name
<code>setValue( )</code>	Sets the member value

## `actuate.xtabanalyzer.MemberValue.addMember`

- Syntax** `void MemberValue.addMember(actuate.xtabanalyzer.MemberValue member)`
- Adds a member value.
- Parameters**
- 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;
}
```

## **actuate.xtabanalyzer.MemberValue.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;
}
```

## **actuate.xtabanalyzer.MemberValue.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;
}
```

## **actuate.xtabanalyzer.MemberValue.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;  
}
```

## **actuate.xtabanalyzer.MemberValue.setLevelName**

**Syntax** void MemberValue.setLevelName(string level)

Sets the level name.

**Parameters** **level**  
String. The name of the level.

**Example** This example sets the level name:

```
function setMemberLevelName(member, lvlName) {  
    if (member) {  
        member.setLevelName(lvlName);  
    }  
}
```

## **actuate.xtabanalyzer.MemberValue.setValue**

**Syntax** void MemberValue.setValue(string level)

Sets the level value.

**Parameters** **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 5-15 lists `actuate.xtabanalyzer.Options` functions.

**Table 5-15**      `actuate.xtabanalyzer.Options` functions

Function	Description
<code>getColumnMirrorStartingLevel()</code>	Returns the column mirror starting level full name
<code>getColumnPageBreakInterval()</code>	Returns the column page break interval
<code>getEmptyCellValue()</code>	Returns the empty cell value
<code>getEnablePageBreak()</code>	Returns the page break enabled or disabled status
<code>getMeasureDirection()</code>	Returns the measure direction
<code>getRowMirrorStartingLevel()</code>	Returns the row mirror starting level full name
<code>getRowPageBreakInterval()</code>	Returns the row page break interval
<code>setColumnMirrorStartingLevel()</code>	Sets the column mirror starting level full name
<code>setColumnPageBreakInterval()</code>	Sets the column page break interval
<code>setEmptyCellValue()</code>	Sets the empty cell value
<code>setEnablePageBreak()</code>	Sets the flag to enable page breaks
<code>setMeasureDirection()</code>	Sets the measure direction
<code>setRowMirrorStartingLevel()</code>	Sets the row mirror starting level full name
<code>setRowPageBreakInterval()</code>	Sets the row page break interval

## `actuate.xtabanalyzer.Options` `.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;
}
```

## **actuate.xtabanalyzer.Options.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;
}
```

## **actuate.xtabanalyzer.Options.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;
}
```

## **actuate.xtabanalyzer.Options.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;
}
```

## **actuate.xtabanalyzer.Options.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;
}
```

## **actuate.xtabanalyzer.Options.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;
}
```

## **actuate.xtabanalyzer.Options.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;  
}
```

## **actuate.xtabanalyzer.Options .setColumnMirrorStartingLevel**

**Syntax** void Options.setColumnMirrorStartingLevel(string levelName)

Sets the column mirror starting level name.

**Parameters** **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);  
    }  
}
```

## **actuate.xtabanalyzer.Options .setColumnPageBreakInterval**

**Syntax** void Options.setColumnPageBreakInterval(integer columnPageBreakInterval)

Sets the column page break interval.

**Parameters** **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);  
    }  
}
```

## **actuate.xtabanalyzer.Options.setEmptyCellValue**

**Syntax** void Options.setEmptyCellValue(string emptyCellValue)

Sets the empty cell value.

**Parameters** **emptyCellValue**  
String. The empty cell value.

**Example** This example sets the empty cell value:

```
function setEmptyCell(options, cellValue) {  
    if (options) {  
        options.setEmptyCellValue(cellValue);  
    }  
}
```

## **actuate.xtabanalyzer.Options.setEnablePageBreak**

**Syntax** void Options.setEnablePageBreak(boolean enablePageBreak)  
Enables or disables page breaks.

**Parameters** **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);  
    }  
}
```

## **actuate.xtabanalyzer.Options.setMeasureDirection**

**Syntax** void Options.setMeasureDirection(string measureDirection)  
Sets the measure direction.

**Parameters** **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);  
    }  
}
```

## **actuate.xtabanalyzer.Options .setRowMirrorStartingLevel**

**Syntax** void Options.setRowMirrorStartingLevel(string levelName)

Sets the row mirror starting level.

**Parameters** **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);  
    }  
}
```

## **actuate.xtabanalyzer.Options .setRowPageBreakInterval**

**Syntax** void Options.setRowPageBreakInterval(integer rowPageBreakInterval)

Sets the row page break interval.

**Parameters** **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 5-16 lists `actuate.xtabanalyzer.PageContent` functions.

**Table 5-16** `actuate.xtabanalyzer.PageContent` functions

Function	Description
<code>getCrosstabByBookmark()</code>	Returns a report cross tab object
<code>getViewerId()</code>	Returns the cross tab viewer ID

## `actuate.xtabanalyzer.PageContent` `.getCrosstabByBookmark`

**Syntax** `actuate.xtabanalyzer.crosstab PageContent.getCrosstabByBookmark(string bookmark)`

Returns a cross tab object associated with a bookmark.

**Parameters** **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;
}
```

## **actuate.xtabanalyzer.PageContent.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 valuesIsNull)`

The `ParameterValue` class is used to specify a cross tab `ParameterValue` object.

**Parameters** **name**  
String. The parameter name.

**value**  
String. The parameter value.

**valuesIsNull**  
Boolean. Whether the value is null.

## Function summary

Table 5-17 lists `actuate.xtabanalyzer.ParameterValue` functions.

**Table 5-17** `actuate.xtabanalyzer.ParameterValue` functions

Function	Description
<code>getName()</code>	Returns the parameter name
<code>getValue()</code>	Returns the parameter value
<code>getValueIsNull()</code>	Returns whether the parameter has a null value
<code>setName()</code>	Sets the parameter name
<code>setEnabled()</code>	Sets the parameter value
<code>setMeasureIndex()</code>	Sets whether the parameter has a null value

## `actuate.xtabanalyzer.ParameterValue.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;
}
```

## **actuate.xtabanalyzer.ParameterValue.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;
}
```

## **actuate.xtabanalyzer.ParameterValue.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);
    }
}
```

## **actuate.xtabanalyzer.ParameterValue.setName**

**Syntax** void ParameterValue.setName(string name)

Sets the parameter name.

**Parameters** **name**  
String. The parameter name.

**Example** This example sets the parameter name:

```
function setParameterName(parametervalue, name){  
    parametervalue.setName(name);  
}
```

## **actuate.xtabanalyzer.ParameterValue.setValue**

**Syntax** void ParameterValue.setValue(string[ ] value)

Sets the parameter value.

**Parameters** **value**  
String. The parameter value.

**Example** This example sets the parameter value:

```
function setParameterValue(parametervalue, value){  
    parametervalue.setValue(value);  
}
```

## **actuate.xtabanalyzer.ParameterValue.setValuesNull**

**Syntax** void ParameterValue.setValuesNull(boolean valuesNull)

Sets the valuesNull for the ParameterValue.

**Parameters** **valuesNull**  
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 5-18 lists `actuate.xtabanalyzer.Sorter` functions.

**Table 5-18** `actuate.xtabanalyzer.Sorter` functions

Function	Description
<code>getKey()</code>	Returns the sort key
<code>getLevelName()</code>	Returns the level name
<code>getMember()</code>	Returns the sort member
<code>isAscending()</code>	Returns the sort direction
<code>setAscending()</code>	Sets ascending or descending sort
<code>setKey()</code>	Sets the sort key
<code>setLevelName()</code>	Sets the level name
<code>setMember()</code>	Sets the sort member

### **actuate.xtabanalyzer.Sorter.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;
}
```

## actuate.xtabanalyzer.Sorter.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;  
}
```

## actuate.xtabanalyzer.Sorter.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;  
}
```

## actuate.xtabanalyzer.Sorter.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;  
}
```

## actuate.xtabanalyzer.Sorter.setAscending

**Syntax** void Sorter.setAscending(boolean ascending)

Sets the sort order to ascending or descending.

**Parameters** **ascending**  
Boolean. Set to true for ascending, set to false for descending.

**Example** This example swaps the sort direction:

```
sorter.setAscending(! (sorter.isAscending));
```

## actuate.xtabanalyzer.Sorter.setKey

**Syntax** void Sorter.setSortKey(string sortKey)

Sets the key to sort on.

**Parameters** **sortKey**  
String. The sort key.

**Example** This example sets the sort key:

```
function setSortKey(sorter, key) {  
    sorter.setKey(key);  
}
```

## actuate.xtabanalyzer.Sorter.setLevelName

**Syntax** void Sorter.setLevelName(string levelName)

Sets the dimension level name to sort on.

**Parameters** **levelName**  
String. A dimension level name.

**Example** This example sets the level name to sort:

```
function setSortLevel(sorter, level) {  
    sorter.setLevelName(level);  
}
```

## actuate.xtabanalyzer.Sorter.setMember

**Syntax** void Sorter.setMember(actuate.xtabanalyzer.MemberValue member)

Sets the member value to sort on.

**Parameters** **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 5-19 lists `actuate.xtabanalyzer.SubTotal` functions.

**Table 5-19** `actuate.xtabanalyzer.SubTotal` functions

Function	Description
<code>addTotal( )</code>	Add a total
<code>getLevelName( )</code>	Returns the full level name
<code>getLocation( )</code>	Returns the location
<code>getTotals( )</code>	Returns the totals array
<code>getType( )</code>	Returns the type string
<code>setLevelName( )</code>	Sets the full level name
<code>setLocation( )</code>	Sets the location
<code>setTotals( )</code>	Sets the totals array

### **actuate.xtabanalyzer.SubTotal.addTotal**

**Syntax** `void SubTotal.addTotal(actuate.xtabanalyzer.Total total)`  
Adds a total to the subtotal.

**Parameters** `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( );
}

```

## actuate.xtabanalyzer.SubTotal.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;
}

```

## actuate.xtabanalyzer.SubTotal.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;
}

```

## actuate.xtabanalyzer.SubTotal.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( );
}
```

## actuate.xtabanalyzer.SubTotal.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;
}
```

## actuate.xtabanalyzer.SubTotal.setLevelName

**Syntax** void SubTotal.setLevelName(string levelName)

Sets the level for the subtotal by name.

**Parameters** **levelName**  
String. The level name.

**Example** This example sets the level name for a subtotal:

```
function subTotalLevel(subTotal,levelName) {
    if(subTotal) {
        subTotal.setLevelName(levelName);
    }
}
```

## actuate.xtabanalyzer.SubTotal.setLocation

**Syntax** void SubTotal.setLocation(string location)

Sets the location for the subtotal.

**Parameters**    **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);
    }
}
```

## **actuate.xtabanalyzer.SubTotal.setTotals**

**Syntax**    void SubTotal.setTotals(actuate.xtabanalyzer.Total[ ] totals)

Sets the totals using an array.

**Parameters**    **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 5-20 lists `actuate.xtabanalyzer.Total` functions.

**Table 5-20** `actuate.xtabanalyzer.Total` functions

Function	Description
<code>getAggregationFunction()</code>	Returns the aggregation function name
<code>getMeasureIndex()</code>	Returns the measure index
<code>isEnabled()</code>	Returns whether or not the total is enabled
<code>setAggregationFunction()</code>	Sets the aggregation function name
<code>setEnabled()</code>	Sets the enabled flag
<code>setMeasureIndex()</code>	Sets the index for the total

### `actuate.xtabanalyzer.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");
    }
}
```

## actuate.xtabanalyzer.Total.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;  
}
```

## actuate.xtabanalyzer.Total.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);  
    }  
}
```

## actuate.xtabanalyzer.Total.setAggregationFunction

**Syntax** void Total.setAggregationFunction(string aggregationFunction)  
Sets the aggregation function name.

**Parameters** **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");  
    }  
}
```

## actuate.xtabanalyzer.Total.setEnabled

**Syntax** void Total.setEnabled(boolean enabled)

Sets whether total is enabled or disabled.

**Parameters** **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);
    }
}
```

## actuate.xtabanalyzer.Total.setMeasureIndex

**Syntax** void Total.setMeasureIndex(integer measureIndex)

Sets the measure index for the total.

**Parameters** **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 Data Analyzer viewer.

### Constructor

**Syntax** void **actuate.xtabanalyzer.UIOptions**( )

Generates a new **UIOptions** object to manage the features of the **xtabanalyzer**.

### Function summary

Table 5-21 lists **actuate.xtabanalyzer.UIOptions** functions.

**Table 5-21** actuate.xtabanalyzer.UIOptions functions

Function	Description
<b>enableCrosstabView</b> ( )	Enables the cross tab layout view feature
<b>enableCubeView</b> ( )	Enables the cube view feature
<b>enableFilterSummaryView</b> ( )	Enables the filter summary view
<b>enableToolBar</b> ( )	Enables the toolbar feature
<b>enableToolBarHelp</b> ( )	Enables the toolbar help feature
<b>enableToolBarSave</b> ( )	Enables the toolbar save feature
<b>enableToolBarSaveDesign</b> ( )	Enables the toolbar save design feature
<b>enableToolBarSaveDocument</b> ( )	Enables the toolbar save document feature
<b>getFeatureMap</b> ( )	Returns a list of enabled and disabled features

### **actuate.xtabanalyzer.UIOptions.enableCrosstabView**

**Syntax** void **UIOptions.enableCrosstabView**(boolean enabled)

Enables or disables the cross tab layout view.

**Parameters** **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);  
}
```

## actuate.xtabanalyzer.UIOptions.enableCubeView

**Syntax** void UIOptions.enableCubeView(boolean enabled)

Enables or disables the cube view.

**Parameters** **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);  
}
```

## actuate.xtabanalyzer.UIOptions. .enableFilterSummaryView

**Syntax** void UIOptions.enableFilterSummaryView(boolean enabled)

Enables or disables the filter summary view.

**Parameters** **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);  
}
```

## actuate.xtabanalyzer.UIOptions.enableToolBar

**Syntax** void UIOptions.enableToolBar(boolean enabled)

Enables or disables the toolbar feature.

**Parameters** **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);  
}
```



## **actuate.xtabanalyzer.UIOptions.enableToolbarHelp**

**Syntax** void UIOptions.enableToolbarHelp(boolean enabled)

Enables or disables the toolbar help feature.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.UIOptions.enableToolbarSave**

**Syntax** void UIOptions.enableToolbarSave(boolean enabled)

Enables or disables the toolbar save feature.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.UIOptions .enableToolbarSaveDesign**

**Syntax** void UIOptions.enableToolbarSaveDesign(boolean enabled)

Enables or disables the toolbar save design feature.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.UIOptions .enableToolbarSaveDocument**

**Syntax** void UIOptions.enableToolbarSaveDocument(boolean enabled)

Enables or disables the toolbar save document feature.

**Parameters** **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);  
}
```

## **actuate.xtabanalyzer.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

## Numerics

3D charts 155

## A

access rights. *See* privileges

access types 224, 227, 234, 238

accessing

- chart themes 32

- cross tab elements 306

- dashboard gadgets 17

- Data Analyzer 2, 306

- Encyclopedia volumes 5

- HTML buttons 27

- HTML5 charts 11

- JavaScript API 2

- JavaScript API class libraries 3, 48

- report content 10, 256

- report elements 10, 274

- report viewers 8, 12

- resources 6

- script editor 33

- source code 2

- web service applications 6

Action Details window 30

actions 28, 31, 115

- See also* events

actuate class 2, 4, 48, 52

Actuate Java Components 5

Actuate JavaScript API. *See* JavaScript API

actuate namespace 48

acviewer container 307

ad hoc parameters

- converting values 112

- defining 139

- generating query output and 129

- getting column names for 129, 144

- getting column type for 130, 144

- testing for 123, 134

addAttribute function 364

addDimension function 39, 328

adding

- bookmarks 10

- chart titles 155

- cross tabs 327

- dashboards 17, 62

- data cubes 37

- data items 159

- data service components 23, 95

- data sorters 43, 92, 388

- display names 116

- filter conditions 79, 354

- filters 31, 43, 79

- Flash gadgets 167

- Flash objects 163

- HTML buttons 27

- HTML5 charts 11, 172

- interactive features 28, 29, 40, 256

- label elements 198

- page breaks 297

- parameter components 19, 100

- parameter groups 126, 138, 148

- passwords 140

- render options 280

- report components 52

- ReportExplorer components 13, 214

- scroll panels 282

- security adapters 7

- sort conditions 92, 388

- standard charts 151

- tables 202

- text elements 210

- URL parameters 4, 6

- viewer components 8, 252

- web pages 2, 3

addLevel function 341

addMeasure function 40, 329

addMember function

- Driller 347

- MemberValue 373

addPoint function 186

addSeries function

- ClientChart 173

- ClientOption 179

addTotal function

- GrandTotal 361

- addTotal function (*continued*)
  - SubTotal 392
- advanced sort feature 290
- AFP formats 255
- aggregation 36, 37, 42, 291
- aggregation function names 368, 370, 397
- aggregation functions 42, 369, 371, 396
- AJAX requests. *See* requests
- alerts 99
- alphaNumericSorterSet array 88
- analyzing data 37, 306
- animation (charts) 187
- Apache Tomcat servers 6
  - See also* application servers
- application programming interface. *See* JavaScript API
- application servers 5, 6
- application services 4
  - See also* web services
- applications
  - accessing 6
  - developing 2, 48
  - displaying data and 23, 24, 37
  - displaying reports and 3, 8, 10, 19
  - establishing connections to 2, 4, 6, 7, 55
  - extracting subsets of data for 11
  - getting version information for 54
  - loading class libraries from 2
  - localizing. *See* localization
  - logging out of 58
  - providing security for 4–8
  - testing connections for 56
- applyOptions function
  - ClientPoint 183
  - Crosstab 329
- arc function 192
- area charts 181
- arrays 20
- ascending sort order 92, 93, 390
- asynchronous operations 31
- authenticate function 5, 7, 53
- authentication 4, 7, 53
- authentication exceptions 59
- authentication information 4, 6, 7, 58, 248
- authentication requests 59
- authentication user IDs 6, 53
- AuthenticationException class 59

- AuthenticationException objects 59
- autosave feature 64, 68
- autosuggest delays 106
- autosuggest threshold values 129, 135
- autosuggestion lists 107, 122, 137
  - See also* parameter lists
- axis labels (charts) 181, 182
- axis type values (cross tabs) 344
- axis types (cross tabs)
  - changing 41, 346
  - getting 343, 362
  - setting 344, 345, 363
- axis values (charts) 155, 176, 180, 189

## B

- bandwidth 126
- bar charts 181
- base class 2
- batch operations 231
- BETWEEN operator 79, 354
- BIRT Data Analyzer. *See* Data Analyzer
- BIRT Designer 26
- BIRT Designer Professional 33, 37
- BIRT Interactive Viewer. *See* Interactive Viewer
- BIRT reports. *See* reports
- BIRT Viewer 2
  - See also* viewers
- bookmark names 86, 198, 210
- bookmark parameter 23
- bookmarks
  - accessing cross tabs and 306, 317, 332, 383
  - adding Flash objects and 164, 260, 275
  - displaying report elements and 9, 23, 87, 263
  - displaying Reportlets and 261, 267
  - getting chart instance for 257, 274
  - getting chart names for 153
  - getting data item for 159, 259, 275
  - getting gadgets associated with 168, 260, 276
  - getting labels associated with 198, 261, 276
  - getting report content for 258, 259, 261
  - getting text element for 210, 262, 277
  - navigating through reports and 263
  - retrieving result sets and 85

- returning table objects for 31, 203, 261, 276
- returning viewer objects for 263
- sending requests for 86, 87
- setting 10
- Boolean expressions 79, 80, 354, 355
- bottom N filter feature 302
- BOTTOM\_N operator 79, 354
- BOTTOM\_PERCENT operator 79, 354
- BrowserPanel class 272
- browsers. *See* web browsers
- bullet gadgets 170
- button constants (navigation) 104
- button elements 27, 104, 137
- button events 27

## C

- calculated columns 291
  - See also* computed columns
- callback functions
  - authenticating users and 6, 8
  - closing Data Analyzer and 321
  - connecting to web services and 6, 56
  - defined 2
  - displaying dashboards and 18
  - displaying data and 23, 24
  - displaying reports and 9, 14, 16
  - downloading parameters and 101
  - filtering data and 11
  - handling exceptions and 59, 61, 97
  - hiding report elements and 11
  - retrieving parameters and 20, 102, 112
  - retrieving result sets and 24, 90, 95, 256
  - sorting data and 12
- cascading parameter names 129, 136
- cascading parameters 115, 119, 123
- category series (charts) 152, 175, 178
  - See also* x-axis values
- categoryData variable 31
- cells (empty) 330, 378, 380
  - See also* cross tabs
- changeMeasureDirection function 41, 330
- changes, undoing 302
- changing
  - aggregation functions 369, 371
  - chart subtype 157
  - chart themes 33

- chart titles 152, 156
- cross tabs 38, 39, 41, 307
- data 292
- data series 177
- gadget types 294
- HTML5 charts 11
- label elements 201
- parameters 100, 115, 126
- report designs 312
- report output 19
- reports 293
- tables 10
- text elements 10, 300
- user interface options 10
- character encoding 3
- character encryption 7
- character filters. *See* filters
- character patterns 79, 80
- chart areas 28
- chart bookmarks 153, 257, 274
- chart builder 28
- Chart class 151
- chart dimension constants 155
- chart elements
  - constructing 151, 172
  - determining type 154
  - displaying 157
  - getting bookmarks for 257, 274
  - hiding 155
  - setting size 156
- chart gadgets 170
  - See also* dashboards
- chart IDs 153, 154
- chart interactive features 28
- chart legends 28
- chart properties 291
- chart subtypes 157, 291
- chart themes 32
- chart titles 152, 153, 155, 156
- chart types 180
- CHART\_DIMENSION\_2D constant 155
- CHART\_DIMENSION\_2D\_WITH\_DEPTH
  - constant 155
- CHART\_SUBTYPE\_PERCENTSTACKED
  - constant 157
- CHART\_SUBTYPE\_SIDE BY SIDE
  - constant 157

- CHART\_SUBTYPE\_STACKED constant 157
- charts (standard)
  - See also* HTML5 charts
  - accessing themes for 32
  - adding interactive features to 28, 29
  - changing subtype of 157
  - changing titles for 152, 156
  - clearing data filters for 152
  - displaying 28
  - drilling through 152, 153
  - getting bookmark name for 153
  - getting HTML element for 153
  - getting page associated with 154
  - instantiating 151
  - selecting subtypes for 291
  - setting filters for 156
  - setting number of dimensions for 155
  - setting title for 155
  - submitting requests for 157
- check boxes 137
- circle function 193
- class libraries 2, 48, 57
- class reference 49, 51, 309
- classes 2, 26, 48, 307
  - See also* JavaScript API
- cleanData function 187
- clearFilters function
  - Chart 152
  - FlashObject 163
  - Gadget 167
  - Table 203
  - XTabAnalyzer 43, 330
- ClientChart class 172
- ClientChart objects 11, 172
- ClientOption class 179
- ClientOption objects 11, 179
- ClientPoint class 183
- ClientSeries class 186
- client-side error constants 111, 221
- client-side errors 97, 352
- clipping rectangles 194
- clipRect function 194
- closing HTTP sessions 7
- code
  - accessing JavaScript API source 2
  - adding chart interactive features and 30
  - constructing requests and 23
  - displaying cross tabs and 39, 40, 43, 306
  - displaying dashboards and 18
  - displaying parameters and 20, 21
  - displaying reports and 10
  - embedding 12
  - enabling user interface options and 10, 44
  - initializing HTTP sessions and 4
  - initializing HTTPS sessions and 6
  - registering event handlers and 38
  - running 27
- collapse/expand feature 292
- column editing feature 292
- column headings 36
  - See also* column names
- column index values
  - accessing result sets and 91
  - displaying cross tabs and 333
  - getting 91, 203, 285
- column mirror starting level
  - getting 377
  - setting 330, 376, 380
- column name parameter 31
- column names
  - displaying charts and 31
  - displaying parameters and 129, 136, 144, 147, 279
  - filtering data and 81, 82
  - getting list of 86, 90
  - sorting data and 92, 93
- column types
  - parameter definitions and 130, 136
  - parameter value objects and 144, 147
- COLUMN\_AXIS\_TYPE constant 344
- columnMirrorStartingLevel parameter 330, 376
- columnPageBreakInterval parameter 376
- columns
  - adding to cross tabs 377
  - changing data in 292
  - filtering values in 152
  - getting 203, 333
  - hiding 11, 206
  - matching top or bottom values in 79
  - moving 296
  - reordering 209, 298
  - resizing 292
  - retrieving data values in 91, 129

- retrieving index values for 91, 203, 285
- retrieving result sets and 24, 87, 91
- selecting 285
- setting page breaks on 378, 380, 381
- showing calculated 291
- showing summary data in 42
- showing table 208
- sorting on 12, 92, 93
- commit function 312
- component names (reports) 57
- components. *See* Actuate Java Components; report components
- computed columns 369, 371
  - See also* calculated columns
- computed measures 369, 371
  - See also* cross tabs
- connection exceptions 61
- connection information 5
- connection parameters 4, 55
- ConnectionException class 61
- ConnectionException objects 61
- connections
  - accessing Encyclopedia volumes and 5
  - authenticating users and 4, 5, 53
  - closing 58
  - displaying dashboards and 69
  - failing 61
  - initializing error handlers for 61
  - loading JavaScript classes and 26
  - logging in to web applications and 6, 7, 95
  - opening 2, 4, 55
  - testing 56
- Constants class 111, 221
- containerID parameter 17
- content components 8, 10, 13, 84
- content panels (viewer) 272, 282, 287, 288
- content variable 308
- context menus 301
- control type UI values 120
- control types 130, 137
- controls 28
- convert function 112
- convertDate function 113
- ConvertUtility class 112
- creating
  - autosuggestion lists 107
  - bookmarks 10
- cross tabs 37, 327
- dashboards 17, 62
- data cubes 37
- data service components 23, 95
- data sorters 43, 92, 388
- display names 116
- filter conditions 79, 354
- filters 31, 43, 79
- Flash gadgets 167
- Flash objects 163
- HTML buttons 27
- HTML5 charts 11, 172
- label elements 198
- parameter components 19, 100
- parameter groups 126, 138, 148
- passwords 140
- report explorer 13, 214
- requests 23
- result sets 90
- scroll panels 282
- security adapters 7
- sort conditions 92, 388
- standard charts 151
- tables 202
- text elements 210
- URL parameters 4, 6
- viewer components 8, 252
- web pages 2, 3
- credentials. *See* user credentials
- cross tab analyzer. *See* Data Analyzer
- cross tab bookmarks 306, 317, 332, 383
- cross tab elements 39, 306, 335
  - See also* cross tabs
- cross tab filter objects 354
- cross tab gadgets 313, 320
  - See also* dashboards
- cross tab layout view 399
- cross tab level objects. *See* level objects
- cross tab objects 307, 327
  - See also* cross tabs
- cross tab report elements 307, 327
- cross tab Reportlets 323
- cross tab viewer. *See* Data Analyzer
- cross tabs
  - accessing 306
  - adding dimensions to 39, 328, 341
  - adding measures to 40, 329, 368

- cross tabs (*continued*)
  - changing 38, 39, 41, 307
  - creating 37, 327
  - displaying 36, 39, 306, 383
  - drilling through 43, 44, 330, 331, 347
  - enabling interactive features for 40
  - filtering data in 43, 330, 338, 354, 400
  - getting bookmarks for 317, 332
  - getting columns in 333
  - getting empty cell values for 378
  - getting level values in 375
  - getting measure direction for 379
  - getting page breaks for 378, 379
  - getting rows in 334
  - handling errors for 351
  - handling events for 38, 318, 350
  - hiding detail data in 46, 335
  - loading 306
  - pivoting elements in 41, 335
  - removing dimensions from 40, 335
  - removing measures from 41, 337
  - removing summary data in 42
  - rendering 326
  - reordering elements in 41, 336, 337
  - retrieving data for 37, 40, 333, 334
  - selecting elements in 350
  - setting display options for 329, 376
  - setting empty cell values in 330, 380
  - setting level values for 375
  - setting measure direction for 381
  - setting page breaks for 380, 381, 382
  - sorting data in 12, 43, 338, 388
  - submitting changes to 340
  - switching measure direction in 330
  - updating 44
  - viewing detail data in 339
  - viewing summary data in 42, 361, 392, 396
- crossContext parameter 6
- crosstab analyzer. *See* Data Analyzer
- Crosstab class 39, 327
- Crosstab objects 307, 327
  - See also* cross tabs
- crosstab variable 308
- CSS position attribute 315, 322
- cube view 400
- cubes 37, 39, 306
- custom security adapters 7

- customizing
  - autosuggestion lists 107
  - security 5
  - URL parameters 4, 6, 53
  - user interfaces 289, 399
  - web pages 2
- cylinder gadgets 170

## D

- Dashboard class 17, 62
- dashboard components 57
  - See also* dashboards
- dashboard definitions 63, 66, 72
- dashboard event constants 73
- dashboard files 17
- dashboard metadata 66
- dashboard names 64, 68
- dashboard objects 62
- dashboard page fragment (HTML) 68
- dashboard tab names 72, 77
- dashboard tab objects 77
- dashboard tab toolbar 70
- dashboard tab type constants 77
- dashboard tabs 68, 72, 75, 76, 77
- dashboard templates 64, 70
- DASHBOARD\_MAX constant 269
- DASHBOARD\_NORMAL constant 269
- DashboardDefinition class 72
- dashboards
  - accessing gadgets in 17
  - adding tabs to 77
  - changing gadgets in 74
  - closing 65
  - creating 17, 62
  - determining status of 318
  - displaying 17
  - downloading 63
  - getting active tab for 64, 72
  - getting content for 66
  - getting tabs in 72
  - handling events for 66, 73, 74
  - loading 17
  - saving 64, 65, 67
  - setting active tab for 67, 68
  - setting auto save delay for 68
  - setting size of 69, 70



- setting viewing mode for 269
- setting web service connection for 69
- showing gadget gallery for 70
- showing personal 71
- showing tab toolbar in 70
- submitting requests for 71
- viewing cross tabs and 313
- viewing gadgets and 320
- data
  - See also* data elements; data items; values
  - analyzing 37, 306
  - changing 292
  - displaying 24, 376
  - downloading 84
  - extracting 293
  - filtering. *See* filters
  - hiding 11, 46, 161, 206, 335
  - preventing loss of 3
  - retrieving 2, 22, 23, 90, 95
  - returning subsets of 11
  - selecting 285
  - sorting. *See* sorters; sorting data
  - submitting requests for 85
  - summarizing 36
- Data Analytics module 308
- Data Analytics viewer 321
- Data Analyzer
  - accessing 2, 306
  - adding toolbar to 400, 401, 402
  - building user interface for 44
  - changing CSS position attribute for 322
  - closing 321
  - determining status of 317, 318
  - displaying cross tabs and 37, 39, 383
  - displaying data cubes and 400
  - drilling in 347
  - enabling 293
  - getting content for 313, 383
  - getting CSS position attribute for 315
  - getting current instance 317
  - getting feature map for 402
  - getting ID for 384
  - getting margins for 314, 315
  - getting parameters for 385
  - getting size 314, 316
  - getting UI options for 315
  - handling errors for 351
  - handling events for 38, 350
  - hiding features of 44, 45
  - initializing 307
  - instantiating 306
  - integrating with Interactive Viewer 317, 321, 326
  - loading 306
  - resizing 319, 320, 325
  - restarting 312
  - restoring initial state 319
  - retrieving 316
  - rolling back changes to 320
  - sessions timing out and 350
  - setting display options for 329, 376
  - setting margins for 321, 324
  - setting UI options for 324, 399
  - submitting requests for 307, 326
- Data Analyzer API 306
- Data Analyzer API class reference 309
- Data Analyzer API classes 307
- Data Analyzer API event constants 350
- Data Analyzer API usage errors 352
- data analyzer component 57
- Data Analyzer objects 310
- Data Analyzer toolbars
  - enabling help feature for 401
  - enabling or disabling 400
  - enabling save design feature for 401
  - enabling save document feature for 402
  - enabling save feature for 401
- Data Analyzer viewer 310, 384
  - See also* Data Analyzer
- data classes 49
- data cubes. *See* cubes
- data elements 159, 259, 275
  - See also* data
- data extraction feature 293
- data fields 36
  - See also* cross tabs
- data filters. *See* filters
- data hierarchies (cubes) 37, 39, 42
  - See also* level objects
- data item IDs 160
- data item objects 159
- data items 159, 160, 161
  - See also* data; data elements
- data point arrays 189

- data point options 183, 186
- data points (charts) 183, 186
- data repositories 5, 13, 214
  - See also* Encyclopedia volumes
- data repository access rights 245, 246
- data repository file paths 15
- data repository type constants 251
- data repository types 249, 251
- data rows. *See* rows
- data series (charts)
  - adding 173, 179, 186
  - changing 177
  - deleting 176, 187
  - displaying values 28
  - drilling through 152, 153
  - getting number of run-time 174
  - getting values for 175, 176
  - managing 186
  - removing 188
  - replacing 188
  - setting values for 177, 178
  - setting visibility for 189
  - setting visibility of 177, 181
- data series names 31
- data series objects 186, 187
- data service components 23, 57
- data service objects 95
- data services 22, 26, 95
- data sets 11
- data sorters 12, 43
  - See also* sorters
- data types
  - computed columns 369, 371
  - parameter definitions 130, 131, 137
  - parameter values 145, 148
- DataItem class 159
- DataService class 22, 23, 95
- date values 79, 113
- default iHub URL 54
- default parameters 53
- default request options 54
- default settings 4, 6
- default values 102, 131, 138
- default web service URL 54
- delays 106
- deleting
  - authentication information 7, 58
  - data groups 207
  - data series 176, 187, 188
  - dimensions 40, 335
  - duplicate values 187
  - event handlers 39, 105, 318
  - filters 43
  - measures 41, 337
  - summary data 42
- dependencies 235, 239
- Deployment Kit 2, 5, 59
- descending sort order 92, 93, 390
- design files 19, 109, 267
  - See also* designs
- designers 26
- designing reports 26
- designs 264, 298, 312
- destroy function
  - ClientPoint 184
  - ClientSeries 187
  - HTML5Chart Renderer 194
- detail data 46, 206, 208, 335, 339
- developing web applications 2, 48
- dialog boxes
  - displaying cross tabs and 306
  - displaying dashboards and 17
  - exporting data and 269
  - loading 8, 26
  - printing reports and 270
  - viewing reports and 8, 22
  - viewing result sets and 270
- Dialog class 26
- dialog components 57
- dialog event constants 273
- Dimension class 39, 41, 341
- dimension index values 41, 337, 345, 346
- dimension names 342, 344
- dimension objects 341
- dimensions
  - See also* cross tabs; cubes
  - adding levels to 341, 345, 364, 367
  - adding to cross tabs 39, 328, 341
  - changing axis type 41, 337, 346
  - creating data cubes and 37
  - drilling through 43, 330, 331, 347, 348
  - expanding or collapsing members in 43, 339
  - filtering 356, 358, 359, 360

- generating summary values for 42
  - getting level values in 375
  - getting levels 343, 357
  - hiding detail data in 46, 335
  - naming 344
  - removing 40, 335
  - reordering 41, 336
  - setting axis type for 344, 345
  - setting index values for 345, 346
  - sorting values 388, 389, 390
  - viewing charts and 155
  - DIRECTION\_HORIZONTAL value 376, 379, 381
  - DIRECTION\_VERTICAL value 376, 379, 381
  - directory paths 13, 15, 64, 70
  - disableIV function 254
  - display names
    - parameter definitions 130, 132, 137, 138
    - parameter values 116, 145, 148
  - displayData function 23, 24
  - displaying
    - aggregate values 36
    - charts 28, 153, 157
    - columns 208
    - cross tabs 36, 39, 306, 383
    - dashboards 17
    - data 24, 160, 161, 376
    - Data Analyzer features 44
    - data cubes 37, 400
    - data series 177, 181, 189
    - Flash objects 163, 166, 171
    - HTML5 charts 190
    - label elements 200, 261
    - parameter groups 108
    - report elements 10
    - report parameters 20, 100, 126
    - Reportlets 261, 267
    - reports 8, 19, 22, 252, 274
    - summary data 42, 361, 392, 396
    - table elements 208
    - table of contents 287, 288, 300
    - tables 10, 31, 202
    - text 211, 212, 262
    - toolbars 300, 400
    - tooltips 299
  - displayname variable 130
  - displayReport function 15
  - div tag 3, 8, 13, 306
  - DOCTYPE tag 3
  - document files 103, 109, 267, 323
    - See also* documents
  - document output formats 255
  - documentation v
  - documentation URLs 260, 266
  - documents 265, 298, 306, 312
    - See also* reports
  - domains 6
  - download result set dialog 270
  - downloadDashboard function 63
  - downloading
    - dashboards 63
    - data 84
    - report parameters 19, 20, 101, 102
    - reports 255
    - result sets 23, 95, 256, 270
  - downloadParameters function 101
  - downloadParameterValues function 20, 102
  - downloadReport function 255
  - downloadResultSet function
    - DataService 23, 95
    - Viewer 256
  - drawing elements (Highchart) 192
  - drill function 43, 330
  - drillDown function 331
  - drillDownCategory function 152
  - drillDownSeries function 152
  - Driller class 347
  - Driller objects 43, 347
  - drilling 43
  - drillUp function 331
  - drillUpCategory function 152
  - drillUpSeries function 153
  - duplicate values 187
  - duplication suppression feature 299
  - dynamic filters 123
- ## E
- Easyscript 40
    - See also* expressions
  - editing. *See* changing
  - editMeasure function 41, 332
  - embedTemplate function 63

- empty cells 330, 378, 380
  - See also* cross tabs
- emptyCellValue parameter 330, 376
- enableAdvancedSort function 290
- enableAggregation function 291
- enableCalculatedColumn function 291
- enableChartProperty function 291
- enableChartSubType function 291
- enableCollapseExpand function 292
- enableColumnEdit function 292
- enableColumnResize function 292
- enableContentMargin function 292
- enableCrosstabView function 399
- enableCubeView function 400
- enableDataAnalyzer function 293
- enableDataExtraction function 293
- enableEditReport function 293
- enableExportReport function 293
- enableFacebookComments function 294
- enableFilter function 294
- enableFilterSummaryView function 400
- enableFlashGadgetType function 294
- enableFormat function 294
- enableGroupEdit function 295
- enableHideShowItems function 295
- enableHighlight function 295
- enableHoverHighlight function 295
- enableIV function 256
- enableLaunchViewer function 296
- enableLinkToThisPage function 296
- enableMainMenu function 296
- enableMoveColumn function 296
- enablePageBreak function 297
- enablePageBreak parameter 376
- enablePageNavigation function 297
- enableParameterPage function 297
- enablePrint function 297
- enableReorderColumns function 298
- enableRowResize function 298
- enableSaveDesign function 298
- enableSaveDocument function 298
- enableShowToolTip function 299
- enableSort function 299
- enableSuppressDuplicate function 299
- enableSwitchView function 299
- enableTextEdit function 300
- enableTOC function 300
- enableToolBar function
  - Viewer.UIOptions 10, 300
  - XTabAnalyzer.UIOptions 400
- enableToolBarContextMenu function 301
- enableToolBarHelp function
  - Viewer.UIOptions 301
  - XTabAnalyzer.UIOptions 401
- enableToolBarSave function 401
- enableToolBarSaveDesign function 401
- enableToolBarSaveDocument function 402
- enableTopBottomNFilter function 302
- enableUndoRedo function 302
- encapsulation 3
- encoding 3
- encryption 7
- Encyclopedia volume file paths 15, 64
- Encyclopedia volume names 53, 56
- Encyclopedia volumes 5, 249, 251
  - See also* repositories
- end parameter 23
- enterprise repository type 251
- EQ operator 12, 31, 79, 354
- ERR\_CLIENT constant
  - parameter 111
  - ReportExplorer 221
- ERR\_CLIENT exception type
  - Exception 97
  - ViewerException 303
  - XTabAnalyzer.Exception 351
- ERR\_SERVER constant
  - parameter 111
  - ReportExplorer 221
- ERR\_SERVER exception type
  - Exception 97
  - ViewerException 303
  - XTabAnalyzer.Exception 351
- ERR\_USAGE constant
  - parameter 111
  - ReportExplorer 221
- ERR\_USAGE exception type
  - Exception 97
  - ViewerException 303
  - XTabAnalyzer.Exception 351
- error callback functions 59, 61, 96
- error codes 98, 352
- error constants 111, 221
- error descriptions 97, 351

- error messages 98, 304, 352
- errorcallback function 6, 8
- errorCallback parameter 4
- errors 4, 54, 56
  - See also* exceptions
- event constants. *See* EventConstants class
- event functions 27
- event handlers
  - creating HTML buttons and 27
  - creating interactive charts and 30
  - designing reports and 26
  - displaying cross tabs and 38, 318, 350
  - displaying dashboards and 66, 73, 74
  - displaying parameters and 105
  - displaying reports and 273
  - exceptions and 97
  - navigating repository content and 15, 216, 222
  - registering 105, 318
  - removing 39, 105, 318
  - running scripts and 34
  - selecting report elements and 285
- EventConstants class
  - Dashboard 73
  - Parameter 115
  - ReportExplorer 222
  - Viewer 273
  - XTabAnalyzer 350
- events 26, 27, 28, 34, 38
  - See also* event handlers
- evt variable 31
- Excel output formats 255
- Excel spreadsheets 280
- Exception class 39, 97, 351
- exception classes 59, 61, 303
- exception events. *See* ON\_EXCEPTION constant
- exception objects 97
- exception types
  - constructing exception objects and 97
  - getting 98, 353
  - testing for 98, 353
- exceptions
  - authentication and 6, 8, 59
  - connections and 61
  - cross tabs and 38, 350, 351
  - data service components and 23
  - report parameters and 115
  - viewer and 273, 303
- executable files 19
- explodePieSlice function 180
- explorer. *See* ReportExplorer components
- explorerpane parameter 13
- export report dialog 269
- exporting reports 255, 269, 293
- expressions
  - aggregating data values and 40
  - computing cross tab values and 40, 369, 371
  - creating Easyscript 40
  - filtering data and 79, 354
- external user credentials 6

## F

- Facebook comments 294
- Facebook comments panel 270
- failed connections 61
- failed requests 59
- FALSE operator 79, 354
- feature maps 302, 402
- features 287, 289, 399
  - See also* interactive features; user interface options
- fetch direction (FileSearch) 236, 239
- fetch handle (FileSearch) 236, 240
- fetch handle (FolderItems) 242, 243
- fetch size (FileSearch) 236, 240
- fields 36
  - See also* columns
- file access types 224, 227, 234, 238
- file attributes 13
- File class 223
- file dependencies 235, 239
- file descriptions 224, 227
- file explorer. *See* ReportExplorer components
- file IDs 225, 227, 235, 239
- file lists 223, 231
- file name extensions 224
- file names
  - getting 103, 225, 237, 261
  - rendering reports and 267
  - setting 228, 241, 264, 265
- file objects 223

- file owners 225, 228, 237, 240
- file paths 13, 15, 64, 70
- file search objects 216, 218, 233
- file size 225, 228
- file system interface 214
- file system repositories 5
  - See also* repositories
- file types 224, 227
- FileCondition class 231
- filedatasource parameter 23
- files. *See* report files
- FileSearch class 233
- FileSearch objects 216, 218, 233
- Filter class 43, 79, 354
- filter conditions
  - adding 79, 354
  - getting operator in 357
  - getting values of 82, 358
  - setting operator in 82, 359
  - setting values for 83, 360
- filter expressions 79, 354
- filter objects 31, 79, 354
- filter operators 79, 81, 82, 354
- filter strings 231
- filter summary view 400
- filtering
  - data. *See* filters
  - file lists 231, 237, 241
- filters
  - adding gadgets and 167, 170
  - adding table elements and 203, 207
  - clearing 43, 163, 203
  - creating 31, 43, 79
  - determining if dynamic 123
  - displaying charts and 30, 152, 156
  - displaying cross tabs and 330, 338, 354
  - displaying Flash objects and 163, 166
  - enabling or disabling 294, 302
  - getting column names for 81
  - getting type for 356
  - retrieving data and 11, 79
  - setting column names for 82
  - setting level attribute names for 358
  - setting level names for 359
  - submitting requests and 86, 88
- Firefox browsers 31, 97
  - See also* web browsers

- FirstTable parameter 10
- Flash charts 155, 166, 170
- Flash object elements 165
- Flash objects 163, 164, 260, 275
  - See also* gadget elements; gadgets
- FlashObject class 163
- focus 265
- folder labels 219
- folder paths 219
- FolderItems class 242
- folders 214, 215, 217, 219
- fonts 108
- forceSoftRestart function 312
- forcing logins 6
- format editing feature 294
- formats 79, 255
- functions 2, 3, 26, 51, 306
  - See also* callback functions

## G

- g function 194
- Gadget class 167
- gadget elements 167, 276
  - See also* gadgets
- gadget gallery 70
- gadget IDs 313
- gadget names 75
- gadget objects 167
  - See also* gadgets
- gadget script objects 74
- gadget type change control 294
- gadget types 170
- GADGET\_TYPE\_BULLET constant 170
- GADGET\_TYPE\_CYLINDER constant 170
- GADGET\_TYPE\_LINEARGAUGE
  - constant 170
- GADGET\_TYPE\_METER constant 170
- GADGET\_TYPE\_SPARK constant 170
- GADGET\_TYPE\_THERMOMETER
  - constant 170
- gadgets
  - See also* Flash objects
  - accessing 17
  - adding to dashboards 66
  - changing 74, 294
  - displaying 171

- filtering 167, 170
- getting instance of 168, 260
- getting title of 75
- getting type 75
- hiding 169
- retrieving bookmarks for 168
- setting size 170
- specifying type 170
- GadgetScript class 74
- garbage collection 105
- generating
  - query output 129, 130, 136, 147
  - report components 52
  - reports 103
- getAccessRights function 245
- getAccessType function
  - File 224
  - FileSearch 234
- getActiveTab function 64
- getAggregationFunction function
  - Measure 368
  - Total 396
- getAttributes function 364
- getAutoSuggestThreshold function 129
- getAxisType function
  - Dimension 342
  - GrandTotal 362
- getBookmark function
  - Chart 153
  - Crosstab 332
  - DataItem 159
  - FlashObject 164
  - Gadget 168
  - Label 198
  - Request 86
  - Table 203
  - TextItem 210
- getCascadingParentName function 129
- getCascadingParentValues function 119
- getCategoryCount function 173
- getChart function 257
- getChartByBookmark function 274
- getChartHeight function 173
- getChartWidth function 174
- getChildData function 120
- getClientChart function 153
- getClientHeight function 257
- getClientOptions function 174
- getClientWidth function 257
- getColumn function
  - Crosstab 333
  - Table 203
- getColumnIndex function 285
- getColumnMirrorStartingLevel function 377
- columnName function
  - Filter 81
  - ParameterDefinition 129
  - ParameterValue 144
  - Sorter 92
- getColumnNames function 90
- getColumnPageBreakInterval function 378
- getColumns function 86
- getColumnType function
  - ParameterDefinition 130
  - ParameterValue 144
- getCondition function 234
- getConditionArray function 234
- getContentByBookmark function 258
- getContentByPageRange function 258
- getContentMargin function 258
- getContentPanel function 287
- getControlType function
  - ParameterData 120
  - ParameterDefinition 130
- getCore function 174
- getCountLimit function 235
- getCrosstabByBookmark function 383
- getCurrentDisplayName function 130
- getCurrentPageContent function
  - Viewer 10, 259
  - XTabAnalyzer 313
- getCurrentPageNum function
  - Viewer 259
  - XTabAnalyzer 313
- getCurrentReportParameters function 75
- getCurrentValue function 120
- getDashboardName function 64
- getData function
  - Crosstab 333
  - DataItem 159
- getDataItem function 259
- getDataItemByBookmark function 275
- getDataType function
  - ParameterDefinition 131

- getDataType function (*continued*)
  - ParameterValue 145
  - XTabAnalyzer.Measure 369
- getDefaultActiveTab function 72
- getDefaultPortalUrl function 54
- getDefaultRequestOptions function 54
- getDefaultValue function
  - ParameterData 120
  - ParameterDefinition 131
- getDefaultValueIsNotNull function 131
- getDependentFileId function 235
- getDependentFileName function 235
- getDescription function
  - Exception 97
  - ReportExplorer.File 224
  - XTabAnalyzer.Exception 351
- getDimension function 348
- getDimensionName function 342
- getDisplayName function
  - ParameterDefinition 131
  - ParameterValue 145
- getElement function
  - ViewerException 303
  - XTabAnalyzer.Exception 352
- getEmptyCellValue function 378
- getEnablePageBreak function 378
- getErrCode function
  - Exception 98
  - XTabAnalyzer.Exception 352
- getErrorMessage function 304
- getExpression function 369
- getFeatureMap function
  - Viewer.UIOptions 302
  - XTabAnalyzer.UIOptions 402
- getFetchDirection function 236
- getFetchHandle function
  - FileSearch 236
  - FolderItems 242
- getFetchSize function 236
- getField function 231
- getFileType function 224
- getFilters function 86
- getFilterType function 356
- getFlashObject function 260
- getFlashObjectByBookmark function 275
- getFolderName function 215
- getGadget function 260
- getGadgetByBookmark function 276
- getGadgetId function 313
- getGadgetName function 75
- getGadgetTitle function 75
- getGadgetType function 75
- getGrantedRoleId function 245
- getGrantedRoleName function 245
- getGrantedUserId function 245
- getGrantedUserName function 246
- getGroup function
  - ParameterDefinition 132
  - ParameterValue 145
- getHeight function
  - Viewer 260
  - XTabAnalyzer 314
- getHelpBase function 260
- getHelpText function
  - ParameterData 121
  - ParameterDefinition 132
- getHtmlDom function
  - Chart 153
  - Crosstab 333
  - DataItem 160
  - FlashObject 164
  - Gadget 168
  - Label 198
  - Table 204
  - TextItem 211
- getId function 224
- getIncludeHiddenObject function 236
- getIndex function
  - Dimension 343
  - Level 365
  - Measure 369
- getInstanceId function
  - Chart 154
  - DataItem 160
  - FlashObject 164
  - Gadget 168
  - Label 199
  - Table 204
  - TextItem 211
- getIpportalUrl function 59
- getIserverUrl function 248
- getItemList function 242
- getKey function 388



- getLabel function
  - Label 199
  - Viewer 261
- getLabelByBookmark function 276
- getLatestVersionOnly function 215
- getLayout function 102
- getLeft function 314
- getLevelAttributeName function 356
- getLevelName function
  - Filter 357
  - Level 365
  - MemberValue 374
  - Sorter 389
  - SubTotal 393
- getLevels function 343
- getLocale function 249
- getLocation function 393
- getMatch function 231
- getMaxRows function 86
- getMeasureDirection function 379
- getMeasureIndex function 397
- getMeasureName function 370
- getMember function 389
- getMembers function
  - Driller 348
  - MemberValue 374
- getMessage function
  - Exception 98
  - XTabAnalyzer.Exception 352
- getMouseScrollingEnabled function 282
- getName function
  - File 225
  - NameValuePair 116
  - ParameterDefinition 132
  - ParameterValue 145, 278, 385
  - Tab 77
  - XTabAnalyzer.LevelAttribute 367
- getNameValueList function 121
- getNewAxisType function 343
- getNewIndex function
  - Dimension 344
  - Measure 370
- getOperator function
  - Data.Filter 81
  - XTabAnalyzer.Filter 357
- getOperatorList function 132
- getOptions function 280
- getOwner function
  - File 225
  - FileSearch 237
- getPageContent function
  - Chart 154
  - Crosstab 334
  - DataItem 160
  - FlashObject 165
  - Gadget 169
  - Label 199
  - Table 204
  - TextItem 211
- getPageCount function 225
- getPanInOutEnabled function 283
- getParameterGroupNames function 103
- getParameterMap function 113
- getParameterName function 121
- getParameterValues function
  - ConvertUtility 114
  - XTabAnalyzer 314
- getParentData function 121
- getPickList function 122
- getPosition function
  - ParameterDefinition 133
  - ParameterValue 146
  - XTabAnalyzer 314
- getPrivilegeFilter function 237
- getPromptParameter function 146
- getPromptText function 122
- getReportletBookmark function 261
- getReportName function
  - Parameter 103
  - Viewer 261
- getRepositoryType function 249
- getRequestOptions function 59
- getRequiredFileId function 237
- getRequiredFileName function 237
- getResultDef function 215
- getRow function
  - Crosstab 334
  - Table 205
- getRowMirrorStartingLevel function 379
- getRowPageBreakInterval function 379
- getScrollControlEnabled function 283
- getSearch function 216
- getSelectedElement function 285
- getSelectNameValueList function 133

- getSelectValueList function 133
- getSeriesCount function 174
- getShowToc function 287
- getSize function 225
- getSorters function 87
- getStartRow function 87
- getSuggestionList function 122
- getTable function 261
- getTableByBookmark function 31, 276
- getTabName function 75
- getTabs function 72
- getTabTitle function 76
- getTabType function 77
- getTemplate function 64
- getText function
  - TextItem 212
  - Viewer 262
- getTextByBookmark function 277
- getTextContent function 84
- getTimeStamp function 226
- getTitle function (dashboard) 78
- getTop function 315
- getTotalCount function 243
- getTotalPageCount function
  - Viewer 262
  - XTabAnalyzer 315
- getTotals function
  - GrandTotal 362
  - SubTotal 394
- getTransientDocumentName function 103
- getType function
  - Chart 154
  - Crosstab 335
  - DataItem 161
  - Exception 98
  - FlashObject 165
  - Gadget 169
  - GrandTotal 363
  - Label 200
  - SubTotal 394
  - Table 205
  - TextItem 212
  - XTabAnalyzer.Exception 353
- getUIConfig function 262
- getUIOptions function
  - Viewer 262
  - XTabAnalyzer 315
- getUrl function 61
- getUserId function 60
- getUserPermissions function 226
- getValue function
  - NameValuePair 117
  - parameter.ParameterValue 146
  - ResultSet 91
  - viewer.ParameterValue 278
  - XTabAnalyzer.MemberValue 375
  - XTabAnalyzer.ParameterValue 386
- getValuesNull function
  - parameter.ParameterValue 146
  - viewer.ParameterValue 279
  - XTabAnalyzer.ParameterValue 386
- getValues function
  - Data.Filter 82
  - XTabAnalyzer.Filter 358
- getVersion function
  - actuate 54
  - ReportExplorer.File 226
- getVersionName function 226
- getViewer function
  - actuate 31, 55
  - Viewer 263
  - XTabAnalyzer 316
- getViewerId function
  - Viewer.PageContent 277
  - XTabAnalyzer.PageContent 384
- getVolume function 249
- getVolumeProfile function 249
- getWidth function
  - Viewer 263
  - XTabAnalyzer 316
- getXAxisMax function 175
- getXAxisMin function 175
- getXTabBookmark function 316
- getXTabId function 317
- getYAxisMax function 175
- getYAxisMin function 176
- global constants 111, 221
- global reporting applications. *See* localization
- gotoBookmark function 263
- gotoPage function 264
- grand totals 42, 361
- GrandTotal class 42, 361
- grantedRoleId value 245, 246
- grantedRoleName value 245, 247

- grantedUserId value 245, 247
- grantedUserName value 246, 247
- graphical user interfaces. *See* user interfaces
- graphics elements 195
- graphs. *See* charts
- GREATER\_THAN operator 79, 354
- GREATER\_THAN\_OR\_EQUAL operator 79, 354
- group editing feature 295
- groupBy function 205
- grouping data rows 27, 205
- groups, removing 207
- GUIs. *See* user interfaces

## H

- headers 28
  - See also* column names
- help 260, 266, 301, 401
- help text 121, 132, 139
- hidden files 237, 240
- hide function
  - Chart 155
  - ClientSeries 187
  - DataItem 161
  - FlashObject 165
  - Gadget 169
  - Label 200
  - Table 206
  - TextItem 212
- hide/show item feature 295
- hideColumn function 206
- hideDetail function
  - Crosstab 45, 335
  - Table 206
- hideNavBar function 103
- hideParameterGroup function 104
- hideParameterNames function 104
- hiding
  - chart elements 155
  - columns 11, 206
  - data 11, 46, 161, 206, 335
  - Data Analyzer features 44
  - Flash objects 165, 169
  - HTML5 charts 187
  - label elements 200
  - navigation bars 103
  - parameters 139
  - report parameters 104, 134
  - table of contents 288
  - tables 206
  - text items 212
  - toolbars 10
- Highchart documentation 193
- Highchart drawing elements 192
- Highchart point configurations 189
- Highchart renderer objects 192, 194, 197
- Highcharts class 191
- Highcharts objects 174, 191
- Highcharts point class 183
- highlighting 295
- hints. *See* tooltips
- hover highlight feature 295
- HTML buttons 27
  - See also* button elements
- HTML code 2, 197
- HTML containers. *See* HTML forms
- HTML elements
  - adding Flash objects to 164, 168
  - adding table elements to 204
  - adding text elements to 199, 211
  - creating cross tabs and 306, 333
  - displaying charts and 153
  - displaying Data Analyzer and 307, 310
  - displaying data and 160, 252
  - displaying parameters and 100
- HTML formats 255
- HTML forms 100, 108
- HTML5 charts
  - accessing 11
  - adding animation feature to 187
  - adding data series to 173, 179, 186
  - applying themes to 33
  - changing data series in 177
  - changing features 11
  - displaying 190
  - drawing functions for 192
  - getting number of run-time series in 174
  - getting options for 174
  - getting series values for 175, 176
  - getting size of 173, 174
  - getting specific instance of 153
  - hiding 187
  - instantiating 172

- HTML5 charts (*continued*)
  - labeling axes values 181, 182
  - pivoting axes values in 180
  - redrawing 176, 188, 189
  - removing data points in 184
  - removing duplicate values for 187
  - removing series from 176, 187, 188
  - rendering 188
  - replacing data series in 188
  - resizing drawing area for 197
  - selecting data points in 184
  - selecting series for 188
  - setting data point properties for 183
  - setting options for 179
  - setting series values for 177, 178
  - setting series visibility in 189
  - setting title of 177, 181
  - setting type 180
  - updating data points in 185
  - viewing data series in 177, 181
- HTTP requests 2
  - See also* requests
- HTTP sessions
  - closing 7
  - initializing 4, 5
  - providing security for 4, 5, 6
  - running Data Analyzer and 350
  - sharing information for 6
  - timing out. *See* ON\_SESSION\_TIMEOUT constant
- HTTPS sessions 6
- hyperlinks 296
  - See also* URLs
- I**
- IBM Advanced Function Printing
  - formats 255
- iHub server URLs 54, 248, 250
- iHub volumes. *See* Encyclopedia volumes
- image elements 195
- image function 195
- IN operator 79, 355
- includeHiddenObject value 240
- inclusive range operators 79, 80
- index values
  - accessing result sets and 91
  - changing axis type and 41, 337
  - creating total objects and 42
  - displaying cross tabs and 333, 334
  - displaying tables and 205
  - getting column 91, 203, 285
  - getting data row 87
  - getting level 365
  - getting measure 369, 370, 397
  - retrieving 344, 346
  - setting data row 23, 88, 89
  - setting dimension 345, 346
  - setting level 365
  - setting measure 372, 398
  - setting parameter position and 141
- Information Console 2, 5
- Information Console URL 59
- initialize function 4, 5, 7, 55
- input 126
- input parameters 2, 90, 112
- interactive features 28, 40, 256
- Interactive mode (data analyzer) 318, 321
- Interactive Viewer
  - disabling 255
  - enabling 256
  - integrating crosstab analyzer with 317, 321, 326
- interactive viewing status 264
- interactivity editor 30
- international reporting applications. *See* localization
- Internet Explorer 31
  - See also* web browsers
- Invoke Script action 31
- iportal URL 4, 54, 59
- iportal web service connection 95
- iportalURL variable 53, 56
- IS\_MULTISHEET constant 280
- isActive function 317
- isAdHoc function 134
- isAdhoc function 123
- isAscending function
  - Data.Sorter 93
  - XTabAnalyzer.Sorter 389
- isAutoSaveEnabled function 64
- isCascadingParameter function 123
- isChartWithAxes function
  - ClientChart 176

- ClientOption 180
- isConnected function 56
- isDashboard function 318
- isDynamicFilter function 123
- isEnabled function 397
- isserverURL parameter 250
- isserverURL variable 53, 56
- isExceptionType function
  - Exception 98
  - XTabAnalyzer.Exception 353
- isHidden function 134
- isInitialized function 57
- isInteractive function
  - Viewer 264
  - XTabAnalyzer 318
- isMultiList function 124
- isPassword function 134
- isRequired function
  - ParameterData 124
  - ParameterDefinition 135
- isSavingNeeded function 65
- isUsingPersonalDashboard function 65
- isViewParameter function
  - ParameterDefinition 135
  - ParameterValue 147
- iterators 24
- IV mode. *See* Interactive mode
- ivMode parameter 321

## J

- Java Components 5
- JavaScript API
  - accessing 2
  - closing sessions for 7
  - designing cross tabs and 306, 308
  - designing reports and 26
  - developing with 2, 48
  - initializing sessions for 4, 5
  - loading dialog boxes and 8
  - loading resources for 6
  - providing security with 4–8
- JavaScript API class libraries 2, 48, 57
- JavaScript API class reference 49, 51, 309
- JavaScript API classes 2, 26, 48, 307
- JavaScript API functions 2, 3, 26, 51, 306
  - See also* callback functions

- JavaScript API usage error constants 111, 221
- JavaScript API usage errors 97, 352
- JavaScript framework 2

## L

- Label class 198
- label elements 198, 261, 276
  - See also* text elements
- label objects 198
- labels (report explorer) 219
- large reports 9
- launch viewer feature 296
- layout type constants (parameters) 102, 109
- LAYOUT\_COLLAPSIBLE event 111
- LAYOUT\_GROUP event 111
- LAYOUT\_NONE constant 111
- LDAP environments 6
- legends (chart) 28
- LESS\_THAN operator 79, 355
- LESS\_THAN\_OR\_EQUAL operator 79, 355
- level attribute names 367
- level attribute objects 367
- level attributes 364
- Level class 39, 364
- level index values 365
- level names
  - getting 357, 365, 374, 393
  - setting 359, 366, 375, 394
- level objects 364
  - See also* cross tabs; data cubes
- LevelAttribute class 367
- libraries 2, 48, 57
- LIKE operator 79, 355
- line charts 181
- linear gauge gadgets 170
- links 296
  - See also* URLs
- list boxes 137
- literal strings 266
- load function
  - actuate 3, 4, 8, 57
  - Dashboard 17
  - DataService 22
  - Parameter 19
  - ReportExplorer 12

- loading
    - class libraries 2
    - cross tabs 306
    - dashboards 17
    - Data Analyzer 306
    - data cubes 306
    - data services 22, 26
    - dialog boxes 8, 26
    - JavaScript API library 48
    - parameter components 19, 26
    - report components 52, 57
    - report content 10
    - report elements 4
    - report viewers 8, 12, 26
    - reports 8
    - resources 6
  - localization
    - converting parameters for 113
    - formatting data for 79
    - getting current locale for 249
    - rendering data for 3
    - setting locale for 250
    - showing data for 110
  - login information 6
  - login pages 5
  - login servlet 6
  - logins, forcing 6
  - logout function 7, 58
- ## M
- main menu 296
  - margins
    - enabling or disabling 292
    - getting Data Analyzer 314, 315
    - getting viewer 258
    - setting Data Analyzer 321, 324
    - setting viewer 265
  - marker lines (charts) 28
  - mashup page 26
  - MATCH operator 80, 355
  - Measure class 39, 41, 368
  - measure index values
    - adding totals and 42
    - changing axis type and 41
    - getting 369, 370, 397
    - setting 372, 398
  - measure names 40, 370, 372
  - measure objects 368
    - See also* measures
  - measureDirection parameter 329, 376
  - measureExpression variable 40
  - measureName variable 40
  - measures
    - See also* cross tabs; cubes
    - adding to cross tabs 40, 329, 368
    - changing direction of 41, 330
    - changing order of 41, 337
    - deleting 41, 337
    - editing 41, 332
    - entering expressions for 371
    - filtering data in 359, 360
    - generating summary values for 42
    - getting aggregate functions for 368
    - getting data type of 369
    - getting direction for 379
    - getting expressions for 369
    - getting index for 369, 370, 397
    - getting level values in 375
    - getting names 370
    - naming 372
    - retrieving 40
    - setting aggregate function for 370
    - setting data type of 371
    - setting direction of 329, 381
    - setting index values for 372, 398
    - sorting values 388
    - viewing data cubes and 37
  - member value objects 44, 373, 374
  - members (cross tabs)
    - defining values for 373
    - drilling through 347, 349
    - getting level names for 374
    - sorting values 389, 390
  - MemberValue class 373
  - memory 126
  - menus 296, 301
  - meta tag 3
  - metadata (dashboards) 66
  - meter gadgets 170
  - mirror column starting level
    - getting 377
    - setting 330, 376, 380

- mirror row starting level
  - getting 379
  - setting 329, 376, 382
- mouse scrolling 282, 283
- move columns feature 296
- multi-clue parameters 112
- multidimensional data structures 37
- multilevel dimensions 42
- multi-list UI elements 124
- multi-select parameters 140
- multisheet render option 280

## N

- namespace 48
- nameValueCollection variable 113
- NameValuePair class 116
- naming report files 228, 241, 264, 265
- NAV\_FIRST constant 111, 221
- NAV\_LAST constant 111, 221
- NAV\_NEXT constant 111, 221
- NAV\_PREV constant 111, 221
- navigate function 104
- navigation bars 103
- navigation buttons 104
- navigation constants 111, 221
- navigation feature (pages) 297
- navigation links 111, 221
- navigator. *See* ReportExplorer components
- next function 24, 91
- NON\_DASHBOARD constant 269
- NOT\_BETWEEN operator 80, 355
- NOT\_EQ operator 80, 355
- NOT\_IN operator 80, 355
- NOT\_LIKE operator 80, 355
- NOT\_MATCH operator 80, 355
- NOT\_NULL operator 80, 355
- NULL operator 80, 355
- null parameter 4, 6, 7
- null values
  - authentication and 53
  - getting 131, 146, 279
  - reserved parameters and 4
  - setting 150, 279, 387
  - specifying as default 138
  - subtotals and 339
  - testing for 80, 355, 386
- numbers 79

## O

- object properties 302, 402
- object types 286, 303
- objects 4, 112, 285, 316
- ON\_CHANGE\_COMPLETED constant 115
- ON\_CHANGED constant 115
- ON\_CONTENT\_CHANGED constant 273, 350
- ON\_CONTENT\_SELECTED constant 273, 350
- ON\_CONTENT\_SELECTED event 285
- ON\_DIALOG\_OK constant 273
- ON\_EXCEPTION constant
  - Dashboard 73
  - Parameter 115
  - ReportExplorer 222
  - Viewer 273
  - XTabAnalyzer 350
- ON\_EXCEPTION event 303
- ON\_SELECTION\_CHANGED constant 115, 222
- ON\_SESSION\_TIMEOUT constant
  - Dashboard 73
  - Parameter 115
  - ReportExplorer 222
  - Viewer 273
  - XTabAnalyzer 350
- online documentation v, 260, 266
- online help. *See* online documentation
- onUnload function
  - Dashboard 65
  - Parameter 105
  - ReportExplorer 216
- opening
  - connections 2, 4, 55
  - dashboards 17
  - reports 8, 15
  - script editor 33
- operations 31
- operator lists 133
- operators (filter expressions) 79, 81, 82, 354
- Options class 376
- output 19
- output formats 255

## P

- page break intervals 378, 379, 380
- page break status 378
- page breaks 297, 378, 381
- page components
  - accessing report elements in 274
  - adding Flash objects to 165, 169
  - adding tables to 204, 208
  - adding text elements to 199, 211
  - creating cross tabs and 313, 334, 383
  - displaying charts and 154, 156, 187
  - displaying data and 160, 161
  - getting content from 10, 259, 313
  - getting current number for 259
  - getting Flash objects in 275, 276
  - linking to 296
  - navigating to specific 111, 221, 264, 297
- page content objects 274, 383
- page counts 225, 228, 262, 315
- page layouts 3
- page navigation constants 111, 221
- page navigation feature 297
- page numbers 259
- page position (viewer) 263
- page ranges 255, 258
- PageContent class 274, 383
- param1 parameter 19
- Parameter class 19, 100
- parameter classes 49
- parameter components 19, 57, 100
  - See also* parameters
- parameter control type UI values 120
- parameter control types 130, 137
- parameter convert utility class 112
- parameter definition names 132
- parameter definition objects 101, 118, 126
- parameter definitions
  - creating 126
  - displaying parameters and 116, 143
  - entering passwords and 134, 140
  - getting autosuggest threshold for 129
  - getting column names for 129
  - getting control type for 130
  - getting data type for 131
  - getting default values for 131
  - getting display names for 130, 132
  - getting help text for 132
  - getting name-value pair for 133
  - getting operator list for 133
  - getting required parameters for 135
  - getting values for 134
  - naming 141
  - setting autosuggest threshold for 135
  - setting column names for 136
  - setting column type for 136
  - setting control type for 137
  - setting data type for 137
  - setting display names for 137, 138
  - setting help text for 139
  - setting multiple values for 140
  - setting name-value pairs for 141
  - setting required parameters for 140
  - specifying data type returned by 130
  - specifying default values for 138
  - storing position of 133, 141
- parameter events 105
- parameter global constants 111
- parameter group names 103, 108
- parameter groups
  - creating 126, 138, 148
  - displaying 108
  - expanding 108
  - hiding parameters in 104
  - returning 132, 145
- parameter index values 141
- parameter layout type constants 102
- parameter lists
  - changing values in 115
  - defining name-value pairs for 116, 141
  - getting autosuggest threshold for 129
  - getting name-value pair for 133
  - getting parameter position in 146
  - setting autosuggest delays for 106
  - setting autosuggest threshold for 135
  - setting column names for 147, 279
  - setting column type for 147
  - setting fetch size of 107
  - setting length of 107
  - setting parameter position in 149
- parameter maps 113
- parameter names
  - getting 121, 129, 145, 278, 385
  - setting 136, 149, 386



- parameter objects 100, 105, 110
  - See also* parameters
- parameter page components 57
- parameter pages
  - changing 104
  - displaying parameters and 100, 111, 126
  - enabling 297
  - getting group names for 103
  - getting layout type for 102
  - hiding navigation bar for 103
  - loading 57
  - navigating through 20, 104, 111
  - rendering content for 100, 106
  - retrieving parameters for 20
  - setting display names for 138
  - setting fonts for 108
  - setting HTML container for 108
  - setting layout of 109
- parameter panels 270
- parameter value objects 143, 278, 385
- ParameterData class 118
- ParameterDefinition class 126
- ParameterDefinition objects 101, 118, 126
  - See also* parameter definitions
- parameters
  - See also* parameter components
  - accessing result sets and 90
  - adding subtotals and 339
  - assigning data types to 137, 148
  - assigning multiple values to 140
  - authenticating web services and 5, 53
  - changing 100, 115, 126
  - converting values for 112
  - customizing 6, 53
  - defining. *See* parameter definitions
  - determining if required 124, 135
  - determining type 123, 134, 135
  - displaying dashboards and 18
  - displaying reports and 8, 12, 15
  - displaying tables and 31
  - downloading 19, 20, 101, 102
  - enabling user interface options and 10
  - filtering data and 79
  - generating query output and 136, 147
  - getting control type for 120
  - getting custom URL 250
  - getting data types for 131, 145
  - getting display names for 145
  - getting file names for 103
  - getting values for 114, 146, 278, 279, 314
  - grouping. *See* parameter groups
  - handling events for 105
  - hiding 104, 134, 139
  - initializing HTTP sessions and 4, 55
  - linking to web services 110
  - localizing 110
  - naming 386
  - prompting for input and 126, 146, 149
  - removing authentication information and 7
  - retrieving data and 2, 23, 118
  - retrieving from reports 19, 20, 26, 109
  - returning Data Analyzer 385
  - running reports and 19, 103, 266, 267
  - selecting 107, 115, 122, 133, 141, 142
  - setting display names for 148
  - setting position of 149
  - setting properties for 113
  - setting values for 112, 143, 150, 279, 387
  - specifying as read-only 109
  - specifying null values for 150, 279, 387
  - submitting requests for 110, 143
  - testing for null values in 386
  - unloading 105
  - viewing 20, 100, 126, 140
- ParameterValue class 143, 278, 385
- paramValues variable 114
- parent parameters 129
- parentname variable 129
- passback variable 2
- password parameter 53
- password variable 7
- passwords 5, 6, 134, 140
- path commands (SVG) 196
- path function 195
- pathname parameter 15
- paths 13, 15, 64, 70
- pattern operators 79, 80
- PDF formats 255
- performance 126
- permissions 226, 229
- personal dashboards 65, 71
- pick lists 122
- pie charts 180, 181

- pivot function 41, 335
- pivotChart function 180
- pixel values (viewer) 259
- PostScript formats 255
- PowerPoint formats 255
- previewing reports 26
- print dialog 270
- printing 270, 297
- private access type 227, 238
- private files 224, 227
- privilege filter objects 244
- privilege filters 237, 241, 244
- PrivilegeFilter class 244
- privileges 226, 229
- processError function 23
- processParameters function 20
- profiles (volume) 250, 251
- prompt text 122
- prompts 126, 146, 149
- properties 113, 291, 302, 402
- Prototype JavaScript Framework 2
- public directories 13

## Q

- queries
  - building data cubes and 37
  - retrieving data and 136, 147
  - running ad hoc 129, 130

## R

- radio buttons 137
- read-only parameters 109
- rect function 196
- rectangles 194, 196
- redo feature 302
- redraw function
  - ClientChart 176
  - ClientSeries 187
- registerEventHandler function
  - Dashboard 65
  - Parameter 105
  - ReportExplorer 216
  - XTabAnalyzer 38, 318
- remove function
  - ClientPoint 184
  - ClientSeries 188
- removeDimension function 40, 335
- removeEventHandler function
  - Dashboard 66
  - Parameter 105
  - ReportExplorer 216
  - XTabAnalyzer 39, 318
- removeGroup function 207
- removeMeasure function 41, 337
- removeSeries function 176
- removing
  - authentication information 7, 58
  - data groups 207
  - data series 176, 187, 188
  - dimensions 40, 335
  - duplicate values 187
  - event handlers 39, 105, 318
  - filters 43
  - measures 41, 337
  - summary data 42
- render function 188
- render options 280
- render options map 280
- renderContent function
  - Dashboard 66
  - Parameter 106
- Renderer class 192
- Renderer objects 192, 194, 197
- rendering
  - cross tabs 326
  - dashboard content 66
  - HTML5 charts 188
  - parameter components 100, 106
  - reports 3, 26, 267
- RenderOptions class 280
- reorderDimension function 41, 336
- reorderMeasure function 41, 337
- report applications. *See* applications
- report classes 10, 50
- report components 52, 57
- report content objects 84
- report design files 19, 109, 267
  - See also* report designs
- report design save feature 298
- report designers 26
- report designs 264, 298, 312
- report document files 103, 109, 267, 323
  - See also* report documents

- report document save feature 298
- report documents 265, 298, 306, 312
  - See also* reports
- report element IDs 160
- report element types 286
- report elements
  - See also* specific type
  - accessing 10, 274
  - assigning bookmarks to 10
  - displaying charts and 151, 172, 191
  - displaying data and 161
  - displaying text and 198, 210
  - embedding code in 12
  - embedding in web pages 2
  - getting bookmarks for 259, 261, 263
  - getting from viewer 259
  - getting type 286
  - handling events for 26
  - handling exceptions for 303, 352
  - loading 4
  - retrieving result sets and 85
  - selecting 285
  - setting bookmarks for 87
  - submitting requests for 200, 213
- report executable files 19
- report explorer component. *See* ReportExplorer components
- report files 103, 214, 233, 261
  - See also* specific report file type
- report gadgets 75
- report items 295
- report names. *See* file names; report titles
- report output formats 255
- report parameters 19, 75, 100
  - See also* parameters
- report template files 63
- report titles 262
- report viewers. *See* viewers
- ReportContent class 84
- ReportContent objects 84
- ReportExplorer class 12, 13, 214
- ReportExplorer components
  - getting file descriptions for 224
  - getting files for 224, 225, 226
  - getting folders for 215
  - getting latest items 215
  - getting results definitions for 215
  - getting timestamps for 226
  - getting user information for 244
  - handling events for 15, 216, 222
  - instantiating 13, 214
  - loading 12, 57
  - navigating through 15
  - retrieving data for 13, 218
  - searching for items for 216, 218, 231, 233
  - selecting items in 15
  - setting container for 217
  - setting file descriptions for 227
  - setting file names for 228
  - setting file type attributes for 13
  - setting file types for 227
  - setting file version for 229
  - setting folder names for 217
  - setting folder paths for 219
  - setting items displayed in 219
  - setting labels for 219
  - setting latest version flag for 217
  - setting results definitions for 218
  - setting target service URL for 218
  - setting timestamps for 229
  - submitting requests for 13, 219
  - viewing report items and 12, 14, 214
- ReportExplorer event constants 222
- ReportExplorer global constants 221
- reporting applications. *See* applications
- reporting services. *See* web services
- Reportlet gadgets 75
- Reportlets 26, 261, 267, 323
- reports
  - accessing content 10, 256
  - adding data items to 159
  - adding Flash objects to 163
  - adding page breaks to 297
  - changing 293
  - counting pages in 225, 228, 262, 315
  - designing 26
  - displaying 8, 19, 22, 252, 274
  - downloading 255
  - embedding code in 12
  - embedding in web pages 2
  - exporting 255, 269, 293
  - generating 103
  - getting content for 84, 160, 258, 259
  - getting parameters for 100

- reports (*continued*)
  - getting user information for 244
  - getting version of 226
  - hiding data in 161
  - navigating through 263, 264, 272, 297
  - opening 8, 15
  - previewing 26
  - printing 270, 297
  - reloading 273
  - rendering 3, 26, 267
  - retrieving content 2, 22, 23
  - retrieving data from 95
  - retrieving parameters from 19, 20, 26, 103, 109
  - retrieving specific pages 255, 258
  - retrieving subsets of data in 11
  - running 4, 266, 267
  - saving 298
  - selecting content in 273, 285
  - setting locale for 250
  - setting version information for 229
  - suppressing duplicate values in 299
  - viewing hidden elements in 166
  - viewing specific parts of 9
  - viewing table of contents for 287, 288, 300
- repositories 5, 13, 214
  - See also* Encyclopedia volumes
- repository access rights 245, 246
- repository file paths 15, 64
- repository types 249, 251
- REPOSITORY\_ENCYCLOPEDIA
  - constant 251
- REPOSITORY\_STANDALONE constant 251
- Request class 23, 85
- request objects 23, 85
- request parameter 23
- RequestOptions class 4, 6, 248
- RequestOptions objects 4, 12, 59, 248
- requests
  - adding filters to 88
  - adding Flash objects and 166, 171
  - adding sorters to 88
  - authenticating 248
  - closing HTTP sessions and 7
  - displaying charts and 157
  - displaying cross tabs and 307
  - displaying dashboards and 18, 71
  - displaying reports and 8, 271
  - displaying tables and 209
  - failing 59
  - getting bookmarks for 86
  - getting options for 54
  - instantiating 23
  - retrieving data and 2, 23, 85, 161, 256
  - retrieving parameters and 19, 21, 110, 143
  - retrieving report elements and 200, 213
  - running Data Analyzer and 326
  - specifying default settings for 4, 6
- required parameters 124, 135, 140
- requiredFileId value 237, 241
- requiredFileName value 237, 241
- reserved parameters 4
- reset function 319
- resetting driller objects 44
- resizeTo function 319
- resizing
  - columns 292
  - Data Analyzer 319, 320, 325
  - rows 298
  - viewers 70, 257, 260, 268
- resources 6
- result set components 24
- result set objects 24, 90, 256
- result sets
  - accessing data in 90
  - creating 90
  - displaying data and 24
  - downloading 23, 95, 256, 270
  - getting content in 91
  - incrementing data rows for 24, 91
  - referencing 90
  - retrieving data for 85, 90, 95
  - sorting values in 88, 92
- ResultSet class 24, 90
- ResultSet objects 24, 90, 256
- role IDs 245, 246
- role names 245, 247
- rollback function 320
- row headings 36
- row index values
  - getting 87
  - retrieving table data and 205, 334
  - setting 23, 88, 89

- row mirror starting level
  - getting 379
  - setting 329, 376, 382
- ROW\_AXIS\_TYPE constant 344
- rowMirrorStartingLevel parameter 329, 376
- rowPageBreakInterval parameter 376
- rows
  - adding to cross tabs 334, 379
  - adding to tables 205
  - displaying summary data in 42
  - getting first 87
  - getting index values for 87
  - getting last 87
  - grouping 27, 205
  - incrementing 24
  - iterating through 91
  - resizing 298
  - retrieving result sets and 24, 88, 91
  - retrieving specific sets of 11, 23, 31, 80
  - setting index values for 23, 88
  - setting page breaks on 379, 381, 382
  - sorting 12
- running
  - Data Analyzer 2, 306
  - JavaScript API library 48
  - report designs 19
  - report executables 19
  - reports 4, 266, 267
  - scripts 34
- runReport function 4
- run-time parameters 135

## S

- sample report document 307
- save function 67
- saveReportDesign function 264
- saveReportDocument function 265
- saving
  - dashboards 64, 65, 67
  - report designs 264, 298
  - report documents 265, 298, 312
  - viewer content 264, 265
- Scalable Vector Graphics 268, 324
- Scalable Vector Graphics elements 192, 193, 195
- scatter charts 181
- script editor 33
- script tag 2, 3, 48
- scripts 3, 26, 34
- scroll bars 9, 272
- scroll controls 282, 283, 284
- scroll panels 282
- scrolling 9, 282, 287
- ScrollPanel class 282
- search conditions 234, 235, 238
- search operations 216, 218, 231, 233
- security 4–8
- security adapters 7
- security role IDs 245, 246
- security role names 245, 247
- select function
  - ClientPoint 184
  - ClientSeries 188
- SelectedContent class 285
- selection lists. *See* autosuggestion lists
- series (charts)
  - adding 173, 179, 186
  - changing 177
  - deleting 176, 187
  - displaying values 28
  - drilling through 152, 153
  - getting number of run-time 174
  - getting values for 175, 176
  - managing 186
  - removing 188
  - replacing 188
  - setting values for 177, 178
  - setting visibility of 177, 181, 189
- series names 31
- series objects 186, 187
- server error constants 111, 221
- server errors 97, 351
- server URLs 54, 248, 250
- servers 5, 6
- services. *See* data services; web services
- serviceurl parameter 6, 7
- session time-out events. *See*
  - ON\_SESSION\_TIMEOUT constant
- sessions. *See* HTTP sessions
- setAccessRights function 246
- setAccessType function
  - File 227
  - FileSearch 238

- setActiveTab function 68
- setAggregationFunction function
  - Measure 370
  - Total 397
- setAscending function
  - Data.Sorter 93
  - XTabAnalyzer.Sorter 390
- setAutoSaveDelay function 68
- setAutoSuggestDelay function 106
- setAutoSuggestFetchSize function 107
- setAutoSuggestListSize function 107
- setAutoSuggestThreshold function 135
- setAxisType function
  - Dimension 344
  - GrandTotal 363
- setBookmark function 87
- setCascadingParentName function 136
- setChartTitle function 155
- setChartType function 180
- setChildData function 124
- setColumnMirrorStartingLevel function 380
- setColumnName function
  - Data.Filter 82
  - Data.Sorter 93
  - parameter.ParameterValue 147
  - ParameterDefinition 136
  - viewer.ParameterValue 279
- setColumnPageBreakInterval function 380
- setColumns function 87
- setColumnType function
  - ParameterDefinition 136
  - ParameterValue 147
- setCondition function 238
- setConditionArray function 238
- setContainer function
  - Dashboard 68
  - Parameter 107
  - ReportExplorer 217
- setContentMarg function 265
- setContentPanel function 288
- setControlType function 137
- setCountLimit function 238
- setCurrentDisplayName function 137
- setCurrentValue function 124
- setCustomParameter function 250
- setDashboardName function 17, 68
- setData function 188
- setDataType function
  - ParameterDefinition 137
  - ParameterValue 148
  - XTabAnalyzer.Measure 371
- setDefaultValue function 137
- setDefaultValuesIsNull function 138
- setDependentFileId function 239
- setDependentFileName function 239
- setDescription function 227
- setDimension function
  - Chart 155
  - Driller 348
- setDimensionName function 344
- setDisplayName function
  - ParameterDefinition 138
  - ParameterValue 148
- setEmptyCellValue function 380
- setEnabled function 398
- setEnablePageBreak function 381
- setExpandedGroups function 108
- setExpression function 40, 371
- setFetchDirection function 239
- setFetchHandle function
  - FileSearch 239
  - FolderItems 243
- setFetchSize function 240
- setField function 232
- setFileType function 227
- setFilters function
  - Chart 156
  - Crosstab 43, 338
  - FlashObject 166
  - Gadget 169
  - Request 87
  - Table 207
- setFilterType function 358
- setFocus function 265
- setFolderName function 13, 217
- setFont function 108
- setGadgetId function 320
- setGadgetType function 170
- setGrantedRoleId function 246
- setGrantedRoleName function 247
- setGrantedUserId function 247
- setGrantedUserName function 247
- setGroup function
  - ParameterDefinition 138

- ParameterValue 148
- setGroupContainer function 108
- setHeight function
  - Dashboard 69
  - Viewer 266
  - XTabAnalyzer 320
- setHelpBase function 266
- setHelpText function 139
- setId function 227
- setIncludeHiddenObject function 240
- setIndex function
  - Dimension 345
  - Level 365
  - Measure 372
- setIsAdHoc function 139
- setIsServerUrl function 250
- setIsHidden function 139
- setIsMultiSelectControl function 140
- setIsPassword function 140
- setIsRequired function 140
- setIsViewParameter function
  - ParameterDefinition 140
  - ParameterValue 148
- setItemList function 243
- setIVMode function 321
- setKey function 390
- setLatestVersionOnly function 217
- setLayout function 109
- setLeft function 321
- setLevelAttributeName function 358
- setLevelName function
  - Filter 359
  - Level 366
  - MemberValue 375
  - Sorter 390
  - SubTotal 394
- setLevels function 345
- setLocale function 250
- setLocation function 394
- setMatch function 232
- setMaxRows function 88
- setMeasureDirection function 381
- setMeasureIndex function 398
- setMeasureName function 372
- setMember function 390
- setMembers function 349
- setMouseScrollingEnabled function 283
- setName function
  - File 228
  - NameValuePair 117
  - ParameterDefinition 141
  - ParameterValue 149, 386
  - XTabAnalyzer.LevelAttribute 367
- setNewAxisType function 345
- setNewIndex function
  - Dimension 346
  - Measure 372
- setOnClosed function 321
- setOperator function
  - Data.Filter 82
  - XTabAnalyzer.Filter 359
- setOption function 280
- setOwner function
  - File 228
  - FileSearch 240
- setPageCount function 228
- setPageNum function 322
- setPanInOutEnabled function 283
- setParameters function 266
- setParameterValues function
  - Viewer 20, 266
  - XTabAnalyzer 322
- setParentData function 125
- setPosition function
  - ParameterDefinition 141
  - ParameterValue 149
  - XTabAnalyzer 322
- setPrivilegeFilter function 241
- setPromptParameter function 149
- setReadOnly function 109
- setReportletBookmark function 9, 267
- setReportletDocumentMode function 323
- setReportName function
  - Parameter 19, 109
  - Viewer 8, 18, 267
  - XTabAnalyzer 306, 323
- setRepositoryType function 251
- setRequiredFileId function 241
- setRequiredFileName function 241
- setResultDef function 218
- setRowMirrorStartingLevel function 382
- setRowPageBreakInterval function 382
- setScrollControlEnabled function 284
- setSearch function 218

- setSelectNameValueList function 141
- setSelectValueList function 142
- setSeriesVisible function
  - ClientChart 177
  - ClientOption 181
- setService function
  - Dashboard 69
  - Parameter 110
  - ReportExplorer 218
  - Viewer 268
  - XTabAnalyzer 323
- setShowDisplayType function 110
- setShowToc function 288
- setSize function
  - Chart 156
  - Dashboard 69
  - File 228
  - Gadget 170
  - HTML5Chart Renderer 197
  - Viewer 268
- setSorters function
  - Crosstab 43, 338
  - Request 88
  - Table 207
- setStartingFolder function 219
- setStartRow function 88
- setSubType function 157
- setSupportSVG function
  - Viewer 268
  - XTabAnalyzer 324
- setTemplate function 70
- setTimeStamp function 229
- setTitle function
  - ClientChart 177
  - ClientOption 181
- setTop function 324
- setTotalCount function 243
- setTotals function
  - Crosstab 339
  - GrandTotal 363
  - SubTotal 395
- setUIOptions function
  - Viewer 10, 269
  - XTabAnalyzer 324
- setUseDescriptionAsLabel function 219
- setUserPermissions function 229
- setValue function
  - NameValuePair 117
  - ParameterValue 149
  - viewer.ParameterValue 279
  - XTabAnalyzer.MemberValue 375
  - XTabAnalyzer.ParameterValue 387
- setValueIsNull function
  - ParameterValue 150
  - viewer 279
  - XTabAnalyzer.ParameterValue 387
- setValues function
  - ClientChart 177
  - Data.Filter 83
  - XTabAnalyzer.Filter 360
- setVersion function 229
- setVersionName function 229
- setViewingMode function 269
- setVisible function 189
- setVolume function 251
- setVolumeProfile function 251
- setWebService function 125
- setWidth function
  - Dashboard 70
  - Viewer 269
  - XTabAnalyzer 325
- setXAxisRange function 178
- setXAxisTitle function 181
- setXTabBookmark function 325
- setXTabId function 325
- setYAxisRange function 178
- setYAxisTitle function 182
- shared access type 227, 238
- show function
  - Chart 157
  - ClientSeries 190
  - DataItem 161
  - FlashObject 166
  - Gadget 171
  - Label 200
  - Table 208
  - TextItem 212
- showColumn function 208
- showDetail function
  - Crosstab 45, 339
  - Table 208
- showDownloadReportDialog function 269



- showDownloadResultSetDialog function 270
- showFacebookCommentPanel function 270
- showFoldersOnly function 219
- showGallery function 70
- showParameterPanel function 270
- showPrintDialog function 270
- showTabNavigation function 70
- showToc flag 287, 288
- showTocPanel function 270
- simple object access protocol. *See* SOAP
- single sign-on authentication 7
- SOAP message error descriptions 97, 351
- SOAP messages 125
- sort conditions 92, 388
- sort feature 290, 299
- sort keys 388, 390
- sort order 92, 93, 389, 390
- Sorter class 43, 92, 388
- sorter object arrays 88, 338
- sorter objects 43, 88, 92, 388
- sorters
  - creating 43, 92, 388
  - getting column names for 92
  - sending requests for 87, 88
  - setting column names for 93
  - setting sort order for 93, 390
  - sorting data and 12, 43
  - specifying specific tables for 207
  - testing sort order for 93
  - testing sort order in 389
- sorting data 12, 43, 88
- sortTable function 12
- source code
  - accessing 2
  - adding chart interactive features and 30
  - constructing requests and 23
  - displaying cross tabs and 39, 40, 43
  - displaying dashboards and 18
  - displaying parameters and 20, 21
  - displaying reports and 10
  - embedding 12
  - enabling user interface options and 10
  - hiding user interface options and 44
  - initializing HTTP sessions and 4
  - initializing HTTPS sessions and 6
  - registering event handlers and 38
  - running 27
- source files 2
- spark gadgets 170
- spreadsheet output formats 255
- spreadsheets 280
- SQL statements. *See* queries
- standard charts. *See* charts
- standards compliance mode 3
- start parameter 23
- strict.dtd value 3
- string pattern operators 79, 80
- strings 3, 231, 266
- subclasses 2
- submit function
  - actions and 31
  - Chart 157
  - Crosstab 340
  - Dashboard 18, 71
  - DataItem 161
  - FlashObject 166
  - Gadget 171
  - Label 200
  - Parameter 19, 110
  - ReportExplorer 219
  - Table 209
  - TextItem 212
  - Viewer 8, 10, 271
  - XTabAnalyzer 326
- submitCallback function 18
- SubTotal class 42, 392
- subtotals 42, 339, 392, 395
- suggestion lists. *See* autosuggestion lists
- summary data
  - See also* totals
  - adding 36, 37, 42
  - deleting 42
  - generating grand totals and 42, 361
  - generating subtotals and 42, 339, 392
  - getting aggregate function for 396
  - getting level names for 393
  - getting location name for 393
  - getting type names for 394
  - setting aggregation function for 397
  - setting level names for 394
  - setting location of 395
- SVG elements 192, 193, 195
- SVG flag 268, 324

SVG path commands 196  
swapColumns function 209  
switch view feature 299

## T

Tab class 77  
tab objects 77  
    *See also* dashboard tabs  
tab type constants 77  
table bookmarks 31, 203  
Table class 10, 202  
table elements 202  
table headers 28  
table names 203, 204  
table objects 202  
table of contents 287, 288, 300  
table of contents panel 270  
tables  
    *See also* cross tabs  
    changing 10  
    displaying 10, 31, 202  
    filtering data in 30, 203, 207  
    getting columns in 203  
    getting instance ID of 204  
    getting rows for 205  
    grouping rows in 27, 205  
    hiding 206  
    hiding data in 11, 206, 335  
    removing groups from 207  
    reordering columns in 209, 296, 298  
    resizing columns in 292  
    resizing rows in 298  
    retrieving 31, 261, 276  
    showing columns in 208  
    showing groups in 208  
    sorting data in 12  
    submitting requests for 209  
    suppressing duplicate values in 299  
tabs. *See* dashboard tabs  
target service URL 218, 268  
template files 63  
template paths 70  
temporary files 103  
testing  
    connections 56  
    scripts 26

text 84, 199, 212, 262  
    *See also* text elements; text items  
Text class 10  
text editing feature 300  
text elements 10, 210, 212, 277  
    *See also* label elements  
text function 197  
text graphic elements 197  
text item objects 210  
text items 210, 211, 212  
    *See also* text elements  
text objects 262  
TextItem class 210  
thermometer gadgets 170  
this keyword 12  
3D charts 155  
time dimensions (cubes) 37  
time stamps 226, 229  
time values 79  
titles  
    charts 152, 153, 155, 156  
    reports 262  
Tomcat servers 6  
    *See also* application servers  
toolbar help feature 301, 401  
toolbar save feature 401, 402  
toolbars 10, 300, 400  
tooltips 28, 299  
top N filter feature 302  
TOP\_N operator 80, 355  
TOP\_PERCENT operator 80, 355  
Total class 42, 396  
total objects 42, 394, 396  
totals  
    *See also* summary data  
    adding grand totals and 361, 363  
    adding to subtotals 339, 392, 395  
    enabling or disabling 397, 398  
    getting 362, 394  
    returning axis type for 362  
    returning index values for 397  
    returning type 363  
    setting axis type for 363  
    setting index values for 398  
    viewing in charts 28  
    viewing in cross tabs 42, 396

transient files 103  
TRUE operator 80, 355

## U

UI configuration objects 262, 287  
UI elements 124  
UI options. *See* user interface options  
UIConfig class 287  
UIOptions class  
    Viewer 10, 289  
    XTabAnalyzer 44, 399  
uncategorized exceptions 97  
undo feature 302  
update function 185  
updating data values 44, 124  
URL parameters 4, 6, 53, 110, 250  
URLs  
    accessing JavaScript library and 48  
    connecting to web services and 6, 7, 61, 218, 268, 323  
    displaying help documentation and 260, 266  
    getting iHub server 248  
    initializing HTTP sessions and 4  
    retrieving parameters and 110  
    returning default web service 54  
    returning failed requests for 59  
    returning from exception objects 59  
    setting iHub server 250  
    testing connections for 56  
    unloading authentication information and 7  
usage error constants 111, 221  
usage errors 97, 352  
usePersonalDashboard function 71  
user authentication. *See* authentication  
user credentials 6, 53  
user IDs  
    authentication and 6, 53, 60  
    privilege filters and 245, 247  
user information 244  
user interface configuration objects 262, 287  
user interface elements 124  
user interface options  
    changing 10  
    enabling 10, 289

    getting 262, 315  
    hiding 45  
    setting Data Analyzer 324, 399  
    setting viewer 269  
user interfaces 10, 44, 214  
user logins, forcing 6  
user names 5, 246, 247  
user permissions. *See* privileges  
userID parameter 6, 53  
userid variable 53, 56  
username variable 7  
userpassword parameter 6  
UTF8 character encoding 3

## V

value parameter 31  
value series (charts) 28, 155, 175, 176, 178  
    *See also* y-axis values  
valueData variable 31  
valueIsNull value 146, 279  
values  
    aggregating 36, 37, 42, 291  
    changing parameter 115, 126  
    converting 112  
    downloading parameters and 102  
    filtering cross tabs and 354  
    filtering locale-specific data and 79  
    filtering top or bottom 302  
    generating summary 42  
    getting default 131  
    getting empty cell 378  
    getting level 375  
    getting parameter 114, 146, 278, 279, 314  
    getting series 175, 176  
    matching set of 79, 80  
    matching top or bottom 79  
    removing duplicate 187  
    returning specific 91, 160  
    selecting parameters and 134, 140  
    setting default 138  
    setting display names for 116  
    setting empty cell 330, 380  
    setting level 375  
    setting parameter 112, 143, 150, 279, 387  
    setting series 177  
    specifying null 150, 279, 387

- values (*continued*)
  - suppressing duplicate 299
  - testing for null 80, 355, 386
  - updating 44, 124
- valueSeriesName variable 31
- variables 2, 105, 216
- Vector Markup Language graphics
  - elements 192, 195, 196
- version information 226, 229
- version names 229
- view parameter 140
- Viewer class 8, 19, 26, 252
- viewer classes 10, 51
- viewer components 8, 57, 252
  - See also* viewers
- viewer event constants 273
- viewer IDs 277, 384
- viewer objects 252
  - See also* viewers
- viewer variable 308
- viewer1 parameter 8
- ViewerException class 303
- viewers
  - accessing report content for 10, 256
  - adding interactive features to 28
  - building user interface for 10, 214
  - determining status of 264
  - disabling 255
  - displaying charts in 31, 257, 274
  - displaying cross tabs and. *See* Data Analyzer
  - displaying parameters in 20
  - displaying reports in 8, 15, 19, 252
  - enabling interactive features for 256, 287, 289
  - getting browser size for 257
  - getting content for 11, 256, 258, 259
  - getting file names for 261
  - getting margins for 258
  - getting size 260, 263
  - getting specific instance 55, 263
  - getting UI options for 262
  - handling events for 273
  - handling exceptions for 303
  - instantiating 8
  - launching 296
  - loading 8, 12, 26
  - reloading 271
  - resizing 70, 257, 260, 268
  - saving contents 264, 265
  - scrolling in 272, 282, 287
  - selecting content in 285
  - sessions timing out and 350
  - setting focus for 265
  - setting margins for 265
  - setting size 266, 268, 269
  - setting UI options for 269
  - showing main menu in 296
  - showing toolbar help in 301
  - showing toolbars in 300
  - submitting requests for 271
  - switching views 299
- viewing
  - aggregate values 36
  - charts 28, 153, 157
  - columns 208
  - cross tabs 36, 39, 306, 383
  - dashboards 17
  - data 24, 160, 161, 376
  - Data Analyzer features 44
  - data cubes 37, 400
  - data series 177, 181, 189
  - Flash objects 163, 166, 171
  - HTML5 charts 190
  - label elements 200, 261
  - parameter groups 108
  - report elements 10
  - report parameters 20, 100, 126
  - Reportlets 261, 267
  - reports 8, 19, 22, 252, 274
  - summary data 42, 361, 392, 396
  - table elements 208
  - table of contents 287, 288, 300
  - tables 10, 31, 202
  - text 211, 212, 262
  - toolbars 300, 400
  - tooltips 299
  - viewing mode (dashboards) 269
  - viewing mode constants 269
  - views, switching 299
  - view-time parameters 135, 147, 148
  - VML graphics elements 192, 195, 196
  - volume names 53, 56, 249
    - See also* Encyclopedia volumes

volume profiles 250, 251  
volume variable 53, 56

## W

web applications. *See* applications  
web browser content panels 272  
web browser exceptions 97  
web browser windows 257  
web browsers 3, 31, 97, 192  
web pages  
    accessing class libraries for 2, 48  
    adding interactive features to 40  
    adding JavaScript functions to 3  
    adding report components to 52  
    customizing 2  
    displaying cross tabs in 306  
    displaying dashboards in 17  
    displaying reports in 8, 19, 22, 252  
    embedding report parameters in 19, 21  
    embedding reports in 2, 3  
    enabling SVG support for 268, 324  
    initializing HTTP sessions for 4  
    integrating with reporting services 5  
    loading report components for 57  
    retrieving data for 2, 22, 23, 84  
web service applications. *See* applications  
web service connections 95  
web services  
    authenticating users for 4, 5  
    closing connections to 58  
    displaying dashboards and 69  
    encapsulating data for 2  
    getting default URL for 54  
    initializing connections for 4  
    integrating web pages with 5  
    linking parameters to 110

    opening connections for 2, 55  
    providing secure sessions for 6  
    retrieving data from 2  
    sending SOAP messages over 125  
    setting URLs for 218, 268, 323  
Word document output formats 255  
workgroup repository type 251  
worksheets 280

## X

x-axis labels (charts) 181  
x-axis values (charts)  
    adding interactive features to 28  
    returning maximum value for 175  
    returning minimum value for 175  
    setting data points for 189  
    setting range for 178  
XTabAnalyzer class 38, 306, 310  
xtabAnalyzer components 57  
    *See also* Data Analyzer  
XTabAnalyzer event constants 350  
XTabAnalyzer exception objects 351

## Y

y-axis labels (charts) 182  
y-axis values (charts)  
    adding interactive features to 28  
    converting chart dimensions and 155  
    returning maximum value for 175  
    returning minimum value for 176  
    setting data points for 189  
    setting value range for 178

## Z

zooming 283

